Sudeep Pasricha
Muhammad Shafique *Editors*

Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing

Use Cases and Emerging Challenges

Springer

Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing

Sudeep Pasricha • Muhammad Shafique
Editors

Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing

Use Cases and Emerging Challenges

Springer

Editors
Sudeep Pasricha
Colorado State University
Fort Collins, CO, USA

Muhammad Shafique
New York University Abu Dhabi
Abu Dhabi, Abu Dhabi
United Arab Emirates

ISBN 978-3-031-40676-8 ISBN 978-3-031-40677-5 (eBook)
https://doi.org/10.1007/978-3-031-40677-5

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Paper in this product is recyclable.

Preface

Machine Learning (ML) has emerged as a prominent approach for achieving state-of-the-art accuracy for many data analytic applications, ranging from computer vision (e.g., classification, segmentation, and object detection in images and video), speech recognition, language translation, healthcare diagnostics, robotics, and autonomous vehicles to business and financial analysis. The driving force of the ML success is the advent of Neural Network (NN) algorithms, such as Deep Neural Networks (DNNs)/Deep Learning (DL) and Spiking Neural Networks (SNNs), with support from today's evolving computing landscape to better exploit data and thread-level parallelism with ML accelerators.

Current trends show an immense interest in attaining the powerful abilities of NN algorithms for solving ML tasks using embedded systems with limited compute and memory resources, i.e., so-called *Embedded ML*. One of the main reasons is that embedded ML systems may enable a wide range of applications, especially the ones with tight memory and power/energy constraints, such as mobile systems, Internet of Things (IoT), edge computing, and cyber-physical applications. Furthermore, embedded ML systems can also improve the quality of service (e.g., personalized systems) and privacy as compared to centralized ML systems (e.g., based on cloud computing). However, state-of-the-art NN-based ML algorithms are costly in terms of memory sizes and power/energy consumption, thereby making it difficult to enable embedded ML systems.

This book consists of three volumes, and explores and identifies the most challenging issues that hinder the implementation of embedded ML systems. These issues arise from the fact that, to achieve better accuracy, the development of NN algorithms have led to state-of-the-art models with higher complexity with respect to model sizes and operations, the implications of which are discussed below:

- *Massive Model Sizes*: Larger NN models usually obtain higher accuracy than the smaller ones because they have a larger number of NN parameters that can learn the features from the training dataset better. However, a huge number of parameters may not be fully stored on-chip, hence requiring large-sized off-chip memory to store them and intensive off-chip memory accesses during run time.

Furthermore, these intensive off-chip accesses are significantly more expensive in terms of latency and energy than on-chip operations, hence exacerbating the overall system energy.
- *Complex and Intensive Operations*: The complexity of operations in NN algorithms depends on the computational model and the network architecture. For instance, DNNs and SNNs have different complexity of operations since DNNs typically employ Multiply-and-Accumulate (MAC) while SNNs employ more bio-plausible operations like Leaky-Integrate-and-Fire (LIF). Besides, more complex neural architectures (e.g., residual networks) may require additional operations to accommodate the architectural variations. These complex architectures with a huge number of parameters also lead to intensive neural operations (e.g., a large number of MAC operations in DNNs), thereby requiring high computing power/energy during model execution.

In summary, achieving acceptable accuracy for the given ML applications while meeting the latency, memory, and power/energy constraints of the embedded ML systems is not a trivial task.

To address these challenges, this book discusses potential solutions from multiple design aspects, presents multiple applications that can benefit from embedded ML systems, and discusses the security, privacy, and robustness aspects of embedded ML systems. To provide a comprehensive coverage of all these different topics, which are crucial for designing and deploying embedded ML for real-world applications, this book is partitioned into three volumes. The first volume covers the *Hardware Architectures*, the second volume covers *Software Optimizations and Hardware/Software Codesign*, and the third volume presents different *Use Cases and Emerging Challenges*.

The brief outline of the third volume of this Embedded ML book targeting *Use Cases and Emerging Challenges* along with the section structure is as follows.

Part I – Mobile, IoT, and Edge Applications: The first part of the Volume 3 of this book elucidates various applications that benefit from embedded ML systems, including applications for mobile, IoT, and edge computing.

- Chapter 1 explains how to employ CNNs for efficient indoor navigation on battery-powered smartphones while leveraging WiFi signatures.
- Chapter 2 discusses a framework for performing an end-to-end NAS and model compression for healthcare applications on embedded systems.
- Chapter 3 describes the challenges and opportunities of robust ML for power-constrained wearable device applications, such as health monitoring, rehabilitation, and fitness.
- Chapter 4 highlights techniques to design the vision of Unmanned Aerial Vehicles (UAVs) for aerial visual understanding, including data selection, NN design, and model optimization.
- Chapter 5 presents optimization techniques for multi-modal ML-based healthcare applications, including an exploration of accuracy-performance trade-offs.

- Chapter 6 provides a comprehensive survey of embedded ML systems for enabling smart and sustainable healthcare applications.
- Chapter 7 proposes a middleware framework that uses reinforcement learning to decide if the processing should be performed in local or offload processing mode.

Part II – Cyber-Physical Applications: The second part of the Volume 3 of this book presents examples of cyber-physical applications that benefit from embedded ML systems.

- Chapter 8 discusses an adaptive context-aware anomaly detection method for fog computing by employing Long-Short Term Memory (LSTM)-based NNs and Gaussian estimator.
- Chapter 9 explores different ML algorithms to perform various tasks in Autonomous Cyber-Physical Systems, such as robotic vision and robotic planning.
- Chapter 10 presents a framework for efficient ML-based perception with Advanced Driver Assistance Systems (ADAS) for automotive cyber-physical systems.
- Chapter 11 describes a DL-based anomaly detection framework in automotive cyber-physical systems by utilizing a Gated Recurrent Unit (GRU)-based recurrent autoencoder network.
- Chapter 12 discusses an embedded system architecture for infrastructure inspection using UAVs based on ML algorithms.

Part III – Security, Privacy, and Robustness of Embedded ML: Embedded ML systems should be trustworthy to produce correct outputs without any privacy leaks. Otherwise, their processing may lead to wrong outputs, undesired behavior, and data leakage. To address this, the third part of the Volume 3 of this book presents techniques for mitigating security and privacy threats, and improving the robustness of embedded ML systems.

- Chapter 13 discusses the vulnerability of deep reinforcement learning against backdoor attacks in autonomous vehicles.
- Chapter 14 analyzes the vulnerability of a CNN-based indoor localization on embedded devices against access point attacks, then proposes a methodology for mitigating the attacks.
- Chapter 15 studies the impact of noise in the data input on the DNN accuracy, then provides a suitable framework for analyzing the impact of noise on DNN properties.
- Chapter 16 proposes techniques for mitigating backdoor attacks on DNNs by employing two off-line novelty detection models to collect samples that are potentially poisoned.
- Chapter 17 highlights the robustness of DNN acceleration on analog crossbar-based IMC against adversarial attacks and discusses energy-efficient attack mitigation techniques.

- Chapter 18 provides an overview of adversarial attacks and security threats on ML algorithms for edge computing, including DNNs, CapsNets, and SNNs.
- Chapter 19 investigates different challenges for achieving trustworthy embedded ML systems, including robustness to errors, security against attacks, and privacy protection.
- Chapter 20 presents a systematic evaluation of backdoor attacks on DL-based systems in various scenarios, such as image, sound, text, and graph analytics domains.
- Chapter 21 discusses different error-resilience characteristics of DNN models and leverages these intrinsic characteristics for mitigating reliability threats in DL-based systems.

We hope this book provides a comprehensive review and useful information on the recent advances in embedded machine learning for cyber-physical, IoT, and edge computing applications.

Fort Collins, CO, USA — Sudeep Pasricha
Abu Dhabi, UAE — Muhammad Shafique
September 1, 2023

Acknowledgments

This book would not be possible without the contributions of many researchers and experts in the field of embedded systems, machine learning, IoT, edge platforms, and cyber-physical systems. We would like to gratefully acknowledge the contributions of Rachmad Putra (Technische Universität Wien), Muhammad Abdullah Hanif (New York University, Abu Dhabi), Febin Sunny (Colorado State University), Asif Mirza (Colorado State University), Mahdi Nikdast (Colorado State University), Ishan Thakkar (University of Kentucky), Maarten Molendijk (Eindhoven University of Technology), Floran de Putter (Eindhoven University of Technology), Henk Corporaal (Eindhoven University of Technology), Salim Ullah (Technische Universität Dresden), Siva Satyendra Sahoo (Technische Universität Dresden), Akash Kumar (Technische Universität Dresden), Arnab Raha (Intel), Raymond Sung (Intel), Soumendu Ghosh (Purdue University), Praveen Kumar Gupta (Intel), Deepak Mathaikutty (Intel), Umer I. Cheema (Intel), Kevin Hyland (Intel), Cormac Brick (Intel), Vijay Raghunathan (Purdue University), Gokul Krishnan (Arizona State University), Sumit K. Mandal (Arizona State University), Chaitali Chakrabarti (Arizona State University), Jae-sun Seo (Arizona State University), Yu Cao (Arizona State University), Umit Y. Ogras (University of Wisconsin, Madison), Ahmet Inci (University of Texas, Austin), Mehmet Meric Isgenc (University of Texas, Austin), and Diana Marculescu (University of Texas, Austin), Rehan Ahmed (National University of Sciences and Technology, Islamabad), Muhammad Zuhaib Akbar (National University of Sciences and Technology, Islamabad), Lois Orosa (ETH Zürich, Skanda Koppula (ETH Zürich), Konstantinos Kanellopoulos (ETH Zürich), A. Giray Yağlikçi (ETH Zürich), Onur Mutlu (ETH Zürich), Saideep Tiku (Colorado State University), Liping Wang (Colorado State University), Xiaofan Zhang (University of Illinois Urbana-Champaign), Yao Chen (University of Illinois Urbana-Champaign), Cong Hao (University of Illinois Urbana-Champaign), Sitao Huang (University of Illinois Urbana-Champaign), Yuhong Li (University of Illinois Urbana-Champaign), Deming Chen (University of Illinois Urbana-Champaign), Alexander Wendt (Technische Universität Wien), Horst Possegger (Technische Universität Graz), Matthias Bittner (Technische Universität Wien), Daniel Schnoell (Technische Universität Wien), Matthias Wess (Technische Universität Wien),

Dušan Malić (Technische Universität Graz), Horst Bischof (Technische Universität Graz), Axel Jantsch (Technische Universität Wien), Floran de Putter (Eindhoven University of Technology), Alberto Marchisio (Technische Universitat Wien), Fan Chen (Indiana University Bloomington), Lakshmi Varshika Mirtinti (Drexel University), Anup Das (Drexel University), Supreeth Mysore Shivanandamurthy (University of Kentucky), Sayed Ahmad Salehi (University of Kentucky), Biresh Kumar Joardar (University of Houston), Janardhan Rao Doppa (Washington State University), Partha Pratim Pande (Washington State University), Georgios Zervakis (Karlsruhe Institute of Technology), Mehdi B. Tahoori (Karlsruhe Institute of Technology), Jörg Henkel (Karlsruhe Institute of Technology), Zheyu Yan (University of Notre Dame), Qing Lu (University of Notre Dame), Weiwen Jiang (George Mason University), Lei Yang (University of New Mexico), X. Sharon Hu (University of Notre Dame), Jingtong Hu (University of Pittsburgh), Yiyu Shi (University of Notre Dame), Beatrice Bussolino (Politecnico di Torino), Alessio Colucci (Technische Universität Wien), Vojtech Mrazek (Brno University of Technology), Maurizio Martina (Politecnico di Torino), Guido Masera (Politecnico di Torino), Ji Lin (Massachusetts Institute of Technology), Wei-Ming Chen (Massachusetts Institute of Technology), Song Han (Massachusetts Institute of Technology), Yawen Wu (University of Pittsburgh), Yue Tang (University of Pittsburgh), Dewen Zeng (University of Notre Dame), Xinyi Zhang (University of Pittsburgh), Peipei Zhou (University of Pittsburgh), Ehsan Aghapour (University of Amsterdam), Yujie Zhang (National University of Singapore), Anuj Pathania (University of Amsterdam), Tulika Mitra (National University of Singapore), Hiroki Matsutani (Keio University), Keisuke Sugiura (Keio University), Soonhoi Ha (Seoul National University), Donghyun Kang (Seoul National University), Ayush Mittal (Colorado State University), Bharath Srinivas Prabakaran (Technische Universität Wien), Ganapati Bhat (Washington State University), Dina Hussein (Washington State University), Nuzhat Yamin (Washington State University), Rafael Makrigiorgis (University of Cyprus), Shahid Siddiqui (University of Cyprus), Christos Kyrkou (University of Cyprus), Panayiotis Kolios (University of Cyprus), Theocharis Theocharides (University of Cyprus), Anil Kanduri (University of Turku), Sina Shahhosseini (University of California, Irvine), Emad Kasaeyan Naeini (University of California, Irvine), Hamidreza Alikhani (University of California, Irvine), Pasi Liljeberg (University of Turku), Nikil Dutt (University of California, Irvine), Amir M. Rahmani (University of California, Irvine), Sizhe An (University of Wisconsin-Madison), Yigit Tuncel (University of Wisconsin-Madison), Toygun Basaklar (University of Wisconsin-Madison), Aditya Khune (Colorado State University), Rozhin Yasaei (University of California, Irvine), Mohammad Abdullah Al Faruque (University of California, Irvine), Kruttidipta Samal (University of Nebraska, Lincoln), Marilyn Wolf (University of Nebraska, Lincoln), Joydeep Dey (Colorado State University), Vipin Kumar Kukkala (Colorado State University), Sooryaa Vignesh Thiruloga (Colorado State University), Marios Pafitis (University of Cyprus), Antonis Savva (University of Cyprus), Yue Wang (New York University), Esha Sarkar (New York University), Saif Eddin Jabari (New York University Abu Dhabi), Michail Maniatakos (New York University Abu Dhabi), Mahum Naseer (Technische Universität Wien), Iram

Tariq Bhatti (National University of Sciences and Technology, Islamabad), Osman Hasan (National University of Sciences and Technology, Islamabad), Hao Fu (New York University), Alireza Sarmadi (New York University), Prashanth Krishnamurthy (New York University), Siddharth Garg (New York University), Farshad Khorrami (New York University), Priyadarshini Panda (Yale University), Abhiroop Bhattacharjee (Yale University), Abhishek Moitra (Yale University), Ihsen Alouani (Queen's University Belfast), Stefanos Koffas (Delft University of Technology), Behrad Tajalli (Radboud University), Jing Xu (Delft University of Technology), Mauro Conti (University of Padua), and Stjepan Picek (Radboud University).

This work was partially supported by the National Science Foundation (NSF) grants CCF-1302693, CCF-1813370, and CNS-2132385; by the NYUAD Center for Interacting Urban Networks (CITIES), funded by Tamkeen under the NYUAD Research Institute Award CG001, Center for Cyber Security (CCS), funded by Tamkeen under the NYUAD Research Institute Award G1104, and Center for Artificial Intelligence and Robotics (CAIR), funded by Tamkeen under the NYUAD Research Institute Award CG010; and by the project "eDLAuto: An Automated Framework for Energy-Efficient Embedded Deep Learning in Autonomous Systems," funded by the NYUAD Research Enhancement Fund (REF). The opinions, findings, conclusions, or recommendations presented in this book are those of the authors and do not necessarily reflect the views of the National Science Foundation and other funding agencies.

Contents

Part I Mobile, IoT, and Edge Application Use-Cases for Embedded Machine Learning

Convolutional Neural Networks for Efficient Indoor Navigation with Smartphones 3
Saideep Tiku, Ayush Mittal, and Sudeep Pasricha

An End-to-End Embedded Neural Architecture Search and Model Compression Framework for Healthcare Applications and Use-Cases 21
Bharath Srinivas Prabakaran and Muhammad Shafique

Robust Machine Learning for Low-Power Wearable Devices: Challenges and Opportunities 45
Ganapati Bhat, Dina Hussein, and Nuzhat Yamin

Efficient Deep Vision for Aerial Visual Understanding 73
Rafael Makrigiorgis, Shahid Siddiqui, Christos Kyrkou, Panayiotis Kolios, and Theocharis Theocharides

Edge-Centric Optimization of Multi-modal ML-Driven eHealth Applications 95
Anil Kanduri, Sina Shahhosseini, Emad Kasaeyan Naeini, Hamidreza Alikhani, Pasi Liljeberg, Nikil Dutt, and Amir M. Rahmani

A Survey of Embedded Machine Learning for Smart and Sustainable Healthcare Applications 127
Sizhe An, Yigit Tuncel, Toygun Basaklar, and Umit Y. Ogras

Reinforcement Learning for Energy-Efficient Cloud Offloading of Mobile Embedded Applications 151
Aditya Khune and Sudeep Pasricha

Part II Cyber-Physical Application Use-Cases for Embedded Machine Learning

Context-Aware Adaptive Anomaly Detection in IoT Systems 177
Rozhin Yasaei and Mohammad Abdullah Al Faruque

Machine Learning Components for Autonomous Navigation Systems 201
Kruttidipta Samal and Marilyn Wolf

Machine Learning for Efficient Perception in Automotive Cyber-Physical Systems 233
Joydeep Dey and Sudeep Pasricha

Machine Learning for Anomaly Detection in Automotive Cyber-Physical Systems 253
Vipin Kumar Kukkala, Sooryaa Vignesh Thiruloga, and Sudeep Pasricha

MELETI: A Machine-Learning-Based Embedded System Architecture for Infrastructure Inspection with UAVs..... 285
Marios Pafitis, Antonis Savva, Christos Kyrkou, Panayiotis Kolios, and Theocharis Theocharides

Part III Security, Privacy and Robustness for Embedded Machine Learning

On the Vulnerability of Deep Reinforcement Learning to Backdoor Attacks in Autonomous Vehicles 315
Yue Wang, Esha Sarkar, Saif Eddin Jabari, and Michail Maniatakos

Secure Indoor Localization on Embedded Devices with Machine Learning..... 343
Saideep Tiku and Sudeep Pasricha

Considering the Impact of Noise on Machine Learning Accuracy 377
Mahum Naseer, Iram Tariq Bhatti, Osman Hasan, and Muhammad Shafique

Mitigating Backdoor Attacks on Deep Neural Networks..... 395
Hao Fu, Alireza Sarmadi, Prashanth Krishnamurthy, Siddharth Garg, and Farshad Khorrami

Robustness for Embedded Machine Learning Using In-Memory Computing 433
Priyadarshini Panda, Abhiroop Bhattacharjee, and Abhishek Moitra

Adversarial ML for DNNs, CapsNets, and SNNs at the Edge..... 463
Alberto Marchisio, Muhammad Abdullah Hanif, and Muhammad Shafique

On the Challenge of Hardware Errors, Adversarial Attacks and Privacy Leakage for Embedded Machine Learning 497
Ihsen Alouani

A Systematic Evaluation of Backdoor Attacks in Various Domains 519
Stefanos Koffas, Behrad Tajalli, Jing Xu, Mauro Conti, and Stjepan Picek

Deep Learning Reliability: Towards Mitigating Reliability Threats in Deep Learning Systems by Exploiting Intrinsic Characteristics of DNNs 553
Muhammad Abdullah Hanif and Muhammad Shafique

Index 569

Part I
Mobile, IoT, and Edge Application Use-Cases for Embedded Machine Learning

Convolutional Neural Networks for Efficient Indoor Navigation with Smartphones

Saideep Tiku, Ayush Mittal, and Sudeep Pasricha

1 Introduction

Contemporary outdoor location-based services have transformed how people navigate, travel, and interact with their surroundings. Emerging indoor localization techniques have the potential to extend this outdoor experience across indoor locales. Beyond academics, many privately funded providers in the industry are focusing on indoor location-based services to improve customer experience. For instance, Google can suggest products to its users through targeted indoor location-based advertisements [1]. Stores such as Target in the United States are beginning to provide indoor localization solutions to help customers locate products in a store and find their way to these products [2]. Services provided by these companies combine GPS, cell towers, and Wi-Fi data to estimate the user's location. Unfortunately, in the indoor environment where GPS signals cannot penetrate building walls, the accuracy of these geo-location services can be in the range of tens of meters, which is insufficient in many cases [3].

Radio signals such as Bluetooth, ultra-wideband (UWB) [4], and radio frequency identification (RFID) [5, 6] are commonly employed for the purpose of indoor localization. The key idea is to use qualitative characteristics of radio signals (e.g., signal strength or triangulation) to estimate user location relative to a radio beacon (wireless access point). These approaches suffer from multipath effects, signal attenuation, and noise-induced interference [8]. Also, as these techniques require specialized wireless radio beacons to be installed in indoor locales, they are costly and thus lack scalability for wide-scale deployment.

S. Tiku (✉) · A. Mittal · S. Pasricha
Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO, USA
e-mail: saideep@colostate.edu; sudeep@colostate.edu

S. Pasricha, M. Shafique (eds.), *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing*, https://doi.org/10.1007/978-3-031-40677-5_1

Wi-Fi-based fingerprinting is perhaps the most popular radio-signal-based indoor localization technique being explored today. Wi-Fi is an ideal radio signal source for indoor localization as most public or private buildings are pre-equipped with Wi-Fi access points (APs). Lightweight middleware-based fingerprinting frameworks have been shown to run in the background to deliver location-based updates on smartphones [29, 37]. Fingerprinting with Wi-Fi works by first recording the strength of Wi-Fi radio signals in an indoor environment at different locations. Then, a user with a smartphone can capture Wi-Fi received signal strength indication (RSSI) data in real-time and compare it to previously recorded (stored) values to estimate their location in that environment. Fingerprinting techniques can deliver an accuracy of 6–8 m [28], with accuracy improving as the density of APs increases. However, in many indoor environments, noise and interference in the wireless spectrum (e.g., due to other electronic equipment, movement of people, and operating machinery) can reduce this accuracy. Combining fingerprinting-based frameworks with dead reckoning can improve this accuracy somewhat [8]. Dead reckoning refers to a class of techniques where inertial sensor data (e.g., from accelerometer and gyroscope) is used along with the previously known position data to determine the current location. Unfortunately, dead reckoning is infamously known to suffer from error accumulation (in inertial sensors) over time. Also, these techniques are not effective for people using wheelchairs or moving walkways.

The intelligent application of machine learning (ML) techniques can help to overcome noise and uncertainty during fingerprinting-based localization [8]. While traditional ML techniques work well at approximating simpler input–output functions, computationally intensive deep learning models are capable of dealing with more complex input–output mappings and can deliver better accuracy. Middleware-based offloading [30] and energy enhancement frameworks [31, 32, 37] may be a route to explore for computation and energy-intensive indoor localization services on smartphones. Furthermore, with the increase in the available computational power on mobile devices, it is now possible to deploy deep learning techniques such as convolutional neural networks (CNNs) on smartphones.

These are a special form of deep neural networks (DNNs) that are purposely designed to for image-based input data. CNNs are well known to automatically identify high-level features in the input images that have the heaviest impact on the final output. This process is known as feature learning. Prior to deep learning, feature learning was an expensive and time-intensive process that had to be conducted manually. CNN has been extremely successful in complex image classification problems and is finding applications in many emerging domains, e.g., self-driving cars [27].

In this chapter, we discuss an efficient framework that uses CNN-based Wi-Fi fingerprinting to deliver a superior level of indoor localization accuracy to a user with a smartphone. Our approach utilizes widely available Wi-Fi APs without requiring any customized/expensive infrastructure deployments. The framework works on a user's smartphone, within the computational capabilities of the device, and utilizes the radio interfaces for efficient fingerprinting-based localization. The main novel contributions of this chapter can be summarized as follows:

- We discuss a newly developed technique to extract images out of location fingerprints, which are then used to train a CNN that is designed to improve indoor localization robustness and accuracy.
- We implemented a hierarchical architecture to scale the CNN, so that our framework can be used in the real world where buildings can have large numbers of floors and corridors.
- We performed extensive testing of our algorithms with the state of the art across different buildings and indoor paths, to demonstrate the effectiveness of our proposed framework.

2 Related Works

Various efforts have been made to overcome the limitations associated with indoor localization. In this section, we summarize a few crucial efforts toward the same.

Numerous RFID-based [5, 6] indoor localization solutions that use proximity-based estimation techniques have been proposed. But the hardware expenses of these efforts increase dramatically with increasing accuracy requirements. Also, these approaches cannot be used with smartphones and require the use of specialized hardware. Indoor localization systems that use UWB [4] and ultrasound [10] have similar requirements for additional (costly) infrastructure and a lack of compatibility for use with commodity smartphones.

Triangulation-based methods, such as [11], use multiple antennas to locate a person or object. But these techniques require several antennas and regular upkeep of the associated hardware. Most techniques therefore favor using the more lightweight fingerprinting approach, often with Wi-Fi signals. UJIIndoorLoc [7] describes a technique to create a Wi-Fi fingerprint database and employs a k-nearest neighbor (KNN)-based model to predict location. Their average accuracy using KNN is 7.9 m. Given the current position (using fingerprinting) of a user walking in the indoor environment, pedestrian dead reckoning can be used to track the user's movement using a combination of microelectromechanical systems (MEMs)-based motion sensors ubiquitously found within contemporary smartphones and other wearable electronics. Dead reckoning techniques use the accelerometer to estimate the number of steps, a gyroscope for orientation, and a magnetometer to determine the heading direction. Such techniques have been employed in [12, 26] but have shown to deliver poor localization accuracy results when used alone.

Radar [12] and IndoorAtlas [26] proposed using hybrid indoor localization techniques. Radar [12] combines inertial sensors (dead reckoning) with Wi-Fi signal propagation models, whereas Indoor Atlas [26] combines information from several sensors such as magnetic, inertial, and camera sensors, for localization. LearnLoc [8] shallow feed-forward neural network models, dead reckoning techniques, and Wi-Fi fingerprinting to trade-off indoor localization accuracy and energy efficiency during localization on smartphones. Similar to LearnLoc, more recent works focus on optimizing and adapting light-weight machine learning techniques for the

purpose of fingerprinting-based indoor localization [28, 34–36]. However, all such techniques are limited by their ability to identify and match complex pattern within RSSI fingerprints. Additionally, a considerable amount of effort needs to be placed into the preprocessing, feature selection, and the tuning of the underlying model. Given these challenges, there is need of robust methodologies and algorithms for the purpose of fingerprinting-based indoor localization.

A few efforts have started to consider deep learning to assist with indoor localization. The work in [13] presents an approach that uses DNNs with Wi-Fi fingerprinting. The accuracy of the DNN is improved by using a hidden Markov model (HMM). The HMM takes temporal coherence into account and maintains a smooth transition between adjacent locations. But our analysis shows that the fine location prediction with the HMM fails in cases such as when moving back on the same path or taking a sharp turn. HMM predictions are also based on the previous position acquired through the DNN and, hence, can be prone to error accumulation. DeepFi [14] and ConFi [15] propose approaches that use the channel state information (CSI) of Wi-Fi signals to create fingerprints. But the CSI information in these approaches was obtained through the use of specialized hardware attached to a laptop. None of the mobile devices available today have the ability to capture CSI data. Due to this limitation, it is not feasible to implement these techniques on smartphones. Deep belief networks (DBNs) [16] have also been used for indoor localization, but the technology is based on custom UWB beacons that lead to very high implementation cost.

In summary, most works discussed so far either have specialized hardware requirements or are not designed to work on smartphones. Also, our real-world implementation and analysis concluded that the abovementioned frameworks slow down as they become resource intensive when being scaled to cover large buildings with multiple floors and corridors.

The framework discussed in this chapter, *CNNLOC*, overcomes the shortcomings of these state-of-the-art indoor localization approaches and was first presented in [33]. *CNNLOC* creates input images by using RSSI of Wi-Fi signals that are then used to train a CNN model, without requiring any specialized hardware/infrastructure. *CNNLOC* is easily deployable on current smartphones. The proposed framework also integrates a hierarchical scheme to enable scalability for large buildings with multiple floors and corridors/aisles.

3 Convolutional Neural Networks

Convolutional neural networks (CNNs) are specialized form of neural networks (NNs) that are designed for the explicit purpose of image classification [9]. They are highly resilient to noise in the input data and have shown to deliver excellent results for complex image classification tasks. The smallest unit of any neural network is

a perceptron and is inspired by the biological neuron present in the human brain. A perceptron is defined by the following equation:

$$y = \sum_{i=1}^{n} w_i x_i + w_0 \tag{1}$$

Here, y is the output, which is a weighted sum of the inputs x_i, with a bias (w_0). NNs have interconnected layers, and in each layer, there are several perceptrons, each with its own tunable weights and biases. Each layer receives some input, executes a dot product, and passes it to the output layer or the hidden layer in front of it [17]. An activation function is applied to the output y, limiting the range of values that it can take and establishes an input–output mapping defined by logistic regression. The most common activation functions used are *sigmoid* and *tanh* functions. The goal of an NN is to approximate a functional relationship between a set of inputs and outputs (training phase). The resulting NN then represents the approximated function that is used to make predictions for any given input (testing phase).

While an NN often contains a small number of hidden layers sandwiched between the input and output layer, a deep neural network (DNN) has a very large number of hidden layers. DNNs have a much higher computational complexity but in turn are also able to deliver very high accuracy. CNNs are a type of DNN that include several specialized NN layers, where each layer may serve a unique function. CNN classifiers are used to map input data to a finite set of output classes. For instance, given different animal pictures, a CNN model can be trained to categorize them into different classes such as cats and dogs. CNNs also make use of rectified linear units (ReLUs) as their activation function, which allows them to handle nonlinearity in the data.

In the training phase, our CNN model uses a stochastic gradient descent (SGD) algorithm. Adam [18] is an optimized variant of SGD and is used to optimize the learning process. The algorithm is designed to take advantage of two well-known techniques: RMSprop [19] and AdaGrad [20]. SGD maintains a constant learning rate for every weight update in the network. In contrast, Adam employs an adaptive learning rate for each network weight, with the learning rate being adapted as the training progresses. RMSprop uses the mean (first-order moment) of past-squared gradients and adjusts the weights based on how fast the gradient changes. Adam, to optimize the process, uses the variance (second-order moment) of past gradients and adjusts the weights accordingly.

The structure of the CNN in *CNNLOC* is inspired from the well-known CNN architectures, LeNet [21] and AlexNet [22]. Our CNN architecture is shown in Fig. 1. For the initial set of layers, our model has 2-D convolutional layer, followed by dense layers and culminates in an output layer. The 2-D convolutional layer works by convolving a specific region of the input image at a time. This region is known as a filter. The filter is shown by a rectangle (red-dotted lines). Each layer performs a convolution of a small region of the input image with the filter and feeds the result to

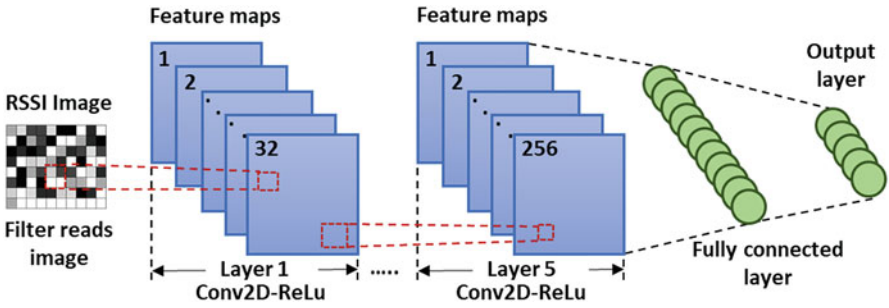


Fig. 1 CNN architecture

the ReLu activation function. Therefore, we refer to each layer as [Conv2D-ReLu]. To capture more details from the input image we can use a larger number of filters. For each filter, we get a feature map. For the first layer of [Conv2D-ReLU], we used 32 filters to create a set of 32 feature maps. We used five hidden layers of [Conv2D-ReLU], but only two are shown for brevity. The number of filters and layers is derived through empirical analysis as discussed in Sect. 4.4. A "stride" parameter determines the quantity of pixels that a filter will shift, to arrive at a new region of the input image to process. The stride and other "hyperparameters" of our CNN are further discussed in Sect. 4.4. In the end, a fully connected layer helps in identifying the individual class scores (in our case each class is a unique location). The class with the highest score is selected as the output. In this layer, all the neurons are connected to the neurons in the previous layer (green-dotted lines).

In a conventional CNN, a pooling layer is used to down-sample the image when the size of the input image is too big. In our case, the input image is small, and therefore, we do not need this step. We want our CNN to learn all the features from the entire image.

4 CNNLOC Framework: Overview

4.1 Overview

An overview of our *CNNLOC* indoor localization framework is shown in Fig. 2. In the framework, we utilize the available Wi-Fi access points (APs) in an indoor environment to create an RSSI fingerprint database. Our framework is divided into two phases. The first phase involves RSSI data collection, cleaning, and preprocessing. This preprocessed data is used to create a database of images. Each image represents a Wi-Fi RSSI-based signature that is unique to a location label. Each location label is further associated with an *x*-*y* coordinate. This database of images is used to train a CNN model. The trained model is deployed on to a smartphone. In the second phase, or the online phase, real-time AP data is converted

into an image and then fed to the trained CNN model to predict the location of the user. The CNN model predicts the closest block that was sampled as the users' location. A detailed description of the preprocessing is described in the next section.

4.2 *Preprocessing of RSSI Data*

The process of image database creation begins with the collection of RSSI fingerprints as shown in the top half of Fig. 2. The RSSI for various APs are captured along with the corresponding location labels and x-y coordinates. Each AP is uniquely identified using its unique media access control (MAC) address. We only maintain information for known Wi-Fi APs and hence clean the captured data. This ensures that our trained model is not polluted by unstable Wi-Fi APs. On the RSSI scale, values typically range between -99 dB (lowest) and $-$ 0 dB (highest). To indicate that a specific AP is unreachable, -100 is used, or no signal is received from it. We normalize the RSSI values on a scale from 0 and 1, where 0 represents no signal and represents the strongest signal.

Assume that while fingerprinting an indoor location, a total of K APs are discovered at N unique locations. These combine to form a two-dimensional matrix of size $N \times K$. Then, the normalized RSSI fingerprint at the Nth location, denoted as l_N, is given by a row vector $[r_1, r_2, \ldots, r_K]$, denoted by R_N. Therefore, each column vector, $[w_1, w_2, \ldots, w_N]$ would represent the normalized RSSI values of the Kth AP at all N locations, denoted by W_K. We calculate the Pearson correlation coefficient (PCC) [23] between each column vector W_K and the location vector $[l_1, l_2, \ldots, l_N]$. The result is a vector of correlation values denoted as C. PCC is useful in identifying the most significant APs in the database that impact localization

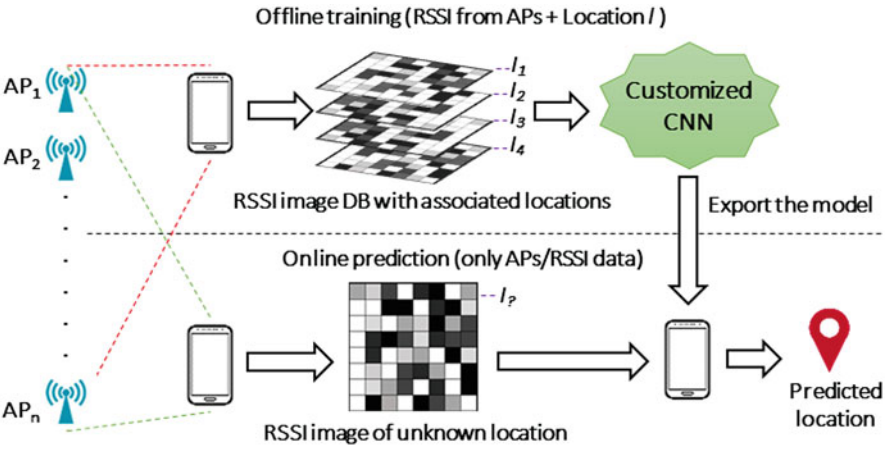


Fig. 2 An overview of the *CNNLOC* framework

accuracy. The coefficient values range across a scale of -1 to $+1$. If the relationship is -1, it represents a strong negative relationship, whereas $+1$ represents a strong positive relationship, and 0 implies that the input and output have no relationship.

We only consider the magnitude of the correlation as we are only concerned with the strength of the relationship. APs with very low correlation with the output coordinates are not useful for the purpose of indoor localization. Therefore, we can remove APs whose correlation to the output coordinates is below a certain threshold ($|\text{PCC}| < 0.3$). This removes inconsequential APs from the collected Wi-Fi data and helps reduce the computational workload of the framework. The normalized RSSI data from the remaining high-correlation APs is used to create an RSSI image database, as explained in the next section.

4.3 RSSI Image Database

In this section, we present our approach to convert RSSI data for a given location into a gray scale image. A collection of these images for all fingerprinted locations forms the RSSI image database. To form gray scale images, a Hadamard product (*HP*) [24] is calculated for each R and C. HP is defined as an element wise multiplication of two arrays or vectors:

$$\text{HP} = \sum_{i=1}^{N} R_i \circ C \tag{2}$$

The dimension of each *HP* is $1 \times K$. Then, the *HP* matrix is reshaped into a $p \times p$ matrix, which represents a 2-D image as shown in Fig. 3. The HP is padded with zeros in the case that K is less than p^2. Therefore, we now have a set of N images of size $p \times p$ in our database. These images are used to train the CNNs.

Figure 3 shows two images of size 8×8 created for two unique fingerprints (signatures) associated with two different locations. Each pixel value is scaled on a scale of 0–1. The patterns in each of these images will be unique to a location and change slightly as we move along an indoor path.

In Eq. (2), the product of PCC and normalized RSSI value for each AP is used to form a matrix. Its purpose is to promote the impact of the APs that are highly correlated to fingerprinted locations. Even though there may be attenuation of Wi-Fi signals due to multipath fading effects, the image may fade but will likely still have the pattern information retained. These patterns that are unique to every location can be easily learned by a CNN. The hyperparameters and their use in *CNNLOC* are discussed next.

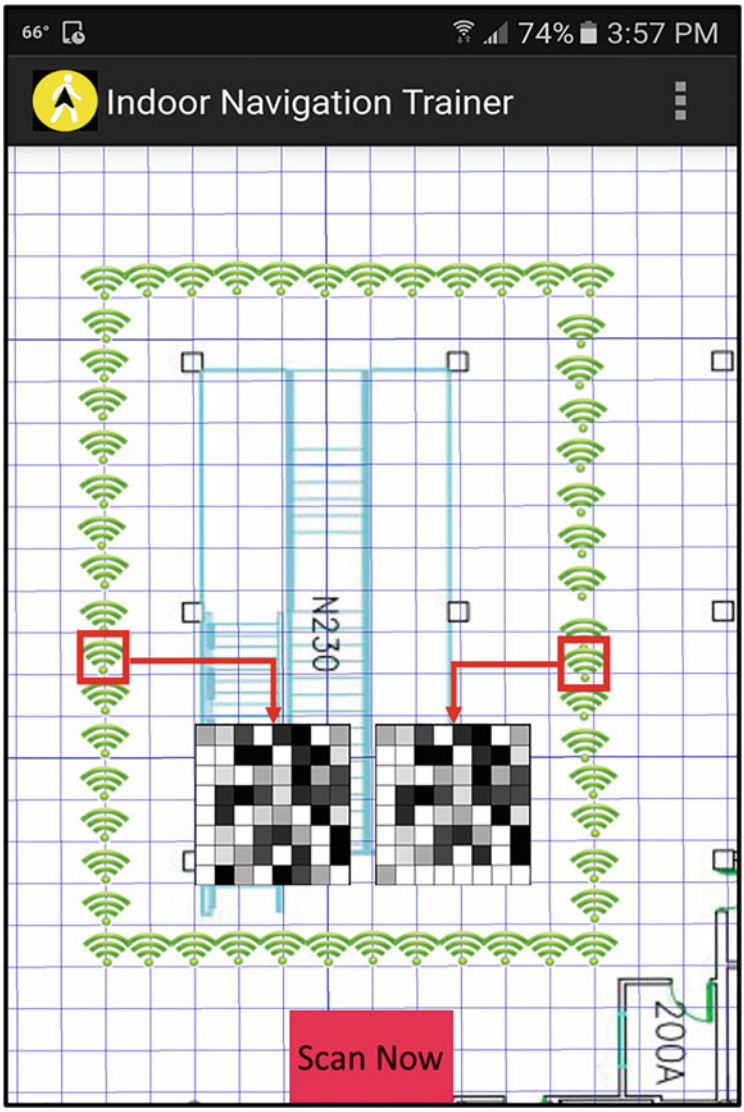


Fig. 3 Snapshot of CNNLOC's offline phase application showing contrasting the images created for two unique locations. The green icons represent locations that are fingerprinted along an indoor path. The two locations shown are 10 m apart

4.4 Hyperparameters

The accuracy of the CNN model depends on the optimization of the hyperparameters that control its architecture, which is the most important factor in the

performance of CNN. A smaller network may not perform well, and a larger network may be slow and prone to overfitting. There are no defined rules in deep learning that help in estimating the appropriate hyperparameters and therefore need to be empirically found through an iterative process. The estimated hyperparameters are also highly dependent on the input dataset. For the sake of repeatability, we discuss some the key hyperparameters of our CNN model below:

- *Number of hidden layers*: A large number of hidden layers lead to longer execution times and conversely, fewer hidden layers may produce inaccurate results due to the challenges associated with vanishing gradients. We found that five layers of [Conv2D-ReLU] worked best for our purposes.
- *Size of filter*: This defines the image area that the filter considers at a time, before moving to the next region of the image. A large filter size might aggregate a large chunk of information in one pass. The optimum filter size in our case was found to be 2×2.
- *Stride size*: The number of pixels a filter moves by is dictated by the stride size. We set it to 1 because the size of our image is very small, and we do not wish to lose any information.
- *Number of filters*: Each filter extracts a distinct set of features from the input to construct different feature maps. Each feature map holds unique information about the input image. The best results were obtained if we started with a lower number of filters and increased them in the successive layers to capture greater uniqueness in the patterns. There were 32 filters in the first layer and were doubled for each subsequent layer up to 256 filters such that both the fourth and fifth layer had 256 filters.

4.5 Integrating Hierarchy for Scalability

We architect our *CNNLOC* framework to scale up to larger problem sizes than that handled by most prior efforts. Toward this, we enhanced our framework by integrating a hierarchical classifier. The resulting hierarchical classifier employs a combination of smaller CNN modules, which work together to deliver a location prediction. Figure 4 shows the hierarchical classification structure of the framework. Each CNN model has a label that starts with C. The C1 model classifies the floor numbers, and then in the next layer, C20 or C21 identifies the corridor on that floor. Once the corridor is located, one of the CNNs from the third layer (C30–C35) will predict the fine-grain location of the user. This hierarchical approach can further be extended across buildings.

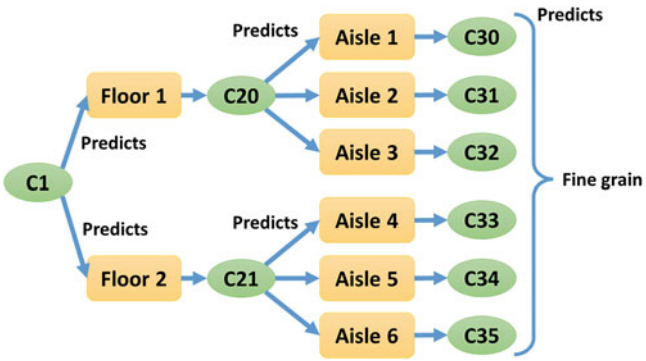


Fig. 4 A general architecture for the hierarchical classifier

Table 1 Indoor paths used in experiments

Path name	Length (m)	Shape
Library	30	U shape
Clark A	35	Semi-octagonal
Physics	28	Square shape

5 Experiments

5.1 Experimental Setup

The following sections describe the *CNNLOC* implementation and experimental results that were conducted on three independent indoor paths as described in Table 1. The overall floor plan of the path is divided into a grid and tiles of interest are labeled sequentially from 1 to *N*. For the purposes of this work, each square in the grid has an area of 1 m^2. Based on our analysis (not presented here), having grid tiles of size smaller than 1 m^2 did not lead to any improvements. Each of these labeled tiles is then treated as a "class." This allows us to formulate indoor localization as a classification problem. Figure 5 shows an example of a path covered in the library building floor plan with labeled squares. Each label further translates into an *x*-*y* coordinate. Five Wi-Fi scans were conducted at each square during the fingerprinting (training) phase.

5.2 Smartphone Implementation

An android application was built to collect Wi-Fi fingerprints (i.e., RSSI samples from multiple APs at each location) and for testing. The application is compatible with Android 6.0 and was tested on a Samsung Galaxy S6. After fingerprint data collection, the data was preprocessed as described in the previous section for the CNN model. The entire dataset is split into training and testing samples, so we can

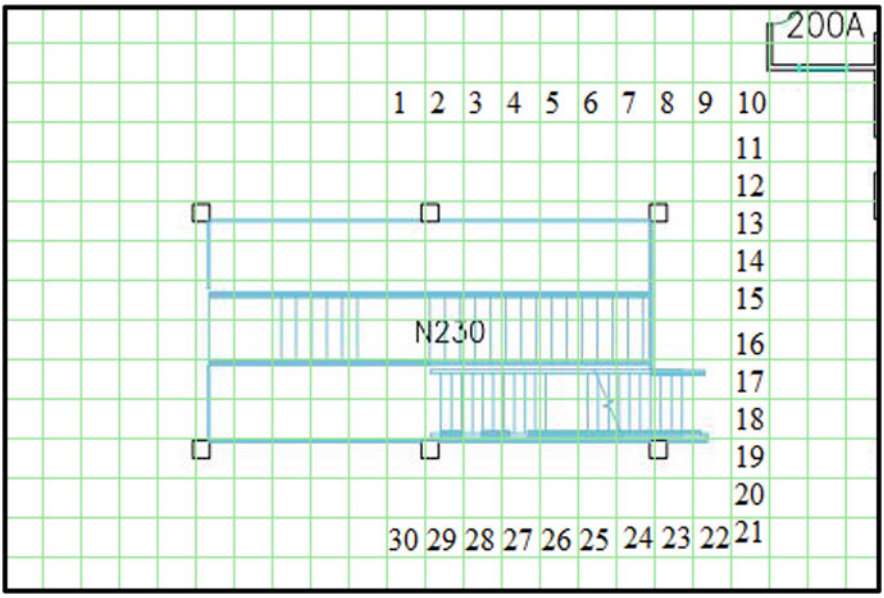


Fig. 5 Library building path divided into a grid, with squares along the path labeled sequentially from 1 to 30

check how well our models perform. We used one-fifth of the total samples for testing, and four-fifth of the samples were used for training.

5.3 Experimental Results

We compared our *CNNLOC* indoor localization framework with three other indoor localization frameworks from prior work. The first work we implemented is based on the approach in [25] and employs support vector regression (SVR). The approach forms one or more hyperplanes in a multidimensional space segregating similar data point, which are then used for regression. The second work is based on the KNN technique from [8], which is a nonparametric approach that is based on the idea that similar input will have similar outputs. Lastly, we compare our work against a DNN based approach [13] that improves upon conventional NNs by incorporating a large number of hidden layers. All of these techniques supplement the Wi-Fi fingerprinting approach with a machine learning model to provide robustness against noise and interference effects. Our experiments in the rest of this section first discuss the localization accuracy results for the techniques. Subsequently, we also discuss results for the scalability of our framework using a hierarchical classification enhancement approach. Lastly, we contrast the accuracy of our framework with that reported by other indoor localization techniques.

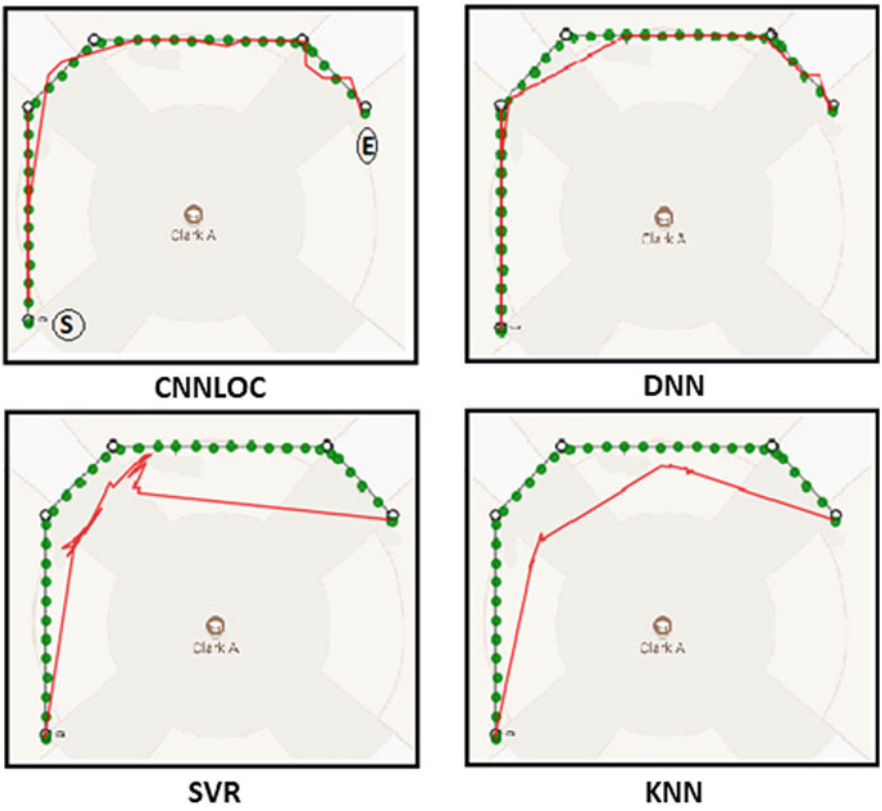


Fig. 6 Path traced using different techniques at the Clark building path. Green and red traces indicate actual and predicted paths respectively

5.3.1 Indoor Localization Accuracy Comparison

The overall indoor localization quality as experienced by a user is heavily impacted by the stability of the predicted path that is traced over an indoor localization session. In an attempt to evaluate this, we compare the paths traced by various indoor localization frameworks as compared with the proposed *CNNLOC* framework. Figure 6 shows the paths predicted by the four techniques, for the indoor path in the Clark building. The green dots along the path represent the points where Wi-Fi RSSI fingerprint samples were collected to create the training fingerprint dataset. The distance between each of the green dots is 1 m. In the offline phase, the RSSI fingerprint at each green dot is converted into an image. The online phase consists of the user walking along this path, and the red lines in Fig. 6 represent the paths predicted by the four techniques. It is observed that KNN [8] and SVR [25] stray off the actual path the most, whereas DNN and *CNNLOC* perform much better. This is likely because KNN and SVR are both regression-based techniques where

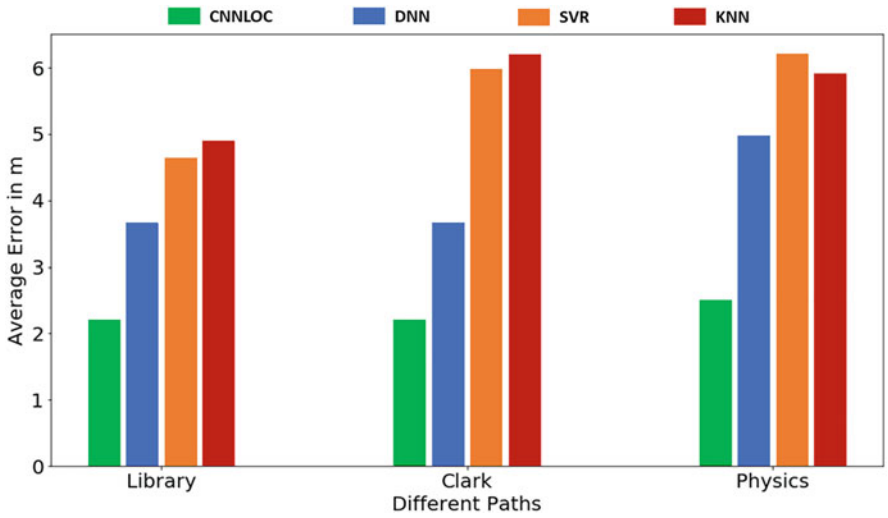


Fig. 7 Comparison of indoor localization techniques

the prediction is impacted by neighboring data points in the RSSI Euclidean space. Two locations that have RSSI fingerprints that are very close to each other in the Euclidian space might not be close to each other on the actual floor plan. This leads to large localization errors, especially when utilizing regression-based approaches. The transition from one location to another is smoother for CNN as it is able to distinguish between closely spaced sampling locations due to our RSSI-to-image conversion technique. The convolutional model is able to identify patterns within individual RSSI images and classify them as locations. From Fig. 6, it is evident that our *CNNLOC* framework produces stable predictions for the Clark path.

Figure 7 shows a bar graph that summarizes the average location estimation error in meters for the various frameworks on the three different indoor floor plans considered. We found that the KNN approach is the least reliable among all techniques with a mean error of 5.5 m and large variations across the paths. The SVR-based approach has a similar mean error as the KNN approach. The DNN-based approach shows lower error across all of the paths. But it does not perform consistently across all of the paths, and the mean error is always higher than that for *CNNLOC*. This may be due to the fact that the filters in CNN are set up to focus on the image with a much finer granularity than the DNN approach is capable of. We also observe that all techniques perform the worst in the Physics department. This is due to the fact that the path in the Physics department is near the entrance of the building and has a lower density of Wi-Fi APs as compared with the other paths. The Library and Clark paths have a higher density of Wi-Fi APs present; hence, better accuracy can be achieved. Our proposed *CNNLOC* framework is the most reliable framework with the lowest mean error of less than 2 m.

5.3.2 CNNLOC Scalability Analysis

The size and complexity of a deep learning model is directly correlated to number of classes and associated dataset in use. The baseline formulation of our proposed framework does not account for the increasing area of floor plan that needs to be covered. To overcome this, we proposed a hierarchal approach for *CNNLOC* (Sect. 4.5). We consider a scenario when *CNNLOC* is required to predict a location inside a building with two floors and with three corridors on each floor. The length of each corridor is approximately 30 m. We combined several small CNNs (in our case 9 small CNNs), such that a smaller number of weights are associated with each layer in the network than if a single larger CNN was used.

We first evaluated the accuracy of predictions, for *CNNLOC* with and without the hierarchical classifier. For the first and second layer of the hierarchical classifier (shown in Fig. 4), the accuracy is determined by the number of times the system predicts the correct floor and corridor. We found that floors and corridors were accurately predicted 99.67% and 98.36% of times, respectively. For the final layer, we found that there was no difference in accuracy between the hierarchal approach and the nonhierarchical approach. This is because in the last level, both the approaches use the same model.

Figure 8 shows the benefits in terms of time taken to generate a prediction with the hierarchical versus the nonhierarchical *CNNLOC* framework. We performed our experiment for four walking scenarios ("runs") in the indoor environment (building with two floors and with three corridors on each floor). We found that the hierarchical *CNNLOC* model only takes 2.42 ms to make a prediction on average, whereas the nonhierarchical *CNNLOC* takes longer (3.4 ms). Thus, the proposed hierarchical classifier represents a promising approach to reduce prediction time due to the fewer number of weights in the CNN layers in the hierarchical approach, which leads to fewer computations in real time.

5.3.3 Accuracy Analysis with Other Approaches

Our experimental results in the previous sections have shown that *CNNLOC* delivers better localization accuracy over the KNN [8], DNN [13], and SVR [25] frameworks. The UJIIndoorLoc [7] framework is reported to have an accuracy of 4–7m. Our average accuracy is also almost twice that of RADAR [12]. If we consider frameworks that used CSI (DeepFi [14] and ConFi [15]), our accuracy is very close to both at just under 2 m. However, [14, 15] use special equipment to capture CSI and cannot be used with mobile devices. In contrast, our proposed *CNNLOC* framework is easy to deploy on today's smartphones, does not require any specialized infrastructure (e.g., custom beacons), and can be used in buildings wherever Wi-Fi infrastructure preexists.

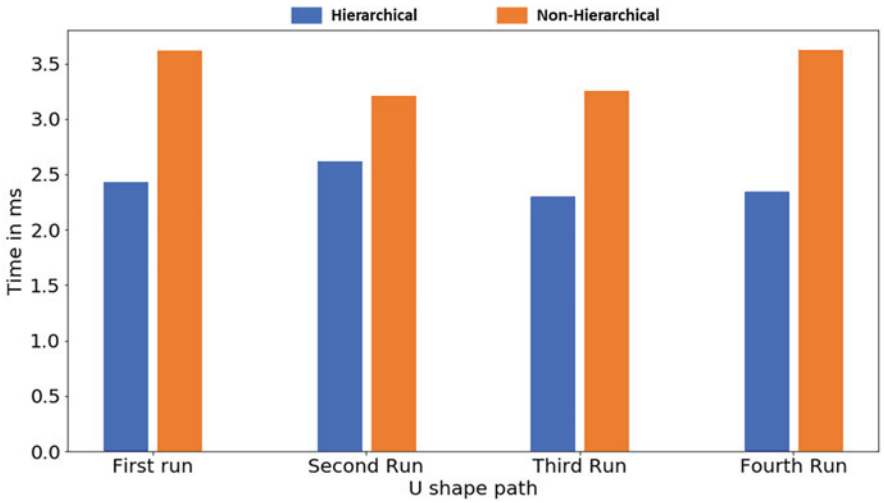


Fig. 8 A comparison of execution times for hierarchical and nonhierarchical versions the CNNLOC framework

6 Conclusion

In this chapter, we discuss the *CNNLOC* framework [33] that uses Wi-Fi fingerprints and convolutional neural networks (CNNs) for accurate and robust indoor localization. We compared our work against three different state-of-the-art indoor localization frameworks from prior work. Our framework outperforms these approaches and delivers localization accuracy under 2 m. *CNNLOC* has the advantage of being easily implemented without the overhead of expensive infrastructure and is smartphone compatible. We also demonstrated how a hierarchical classifier can improve the scalability of this framework. *CNNLOC* represents a promising framework that can deliver reliable and accurate indoor localization for smartphone users.

Acknowledgments This work was supported by the National Science Foundation (NSF), through grant CNS-2132385.

References

1. How Google Maps Makes Money. (2022) [Online] https://www.investopedia.com/articles/investing/061115/how-does-google-maps-makes-money.asp. Accessed 1 Apr 2022
2. Target Rolls Out Bluetooth Beacon Technology in Stores to Power New Indoor Maps in its App. (2017) [Online] https://techcrunch.com/2017/09/20/target-rolls-out-bluetooth-beacon-technology-in-stores-to-power-new-indoor-maps-in-its-app/. Accessed 1 Apr 2022

3. Case Study: Accuracy & Precision of Google Analytics Geolocation. (2017) [Online] Available at: https://radical-analytics.com/case-study-accuracy-precision-of-google-analytics-geolocation-4264510612c0. Accessed 1 Dec 2017
4. Ubisense Research Network. [Online] Available at: http://www.ubisense.net/. Accessed 1 Dec 2017
5. Jin, G., Lu, X., Park, M.: An indoor localization mechanism using active RFID tag. In: IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC). IEEE (2006)
6. Chen, Z., Wang, C.: Modeling RFID signal distribution based on neural network combined with continuous ant colony optimization. Neurocomputing. **123**, 354–361 (2014)
7. Torres-Sospedra, J., et al.: UJIIndoorLoc: a new multi-building and multi-floor database for WLAN fingerprint-based indoor localization problems. In: IEEE Indoor Positioning and Indoor Navigation (IPIN). IEEE (2014)
8. Pasricha, S., Ugave, V., Han, Q., Anderson, C.: LearnLoc: a framework for smart indoor localization with embedded mobile devices. In: ACM/IEEE International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS). IEEE (2015)
9. Pouyanfar, S., Sadiq, S., Yan, Y., Tian, H., Tao, Y., Reyes, M.P., Shyu, M., Chen, S., Iyengar, S.S.: A survey on deep learning: algorithms, techniques, and applications. ACM Comput. Surv. **51**(5), 1–36 (2019)
10. Borriello, G., Liu, A., Offer, T., Palistrant, C., Sharp, R.: WALRUS: wireless acoustic location with room-level resolution using ultrasound. In: Mobile systems, applications, and services (MobiSys). ACM (2005)
11. Yang, C., Shao, H.R.: WiFi-based indoor positioning. IEEE Commun. Mag. **53**(3), 150–157 (2015)
12. Bahl, P., Padmanabhan, V.: RADAR: an in-building RF-based user location and tracking system. In: IEEE International Conference on Computer Communications (INFOCOM). IEEE (2000)
13. Zhang, W., Liu, K., Zhang, W., Zhang, Y., Gu, J.: Deep neural networks for wireless localization in indoor and outdoor environments. Neurocomputing. **194**, 279–287 (2016)
14. Wang, X., Gao, L., Mao, S., Pandey, S.: DeepFi: deep learning for indoor fingerprinting using channel state information. In: IEEE Wireless Communications and Networking Conference (WCNC). IEEE (2015)
15. Chen, H., Zhang, Y., Li, W., Tao, X., Zhang, P.: ConFi: convolutional neural networks based indoor WiFi localization using channel state information. IEEE Access. **5**, 18066–18074 (2017)
16. Hua, Y., Guo, J., Zhao, H.: Deep belief networks and deep learning. In: IEEE International Conference on Intelligent Computing and Internet of Things (ICIT). IEEE (2015)
17. Stanford CNN Tutorial. [Online] Available at: http://cs231n.github.io/convolutional-networks. Accessed 1 Apr 2022
18. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. In: International Conference on Learning Representations (ICLR) (2015)
19. RMSProp. [Online] https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf. Accessed 1 Apr 2022
20. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. ACM J. Mach. Learn. Res. **12**, 2121–2159 (2011)
21. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE. **86**(11), 2278–2324 (1998)
22. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems. Neural IPS 2012 (2012)
23. Benesty, J., Chen, J., Huang, Y., Cohen, I.: Pearson correlation coefficient. In: Topics in Signal Processing, vol. 2, pp. 1–4. Springer (2009)
24. Styan, G.: Hadamard products and multivariate statistical analysis. In: Linear Algebra and its Applications, vol. 6, pp. 217–240. Elsevier (1973)

25. Cheng, Y.K., Chou, H.J., Chang, R.Y.: Machine-learning indoor localization with access point selection and signal strength reconstruction. In: IEEE Vehicular Technology Conference (VTC). IEEE (2016)
26. IndoorAtlas. [Online] http://www.indooratlas.com/. Accessed 1 Apr 2022
27. Rausch, V., Hansen, A., Solowjow, E., Liu, C., Kreuzer, E., Hedrick, J.K.: Learning a deep neural net policy for end-to-end control of autonomous vehicles. In: IEEE American Control Conference (ACC). IEEE (2017)
28. Langlois, C., Tiku, S., Pasricha, S.: Indoor localization with smartphones: harnessing the sensor suite in your pocket. IEEE Consum. Electron. **6**(4), 70–80 (2017)
29. Tiku, S., Pasricha, S.: Energy-efficient and robust middleware prototyping for smart mobile computing. In: IEEE International Symposium on Rapid System Prototyping (RSP). IEEE (2017)
30. Khune, A., Pasricha, S.: Mobile network-aware middleware framework for energy-efficient cloud offloading of smartphone applications. In: IEEE Consumer Electronics. IEEE (2017)
31. Donohoo, B., Ohlsen, C., Pasricha, S.: A middleware framework for application-aware and user-specific energy optimization in smart Mobile devices. Journal Pervasive Mob. Comput. **20**, 47–63 (2015)
32. Donohoo, B., Ohlsen, C., Pasricha, S., Anderson, C., Xiang, Y.: Context-aware energy enhancements for smart mobile devices. IEEE Trans. Mob. Comput. **13**(8), 1720–1732 (2014)
33. Mittal, A., Tiku, S., Pasricha, S.: Adapting convolutional neural networks for indoor localization with smart mobile devices. In: ACM Great Lakes Symposium on VLSI (GLSVLSI). ACM (2018)
34. Tiku, S., Pasricha, S., Notaros, B., Han, Q.: SHERPA: a lightweight smartphone heterogeneity resilient portable indoor localization framework. In: IEEE International Conference on Embedded Software and Systems (ICESS). IEEE (2019)
35. Tiku, S., Pasricha, S.: PortLoc: a portable data-driven indoor localization framework for smartphones. IEEE Des. Test. **36**(5), 18–26 (2019)
36. Tiku, S., Pasricha, S., Notaros, B., Han, Q.: A hidden markov model based smartphone heterogeneity resilient portable indoor localization framework. J. Syst. Archit. **108**, 101806 (2020)
37. Pasricha, S., Ayoub, R., Kishinevsky, M., Mandal, S.K., Ogras, U.Y.: A survey on energy management for mobile and IoT devices. IEEE Des. Test. **37**(5), 7–24 (2020)

An End-to-End Embedded Neural Architecture Search and Model Compression Framework for Healthcare Applications and Use-Cases

Bharath Srinivas Prabakaran and Muhammad Shafique

1 Introduction

As discussed in chapter "Massively Parallel Neural Processing Array (MPNA): A CNN Accelerator for Embedded Systems", deep learning has revolutionized domains worldwide by improving machine understanding and has been used to develop state-of-the-art techniques in fields like computer vision [15], speech recognition and natural language processing [21], healthcare [9], medicine [49], bioinformatics [20], etc. These developments are primarily driven by the rising computational capabilities of modern processing platforms and the availability of massive new annotated datasets that enable the model to learn the necessary information. Fields like medicine and healthcare generate massive amounts of data, in the order of hundreds of exabytes is the USA alone, which can be leveraged by deep learning technologies to significantly improve a user's quality of life and obtain substantial benefits. Furthermore, healthcare is one of the largest revenue-generating industries in the world, requiring contributions upward of 10% of the country's Gross Domestic Product (GDP) annually [4]. Countries like the United States routinely spend up to 17.8% of their GDP on healthcare [35]. The global health industry is expected to generate over $10 trillion revenue, annually by 2022, which is a highly conservative estimate as it does not consider the increasing global elderly population percentages [47]. The rising global average life expectancy is another byproduct of the substantial technological advancements in medicine and healthcare [34]. The Internet of Things (IoT) phenomenon serves as an ideal

B. S. Prabakaran (✉)
Institute of Computer Engineering, Technische Universität Wien (TU Wien), Vienna, Austria
e-mail: bharath.prabakaran@tuwien.ac.at

M. Shafique
Engineering Division, New York University Abu Dhabi, Abu Dhabi, UAE
e-mail: muhammad.shafique@nyu.edu

S. Pasricha, M. Shafique (eds.), *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing*, https://doi.org/10.1007/978-3-031-40677-5_2

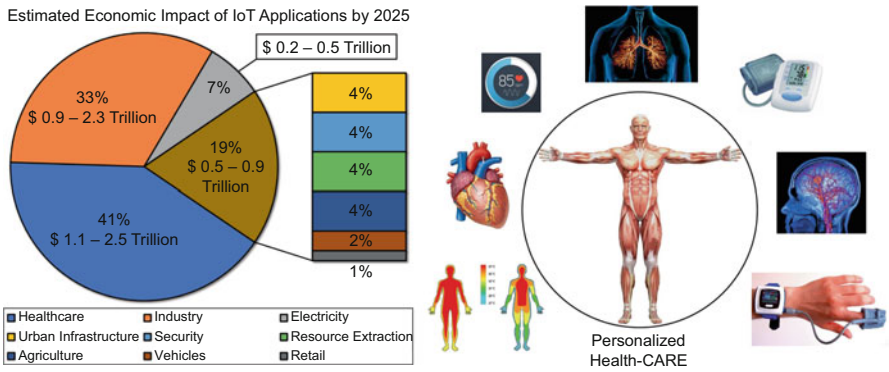


Fig. 1 Breakdown of the estimated economic impact of Internet of Things applications by 2025; an overview of the human bio-signals that can be monitored and analyzed for patient-specific care

opportunity that can be exploited by investigating its applicability in the healthcare sector to offer more efficient and user-friendly services that can be used to improve quality of life. The Internet of Medical Things (IoMT) market is expected to grow exponentially to achieve an annual economic impact of $1.1–$2.5 trillion by 2025, which constitutes 41% of the impact of the complete IoT sector [29]. This includes applications like personalized health monitoring, disease diagnostics, patient care, and physiological signal, or bio-signal, monitoring, and analytics to recommend person-specific lifestyle changes or health recommendations [2, 19], especially by investing heavily in the capabilities and advancements of deep learning. A breakdown of the estimated economic impact of IoT applications by the year 2025 and an overview of human bio-signals, which can be monitored and analyzed, are presented in Fig. 1.

An overview of a few healthcare use-cases and applications is discussed next, before moving on to the framework that can be used for exploring the deep learning models that can be deployed for a given use-case, given its output quality requirements and hardware constraints of the target execution platform.

1.1 Deep Learning in Healthcare: Potential Use-Cases and Applications

Medical Imaging Deep learning has been largely investigated as a solution to address research challenges in the computer vision and imaging domains due to the availability of massive labeled and annotated datasets. Therefore, the primary healthcare domain suitable for investigating the applicability of deep learning would be medical imaging. Since various technologies like X-Rays, CT (Computed Tomography) and MRI (Magnetic Resonance Imaging) scans, ultrasound, etc. are regularly used by clinicians and doctors to help patients, deep learning can be highly

beneficial in such scenarios where they can be deployed as clinical assistants that can aid in diagnostics and radiology.

Electronic Health Record Analysis Electronic Health Record (EHR) is a collection of health data related to a patient across time, including their medical history, current and past medications, allergies, immunization information, lab test results, radiology imaging studies, age, weight, etc. These EHRs, from several patients, are combined into a large pool, based on their demographics, to mine and extract relevant information that can be used to devise new treatment strategies and improve the health status of the patients. Deep learning techniques have successfully demonstrated the ability to combine all this information to extract vital health statistics, including the ability to predict a patient's mortality.

Drug Discovery The processing capabilities of deep learning models can also be leveraged on massive genomic, clinical, and population-level data to identify potential drugs or compounds that can, "by-design," explore associations with existing cell signaling pathways, pharmaceutical and environmental interactions, to identify cures for known health problems. For instance, the protein folding problem, which had fazed the community for more than five decades, was recently solved by the AlphaFold deep learning model, proposed by researchers from Google [20]. This enables researchers to predict the structure of a protein complex, at atomic granularity, using just its amino acid composition, which can further enable scientists to identify compounds that can prevent the formation of lethal proteins in hereditary medical conditions like Alzheimer's or Parkinson's.

Precision Medicine Genomic analysis, in combination with drug discovery approaches, might be the key to develop the next generation of precise targeted medical treatments, which improve the user's quality of life. Understanding the genetic capability of the underlying condition, such as the type of cancer, its ability to reproduce, and the way it propagates, can enable scientists to better develop user-specific treatment options. However, processing such large amounts of data can take anywhere from weeks to months, which can be circumvent by deep learning models to the order of hours, enabling such explorations.

Similarly, there are plenty of other healthcare applications, like real-time monitoring and processing of bio-signals, sleep apnea detection, detecting gait patterns, genomic analysis, artificial intelligence-based chatbots and health assistants, and many more, that can benefit by investigating the applicability of deep learning in these use-cases (Table 1). We delve into the field of deep learning for healthcare,

Table 1 A summary of key state-of-the-art techniques in deep learning for healthcare

Deep learning in healthcare	References
Medical imaging	[1, 10, 15, 25, 27, 43]
Electronic health record analysis	[18, 39, 45, 46, 50]
Drug discovery and precision medicine	[7, 20, 24, 36, 38, 40, 42]
Others	[17, 23, 37]

next, by presenting a comprehensive embedded neural architecture search and model compression framework for healthcare applications and illustrate the benefits by evaluating its efficacy on a bio-signal processing use-case.

2 Embedded Neural Architecture Search and Model Compression Framework for Healthcare Applications

Figure 2 illustrates an overview of the deep neural network (DNN) model search and compression framework for healthcare, which is composed of six key stages. The framework considers (1) the user specifications and quality requirements, such as the required prediction labels, output classes, expected accuracy or precision of the model and (2) the hardware constraints of the target execution platform, such as the available on-chip memory (MBs) and the maximum number of floating-point operations (FLOPs) that can be executed per second to construct the required dataset from the existing labeled annotations (more details provided in Sect. 2.3) and explore the design space of DNN models that can be useful for the application.

2.1 User Specifications and Requirements

The framework enables dynamic model exploration by restricting the output classes of the DNN, based on the user requirements; besides the normal and anomalous classes, the user might require an output class specific to the target use-case. For instance, a hospital might require the model to classify X-Rays or CT scans to explicitly detect cases of lung infection caused by the novel SARS-CoV-2 coronavirus as a separate classification. These instances can be included by the framework to generate and explore DNN models specific to this case, given that the corresponding annotated data is already present in the dataset used for construction.

The framework currently considers four metrics for evaluating the quality of a model (Q), namely, Accuracy (A), Precision (P), Recall (R), and F1-score (F). Accuracy of a model is defined as the ratio of correct classifications with respect to the total number of classifications, Precision signifies the percentage of classified items that are relevant, Recall is defined as the percentage of relevant items that are classified correctly, and F1-score is used to evaluate the Recall and Precision of a model by estimating their harmonic mean. The system designer can specify a constraint in the framework using any of the abovementioned metrics, while exploring the DNN models for the application to ensure that the models obtained after exploration satisfy the required quality constraint. Evaluation with other use-case specific metrics is orthogonal to these standards and can be easily incorporated into the framework. These metrics are estimated as follows:

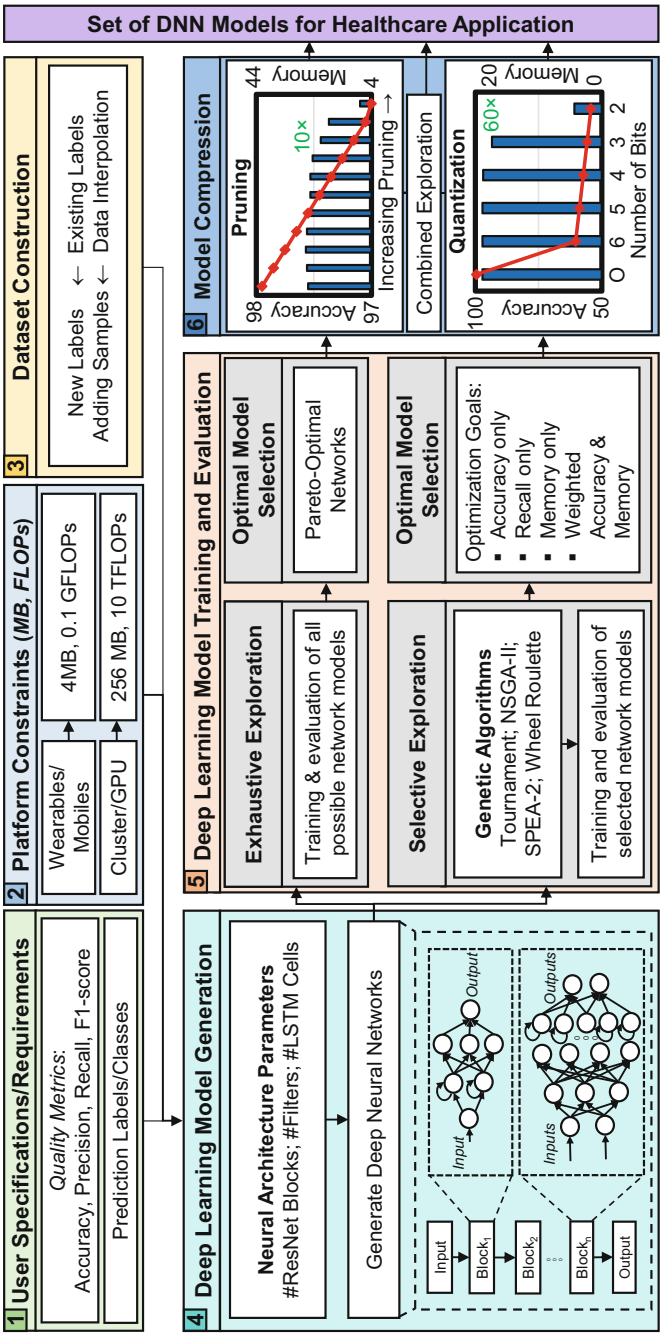


Fig. 2 An overview of the key stages in the deep learning model exploration and compression framework for healthcare applications (adapted from [37])

$$A = \frac{TP + TN}{\#Classifications}, P = \frac{TP}{TP + FP}, R = \frac{TP}{TP + FN}, F = \frac{2 * P * R}{P + R}$$

where TP stands for the number of true positive predictions, whereas TN, FP, and FN depict the number of true negative, the number of false positive, and the number of false negative predictions, respectively.

2.2 Platform Constraints

Similarly, to ensure that the explored networks do not require computational resources beyond those available on the target execution platform, a hardware constraint is implemented before the evaluation stage, which the user can define as per their requirements. Currently, the hardware constraints of the system can be specified by the system designer either in terms of the memory overhead (B), i.e., the maximum size of the model that can be accommodated on the platform, or the execution time in terms of the maximum number of floating-point operations that the platform can execute for a single inference (FP). Like the quality metrics, incorporating other additional platform-specific hardware constraints are orthogonal to our current approach and can be an easily added functionality to the framework. Explicitly specifying hardware constraints requires the framework to identify models that offer the best quality under the given constraints, which enables the exploration of a trade-off between output quality and hardware requirements of the model, two metrics that typically maintain an inverse correlation.

2.3 Dataset Construction

To ensure that the model developed is application-driven, a custom dataset is constructed by fusing labels, in an existing healthcare dataset, in order to create the required output classes. Note that each label in the custom dataset needs to be correspond to one of the labels in the existing healthcare dataset, to ensure coherence. For instance, with respect to the COVID-19 classifier application discussed earlier, there could be varying diagnosis for the lung X-Rays or CT scans present in the dataset, including pneumonia, pleural effusion, cystic fibrosis, or lung cancer, which are ultimately labeled as “anomaly” in the constructed dataset, given the sole focus of the application is to just classify amongst normal, anomaly, and “COVID-19.” A similar methodology can be used to construct custom datasets for a given healthcare application, as discussed with the help of a use-case in Sect. 3.

2.4 Deep Learning Model Generation

With the necessary information regarding the user specifications, platform constraints, and the constructed dataset, we generate the set of possible DNN models (ψ) by varying the key neural architecture parameters. Since our approach considers a relevant state-of-the-art model as a baseline, we extract the key architectural parameters from the baseline model and vary them to generate different models that can achieve (near) state-of-the-art accuracy with reduced hardware requirements. For instance, the use-case discussed in Sect. 3 explores three DNN model parameters, namely, (1) No. of Residual Network Blocks (#ResNet Blocks), (2) No. of Filters (#Filters), and (3) No. of LSTM Cells (#LSTM Cells), which can, theoretically, be any value in the $\mathbb{R}^+$ domain, leading to an explosion of the designs that need to be explored under an unbounded design space. By considering the state-of-the-art model as the upper bound, we restrict the number of designs to be explored, thereby ensuring that the algorithm converges in finite time. Furthermore, since the exploration of the models is heavily dependent on the state of the art, any modifications to the block-level structure of the baseline model, including changes to the *block*, will affect the design space of the models (ψ) to be explored.

2.5 Deep Learning Model Training and Evaluation

The DNN models generated earlier need to be trained and evaluated on the constructed dataset, individually, before their real-world deployment. However, since the training and evaluation of each individual DNN in the design space is a compute-intensive and time-consuming task, first, we need to reduce the number of models generated, which we ensure by constraining the hardware requirements of the model (as discussed earlier in Sect. 2.2), and second, we need to quickly explore the design space of DNNs, to reduce the overall duration of the task. Exploration of the design space, in our framework, can be conducted either exhaustively or selectively, as discussed below:

(1) Exhaustive Exploration It requires each individual model of the design space to be trained and evaluated on the constructed dataset in order to determine the set of Pareto-optimal DNN models, which essentially trade off between output quality and hardware requirements. The hardware constraints imposed by the execution platform combined with the state-of-the-art imposed upper bound enable the framework to exhaustively explore the design space of DNN models in tens of GPU hours, as opposed to hundreds or thousands of hours in the case of unconstrained exploration. Therefore, when the complexity of the model, its parameter format, the number of weights and biases, and the variation in the hyper-parameters increase, it is recommended to selectively explore the design space to circumvent the exponential rise in design space models. Exhaustive exploration is primarily

included as a functionality to illustrate the efficacy of the selective exploration technique that has been discussed next.

(2) Selective Exploration It involves the effective selection, training, and evaluation of a small subset of the models in ψ in order to reduce the exploration time to a couple of GPU hours. Genetic algorithms utilize a cost function, which defines the optimization goal, to effectively obtain near-optimal solutions while reducing the exploration time for a wide range of real-world optimization problems [44]. The framework uses genetic algorithms that rely on the concepts of reproduction and evolution to select the models that need to be trained in each generation to create a new generation of models that have the potential to further optimize the cost function. The use of other meta-heuristic approaches to explore the design space is orthogonal to our use of genetic algorithms, which encompasses techniques like ant colony optimization [8] and simulated annealing [48], and can be incorporated into the framework, if essential.

In the selective exploration process, we start with an initial population of 30 random DNN models present in the design space, referred to as individuals, based on the recommendation of previous works [41] in order to obtain the best results. The genetic algorithms require the presence of a "chromosome," which encodes all the key neural architecture parameters ("genes") that can be varied to obtain the complete design space of DNN models. All the *genes* are stitched together to generate the *chromosome* string, which, when decoded, constructs a DNN model, or an individual, in the design space. Each individual is subsequently trained on the constructed dataset to evaluate its viability in terms of a *fitness value*, which enables it to compete with other individuals in the design space. The fitness value is estimated as the cost function when the decoded DNN model (M) exists in the design space (ψ) or is considered to be a `NULL` value otherwise and is discarded from the search. Next, on the basis of their fitness values, two individuals are selected to pass on their genes to the next generation while undergoing the process of mutation and crossover, which are essential reproduction principles. We ensure an ordered 0.4 crossover probability for a mating parent pair with a random crossover location in the pair's chromosomes. The next generation of the population, i.e., the offspring, has their parents' chromosomes exchanged from the start until the crossover point and is considered for exploration based on their fitness value. The offspring also have a mutation probability of 0.11 to enable a bit-flip in the chromosome, thereby ensuring a diverse population and enabling a comprehensive exploration of DNN models. The experiments are run to determine a population of 30 individuals in each generation, based on their fitness values, to create 5 consecutive iterations of offspring that can be trained and evaluated to determine the set of best-fit individuals (see Fig. 3).

By default, the framework includes the ability to explore the design space using the following recognized genetic algorithms: NSGA-II [6], Roulette Wheel [11], Tournament Selection [31], and SPEA-2 [51]. Likewise, other algorithms and heuristics can be incorporated into the framework, as discussed earlier. The time complexity of each algorithm determines the order of execution time required for

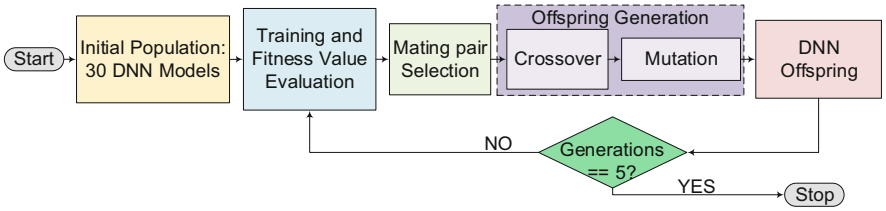


Fig. 3 Flow chart illustrating the selective design space exploration technique (adapted from [37])

exploring the design space ψ. If the size of the design space is considered to be N, the time complexity of the algorithms would be $O(N^2)$, $O(N * \log N)$, $O(N)$, and $O(N^2 * \log N)$, respectively. The efficacy of these algorithms, illustrated by the varying subset of individuals selected and evaluated, is discussed in Sect. 3 with the use-case. The genetic algorithms used by the framework require a cost function (ϕ) that needs to be specified by the system designer, which can be optimized to obtain the set of near-optimal network models (ω) for the explored design space. The weighted cost function used in this framework is

$$\phi = \alpha * Q + \beta * \left[1 - \frac{H}{H_{\max}}\right]$$

where $\alpha, \beta \in [0, 1]$ depict the weights for output quality (Q) and hardware requirements (H) of the model, respectively. $H_{\max}$ denotes the hardware requirements of the state-of-the-art baseline model. As discussed earlier, Q can be evaluated as Accuracy, Precision, Recall, or F1-score, whereas H can be estimated as the memory overhead or the number of floating-point operations for an inference. Other application-specific quality metrics or additional hardware requirements, such as the power consumption of the model or its energy requirements on the target platform, can also be included in the framework. The weights α and β depict the importance of the quality and hardware metrics, respectively, during the algorithm's exploration of the design space. Algorithm 1 discusses the pseudo-code for the weighted DNN model exploration technique deployed in the framework. Given (1) the inputs (design space (ψ), weights for the cost function (α, β), and the hardware requirement for the state-of-the-art model ($H_{\max}$)) and (2) quality and hardware constraints (Q_{Const}, H_{Const}), the weighted DNN model exploration algorithm generates a set of DNN models ω that satisfies the quality and hardware constraints of the application. The *ExplorationAlgorithm* function call in Line 10 can call any of the selective exploration algorithms (genetic algorithms) or the exhaustive exploration technique discussed earlier. Table 2 illustrates an overview of the symbols and denotations used in this chapter.

Algorithm 1 Weighted DNN model exploration

Input: ψ, α, β, $H_{\max}$
Constraints: Q_{Const}, H_{Const}
Output: ω
$H = []$;
for M **in** ψ **do**
  if $HardwareRequirements(M) \leq H_{Const}$ **then**
    $H.append(HardwareRequirements(M))$;
  else
    $\psi.remove(M)$;
  end if
end for
$\phi = \alpha * Q + \beta * \left[1 - \frac{H}{H_{\max}}\right]$
$\omega = ExplorationAlgorithm(\phi, \psi)$;
for DNN **in** ω **do**
  if $Q.(\text{DNN}) < Q_{Const}$ **then**
    $\omega.remove(\text{DNN})$;
  end if
end for

Table 2 Overview of the symbols used in this work along with their denotations [37]

Symbol	Denotation	Symbol	Denotation
Q	Model quality	N	Size of design space (ψ)
Q_{Const}	User quality constraint	ϕ	Cost function to be optimized
A	Accuracy of the model	ω	Output set of near-optimal DNN models
P	Precision of the model	α	Weight for output quality Q
R	Recall of the model	β	Weight for hardware requirement H
F	F1-score of the model	M	DNN model in ψ
B	Memory overhead of the model	H_{Const}	Platform's hardware constraint
FP	No. of floating-point operations reqd. by the model	$H_{\max}$	Hardware requirements of the state-of-the-art model
ψ	Design space of DNN models	H	Hardware requirements of the model

2.6 Model Compression

The framework also includes the capability of further reducing the model's hardware requirements through the means of compression techniques like pruning and quantization. Besides neural architecture search approaches, model compression techniques have proven to be highly successful in reducing the hardware requirements of the model while retaining output quality [12].

2.6.1 Pruning

As the name conveys, the core concept of this approach involves identifying less-important parameters of the model, such as the weights, kernels, biases, or even

neurons or layers, and eliminating them to further reduce the hardware requirements of the DNN model, increasing their deployability in edge platforms. Eliminating the model parameters reduces many of its requirements, such as memory overhead of the model and the number of floating-point operations required for an inference, which tend to further improve performance and reduce energy consumption on the target platform during inference. The pruned model is subsequently retrained on the constructed dataset to ensure that the model achieves an output quality similar to that of the original unpruned model obtained from the design space. The framework integrates the pruning techniques presented in [3, 12, 26, 28, 30] to provide the system designer with a range of options that can be implemented in order to meet the application requirements based on the DNN model's capabilities. For example, the technique proposed in [12] determines the lowest $x\%$ of weights, based on their absolute magnitude, in each individual layer of the model and eliminates them, followed by a retraining stage, as discussed earlier, to achieve an accuracy similar to the original model. Whereas the technique presented in [30] sorts the complete set of weights in the model to iteratively eliminate the lowest $x\%$ of overall weights in each iteration, regardless of the layer, followed by model retraining to achieve original model accuracy. Section 3 illustrates an overview of the benefits of pruning DNN models obtained using this approach. Incorporating other pruning techniques into the framework can be easily achieved as long as the new technique complies with the original interfaces of standard pruning techniques.

2.6.2 Quantization

The model parameters are usually stored in a floating-point format requiring 32 bits, leading to a large memory overhead on the execution platform. Accessing each floating-point parameter from memory requires increased access latency and energy consumption, as opposed to traditional 8-bit or 16-bit integers. Likewise, a high-precision floating-point addition operation requires nearly an order of magnitude more energy as opposed to a 32-bit integer ADD operation [13]. Hence, approaches that can be used to reduce the precision from 32 bits to 16 or 8 bits, through the process of quantization, can be used to substantially reduce the hardware requirements of the model. Quantization techniques can be implemented to further reduce the precision of the DNN model to less than 8 bits, by analyzing its trade-off with output quality for the target application. The process involves the construction of 2^p clusters, where p stands for the number of quantized bits, using the k-means algorithm, which evaluates the parameters in each layer of the DNN model. Once the clusters are determined, equally spaced values are allocated to each cluster ranging from minimum to maximum value for corresponding cluster weights composed of all zeros to all ones, respectively. For simplicity, all layers in the DNN model are quantized with the same number of bits. Similar to pruning, other quantization techniques can be incorporated into the framework as long as the new technique complies with the original interfaces of standard quantization.

Based on recommendations from the studies presented in [12] and from exhaustive experimentation, the optimal approach for minimizing the hardware requirements of the model requires pruning the selected DNN model obtained from the design space, followed by model quantization, to eliminate the redundant parameters and subsequently reduce parameter precision, respectively.

3 Case Study: Bio-signal Anomaly Detection

We present the efficacy of the framework by deploying it to generate, explore, and compress a wide range of DNN models for our use-case: ECG Bio-signal processing. We explore 5 different sub-cases as a part of this study:

- $\mathbf{UC}_1$: Binary Classification: [Normal, Anomaly]
- $\mathbf{UC}_2$: Multi-class Classification: [Normal, Premature Ventricular Contraction, Other Anomaly]
- $\mathbf{UC}_3$: Multi-class Classification: [Normal, Bundle Branch Block, Other Anomaly]
- $\mathbf{UC}_4$: Multi-class Classification: [Normal, Atrial Anomaly, Ventricular Anomaly, Other Anomaly]
- $\mathbf{UC}_5$: Multi-class Classification: [Normal, Ventricular Fibrillation, Other Anomaly]

Hannun et al. [14] proposed a deep neural network model architecture that can differentiate between 12 classes of ECG signals, evaluated on their private dataset. This model is considered to be the current state of the art in ECG signal classification and is the baseline model of this use-case. The primary block used in [14] is adopted in this use-case to generate the design space of DNN models for each of the 5 different sub-cases discussed above. The input and output layers have been modified to consider the data from the open-source ECG dataset adopted in this case study to process and categorize them into the required output classes. The default model of the DNN is modified to include LSTM cells at the end, enabling accuracy improvements in cases where the number of feature extraction layers is substantially reduced during neural architecture search.

3.1 Experimental Setup

Dataset Construction For this bio-signal processing case study, the MIT-BIH dataset [32] is used to construct the required datasets by collecting a 256-sample window, which is subsequently assigned a label corresponding to the original labels of the parent dataset. The 41 different annotations of the parent dataset are categorized as one of the labels for each sub-case to ensure coherence in the dataset. To construct an enriched dataset that can provide the relevant information to the DNN model and enable it to learn effectively across labels like ventricular tachy-

cardia and ventricular fibrillation, the framework also includes the CU Ventricular dataset [33] during the construction of the custom datasets. The constructed datasets are split in the ratio of 7:1:2 to generate the training, validation, and testing datasets, respectively.

Neural Architecture Parameters An overview of the modified DNN architecture used in this case study is presented in Fig. 4. Therefore, the three primary neural architecture parameters that can be varied to generate the DNN model design space are (1) #ResNet Blocks, (2) #Filters, and (3) #LSTM Cells. The ResNet blocks are made of 1D convolutional layers, batch normalization, ReLU activation blocks, and dropout layers, as illustrated in Fig. 4, and can vary between 0 and 15. The number of filters, of size 16, in each convolution layer is determined as a function of z–$[32 \times 2^z]$—where z starts from the value of 0 and is increased by 1 after every y ResNet blocks (y varies from 1 to 4 in increments of 1, i.e., $y \in \{1, 2, 3, 4\}$). The number of LSTM cells is varied as 2^x, where $x \in \{4, 5, 6, 7, 8\}$.

By varying these parameters, we can generate up to 320 different DNN models as part of a given application's design space. However, due to the hardware limitation imposed by the state-of-the-art model, the framework reduces the number of models explored to 135, thereby drastically reducing the exploration time.

Selective Exploration Figure 5 presents the composition of the chromosome used by the genetic algorithms in this case study. The chromosome is a binary string of size 9, which encodes the key neural architecture parameters discussed

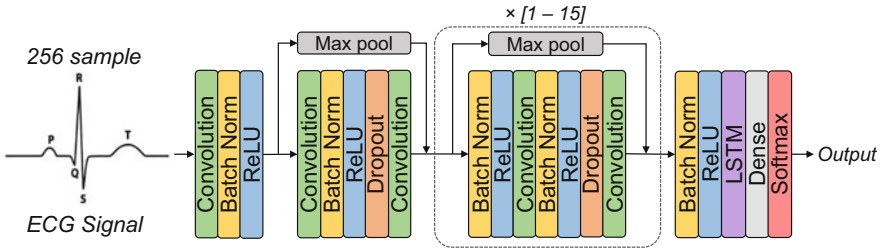


Fig. 4 Modified state-of-the-art DNN architecture used in the case study (adapted from [37])

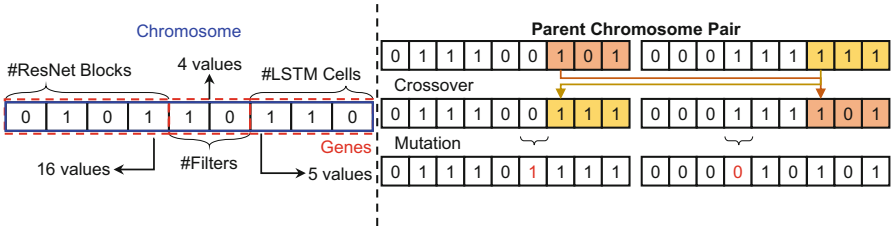


Fig. 5 The composition of the chromosome used by the genetic algorithms in this case study; example of chromosomal crossover and mutation (adapted from [37])

Table 3 Optimal hyper-parameter values used for training the DNN Models [37]

Hyper-parameter	Optimal value
Weights initialization	He et al. [16]
Adam optimizer [22]	$\beta_1 = 0.9$, $\beta_2 = 0.999$
Learning rate	0.001
Batch size	128
Dropout	0.2

Table 4 The optimal values of the constants used by the genetic algorithms [37]

Constant	Optimal value
Population size	30
Chromosome length	9
Generation size	5
Mutation probability	0.11
Crossover probability	0.4

above as genes. The chromosome can therefore construct $2^{10} - 1$, or 1023, DNN models in design space for each of the sub-cases. However, since only 5 of the 7 possible #LSTM cell values lead to valid DNN model architectures, we can directly eliminate invalid configurations not present in ψ. Once the parent chromosome pair is selected, based on their fitness value, for generating offspring, they undergo the process of crossover to exchange genes and undergo potential mutation to introduce diversity, as illustrated in Fig. 5.

Tool Flow The TensorFlow platform is used for the implementation of the DNN models in the Python programming environment with the help of the Keras package. The DNN models are trained over multiple iterations with varying hyper-parameter values to determine the ones that offer maximum accuracy. Table 3 presents the optimal values of these hyper-parameters. The DEAP library [5] in Python contains implementations of the four genetic algorithms that are used in the case study. Table 4 presents the constants and their optimal values, which are used by the genetic algorithms during selective exploration of the design space. The exploration stage is executed on a GPU server composed of four i9 CPUs and 8 Nvidia RTX 2080 GPUs, with the early stopping mechanism enabled. The selected models are then trained using the custom dataset for quality evaluation and studying the trade-off with their hardware requirement.

3.2 Exhaustive Exploration

Figure 6 illustrates the results and trade-offs between quality and memory of exhaustively exploring the models in the UC_3 design space. The Pareto-frontier of the complete design space, which connects all the Pareto-optimal DNN models and offers the best trade-off between quality and memory, is illustrated by Ⓐ. Label Ⓑ,

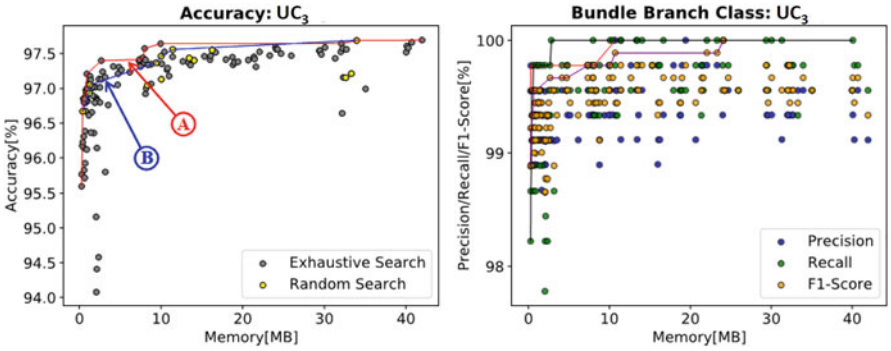


Fig. 6 Analysis of exhaustive exploration on the UC_3 design space (adapted from [37])

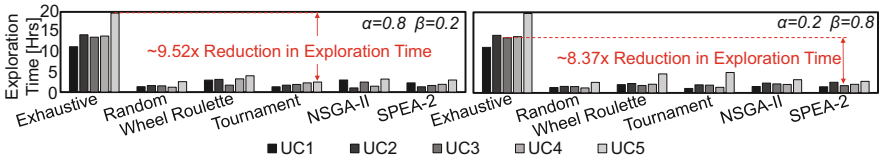


Fig. 7 Analyzing the time benefits of the selective exploration approach (adapted from [37])

on the other hand, depicts the pseudo-Pareto-frontier constructed using the set of optimal designs obtained by random exploration (i.e., baseline). The large number of inter-dependent parameters in DNN models leads to the situation where the designs depicted in Ⓐ and Ⓑ are very similar to each other. Exhaustive exploration of the design space has led to the successful identification of a DNN model that can reduce the overhead by ∼30 MB for a quality loss of less than 0.5%. However, due to the time required for such exhaustive exploration, it might be more suitable to obtain a near-optimal point that offers similar trade-offs using selective exploration for much less time. The variance in the Precision, Recall, and F1-score of the model for the specialized bundle branch class indicates suitability of the framework to impose a quality constraint on these metrics as well.

3.3 Selective Exploration: Time Benefits

The primary benefit of the selective exploration process is the reduction in time required to search the design space of DNN models with the use of genetic algorithms. Figure 7 illustrates the reduction in time for the five different use-cases when explored using the genetic algorithms, as opposed to exhaustive exploration. We have also varied the weights used by the cost function (α, β) to emphasize that changing weights does not drastically modify the time needed for exploring the design space. Randomly selecting and evaluating 10% of the DNN models

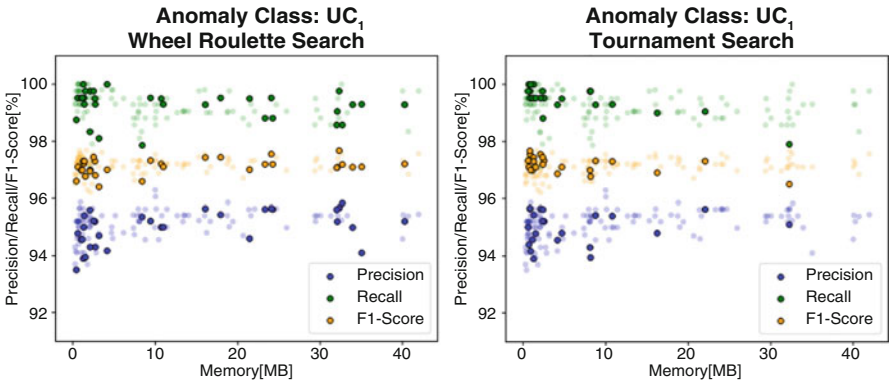


Fig. 8 Evaluation of the quality and memory trade-offs for the models obtained using wheel roulette search and tournament search on the UC_1 design space (adapted from [37])

in the design space acts as baseline comparison for the selective exploration strategies discussed in this chapter, with practically no algorithmic overhead. The selective exploration strategies achieve 9× reduction in exploration time, on average, as opposed to the bounded exhaustive exploration strategy. The use of genetic algorithms for exploring the search space is highly beneficial in scenarios where the application requires the use of highly complex deep neural networks with tens of millions of parameters. Exhaustively training and evaluating each model in the design space, in such instances, would lead to exploration time overheads of hundreds of GPU hours, which might not be feasible for the system designer.

3.4 Selective Exploration: Efficacy and Analysis

The primary benefit of using genetic algorithms, reduction in exploration time, was already discussed earlier. In this subsection, we focus on the capability of the genetic algorithms in exploring the design space and analyze their efficacy. The results of these experiments for the UC_1 and UC_5 design spaces, with α and β set to 0.5 are illustrated in Figs. 8 and 9, respectively. The transparent points in these results depict the models obtained from the design space using exhaustive exploration, enabling us to determine the efficacy of the genetic algorithms. The genetic algorithms are highly successful at identifying a set of near-optimal DNN models without traversing the complete design space, especially in cases where the accuracy improvements or the hardware memory reductions are minimal when compared to the Pareto-optimal design. The number of models evaluated by the NSGA-II and SPEA-2 algorithm is smaller than their counterparts. Note that a significant number of models in the UC_5 design space exhibit 0% quality due to

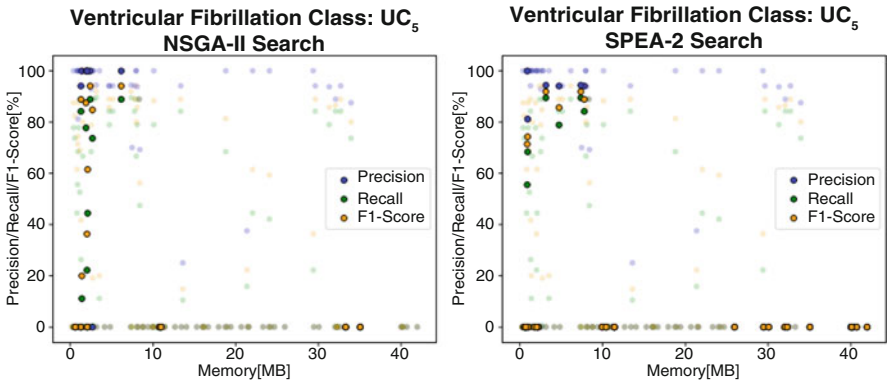


Fig. 9 Evaluation of the quality and memory trade-offs for the models obtained using NSGA-II search and SPEA-2 search on the UC_5 design space (adapted from [37])

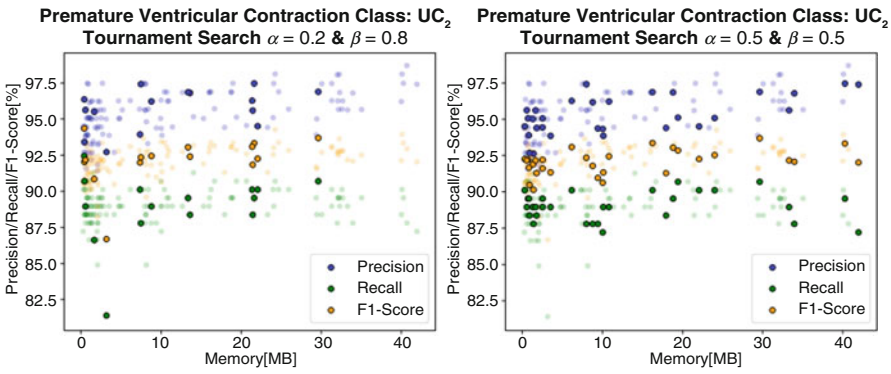


Fig. 10 Evaluation of the weighted exploration technique on the UC_2 design space using tournament search with two different weight values for the cost function (adapted from [37])

the inherent differences in the number of samples of class Ventricular Fibrillation, leading to a bias against it.

3.5 Selective Exploration: Weighted Exploration

Next, we discuss a subset of the results obtained when exploring the design space using different weights for the cost function (ϕ), which is used by the genetic algorithms. Figure 10 illustrates the results when the algorithm focuses on optimizing (1) memory alone ($\alpha = 0.2, \beta = 0.8$) or (2) memory and accuracy ($\alpha = 0.5, \beta = 0.5$), for a model in the design space of UC_2. Similarly, Fig. 11 presents the results when the algorithm optimizes for (1) memory alone ($\alpha = 0.2, \beta = 0.8$) or (2) accuracy alone ($\alpha = 0.5, \beta = 0.5$), in the design space of

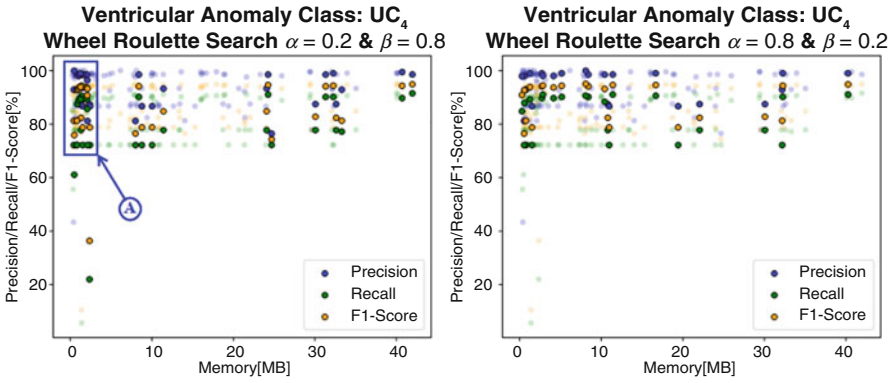


Fig. 11 Evaluation of the weighted exploration technique on the UC_4 design space using wheel roulette search with two different weight values for the cost function (adapted from [37])

UC_4. The weighted parameterization of the cost function is highly beneficial in guiding the genetic algorithms to optimize for the required parameter, i.e., memory or quality or both. For example, as illustrated by Ⓐ in Fig. 11, the algorithm focuses on optimizing memory, thereby selecting a large number of points with minimal overhead. Similarly, when the optimization goal is either accuracy only (see Fig. 11) or memory and accuracy (see Fig. 10), appropriate models are selected for evaluation by the algorithm.

3.6 Pruning and Quantization: Compression Efficacy and Receiver Operating Characteristics

Next, we select three near-optimal models obtained from the UC_4 design space to evaluate the efficacy of our pruning and quantization techniques. Without loss of generality and for the purpose of illustration, the three models, Z_1, Z_2, and Z_3, focus on accuracy alone, trade-off between accuracy and memory, or memory alone, respectively. Pruning, alone, is quite effective in reducing the memory by nearly 40% for roughly 0.15% increase in accuracy. This contra-indicative improvement in accuracy can be attributed to the over-redundant parameterization of the network model, which is eliminated by pruning. Due to similar reasons, model Z_1 can tolerate pruning of a significant percentage of parameters before exhibiting accuracy losses, as opposed to the other models that are not as over-parameterized. Likewise, quantization can drastically reduce the memory requirements of the network by lowering the precision of the parameters storing the weights and biases. This process can further reduce the memory requirements by up to 5×, as opposed to FP32 precision, for <0.1% quality loss. Combining both these approaches can reduce the memory by a factor of 53× for <0.2% loss in quality (Fig. 12).

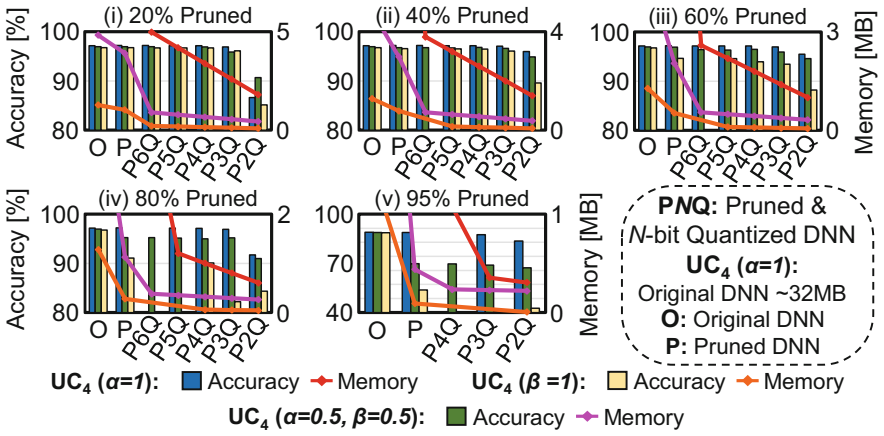


Fig. 12 Compression of the near-optimal UC_4 design space DNN models (adapted from [37])

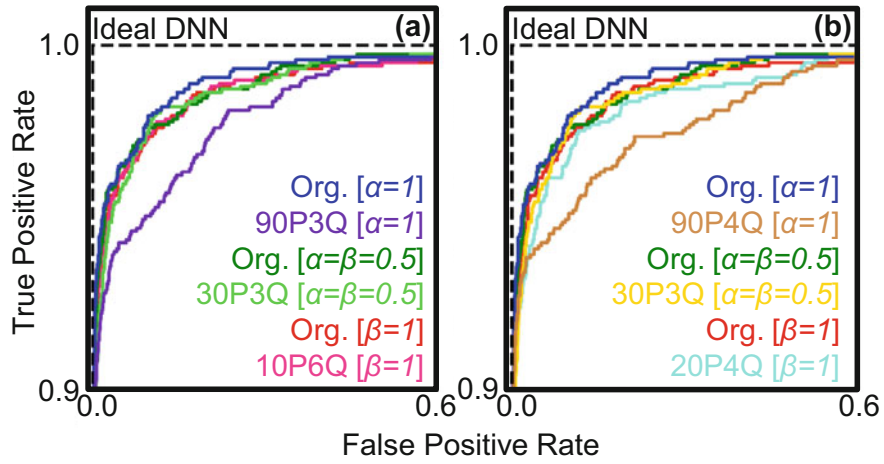


Fig. 13 ROC evaluation of the similar DNN models from the UC_4 design space [37]

Next, the receiver operating characteristics of selected pruned and quantized models are evaluated to determine their behavior, when compared to the original model obtained from the design space. The evaluation is completed for two scenarios:

(a) With a memory constraint of 0.5MB, maximize the model accuracy.
(b) With an accuracy constraint of 96.7%, minimize the model's memory.

As can be observed from the results presented in Fig. 13, model Z_1 exhibits the best operating characteristics, with Z_2 and Z_3 not lagging far behind. The maximum accuracy models, which are subsequently pruned and quantized, exhibit the worst operating characteristics, far behind the pruned and quantized models of Z_2 and Z_3,

albeit with similar accuracy metrics. This makes the latter two models more suitable for deployment in the real world on constrained edge devices like wearables.

4 Conclusion

Healthcare is one of the world's largest industries requiring a lot of investments, man power, training, and expertise, especially with the rising older population (above the age of 65) in most western nations, a significant percentage of whom require continuous support and healthcare. This requires scientists and researchers to develop technologies that can cater to the requirements of global healthcare systems with currently available technologies. Deep learning, which is currently at the forefront of major technological innovation, has proven to be highly effective in various healthcare domains like medical imaging, electronic health data analytics, precision medicine, and drug discovery. In this chapter, an embedded neural architecture search and model compression framework was discussed to enable their deployment in healthcare applications. The framework considers the user requirements, in terms of quality, or specifications, like type of output, and hardware constraints of the target platform to effectively search the design space of DNN models to generate a set of near-optimal DNNs suitable for the application. Besides achieving a 53× reduction in memory, models optimized for both accuracy and memory during the design space search were observed to have better operating characteristics, even when compressed. The framework is open source and available online at https://bionetexplorer.sourceforge.io/.

Acknowledgments This work has been supported by the Doctoral College Resilient Embedded Systems, which is run jointly by the TU Wien's Faculty of Informatics and the UAS Technikum Wien. We would also like to acknowledge the coauthors (collaboration partners) of our BioNetExplorer journal paper, which is discussed in this chapter.

References

1. Aggarwal, R., Sounderajah, V., Martin, G., Ting, D.S., Karthikesalingam, A., King, D., Ashrafian, H., Darzi, A.: Diagnostic accuracy of deep learning in medical imaging: a systematic review and meta-analysis. NPJ Digit. Med. **4**(1), 1–23 (2021)
2. Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., Ayyash, M.: Internet of things: a survey on enabling technologies, protocols, and applications. IEEE Commun. Surv. Tutorials **17**(4), 2347–2376 (2015)
3. Anwar, S., Hwang, K., Sung, W.: Structured pruning of deep convolutional neural networks. ACM J. Emerg. Technol. Comput. Syst. **13**(3), 1–18 (2017)
4. Bloomberg: These are the economies with the most (and least) efficient health care. [Online Link]
5. De Rainville, F.M., Fortin, F.A., Gardner, M.A., Parizeau, M., Gagné, C.: DEAP: A python framework for evolutionary algorithms. In: Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation, pp. 85–92 (2012)

6. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-ii. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002)
7. Dincer, A.B., Celik, S., Hiranuma, N., Lee, S.I.: DeepProfile: Deep learning of cancer molecular profiles for precision medicine. BioRxiv, p. 278739 (2018)
8. Dorigo, M., Birattari, M., Stutzle, T.: Ant colony optimization. IEEE Comput. Intell. Mag. **1**(4), 28–39 (2006)
9. Esteva, A., Robicquet, A., Ramsundar, B., Kuleshov, V., DePristo, M., Chou, K., Cui, C., Corrado, G., Thrun, S., Dean, J.: A guide to deep learning in healthcare. Nat. Med. **25**(1), 24–29 (2019)
10. Gibson, E., Li, W., Sudre, C., Fidon, L., Shakir, D.I., Wang, G., Eaton-Rosen, Z., Gray, R., Doel, T., Hu, Y., et al.: NiftyNet: a deep-learning platform for medical imaging. Comput. Methods Programs Biomed. **158**, 113–122 (2018)
11. Goldberg, D.E.: Optimization, and machine learning. Genetic algorithms in Search (1989)
12. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. arXiv preprint arXiv:1510.00149 (2015)
13. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in Neural Information Processing Systems, vol. 28 (2015)
14. Hannun, A.Y., Rajpurkar, P., Haghpanahi, M., Tison, G.H., Bourn, C., Turakhia, M.P., Ng, A.Y.: Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network. Nat. Med. **25**(1), 65–69 (2019)
15. Hassaballah, M., Awad, A.I.: Deep Learning in Computer Vision: Principles and Applications. CRC Press, Boca Raton (2020)
16. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: surpassing human-level performance on ImageNet classification. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1026–1034 (2015)
17. Horst, F., Lapuschkin, S., Samek, W., Müller, K.R., Schöllhorn, W.I.: Explaining the unique nature of individual gait patterns with deep learning. Sci. Rep. **9**(1), 1–13 (2019)
18. Huang, S.C., Pareek, A., Seyyedi, S., Banerjee, I., Lungren, M.P.: Fusion of medical imaging and electronic health records using deep learning: a systematic review and implementation guidelines. NPJ Digit. Med. **3**(1), 1–9 (2020)
19. Islam, S.R., Kwak, D., Kabir, M.H., Hossain, M., Kwak, K.S.: The internet of things for health care: a comprehensive survey. IEEE Access **3**, 678–708 (2015)
20. Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., et al.: Highly accurate protein structure prediction with AlphaFold. Nature **596**(7873), 583–589 (2021)
21. Kamath, U., Liu, J., Whitaker, J.: Deep Learning for NLP and Speech Recognition, vol. 84. Springer, Berlin (2019)
22. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization (2014). arXiv preprint arXiv:1412.6980
23. Korkalainen, H., Aakko, J., Nikkonen, S., Kainulainen, S., Leino, A., Duce, B., Afara, I.O., Myllymaa, S., Töyräs, J., Leppänen, T.: Accurate deep learning-based sleep staging in a clinical population with suspected obstructive sleep apnea. IEEE J. Biomed. Health Inform. **24**(7), 2073–2081 (2019)
24. Lavecchia, A.: Deep learning in drug discovery: opportunities, challenges and future prospects. Drug Discovery Today **24**(10), 2017–2032 (2019)
25. Lee, J.G., Jun, S., Cho, Y.W., Lee, H., Kim, G.B., Seo, J.B., Kim, N.: Deep learning in medical imaging: general overview. Korean J. Radiol. **18**(4), 570–584 (2017)
26. Lin, J., Rao, Y., Lu, J., Zhou, J.: Runtime neural pruning. In: Advances in Neural Information Processing Systems, vol. 30 (2017)
27. Lundervold, A.S., Lundervold, A.: An overview of deep learning in medical imaging focusing on MRI. Zeitschrift für Medizinische Physik **29**(2), 102–127 (2019)
28. Luo, J.H., Wu, J., Lin, W.: ThiNet: A filter level pruning method for deep neural network compression. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 5058–5066 (2017)

29. Manyika, J., Chui, M., Bughin, J., Dobbs, R., Bisson, P., Marrs, A.: Disruptive technologies: advances that will transform life, business, and the global economy, vol. 180. McKinsey Global Institute San Francisco (2013)
30. Marchisio, A., Hanif, M.A., Martina, M., Shafique, M.: PruNet: Class-blind pruning method for deep neural networks. In: 2018 International Joint Conference on Neural Networks (IJCNN), pp. 1–8. IEEE (2018)
31. Miller, B.L., Goldberg, D.E., et al.: Genetic algorithms, tournament selection, and the effects of noise. Complex Syst. **9**(3), 193–212 (1995)
32. Moody, G.B., Mark, R.G.: The impact of the MIT-BIH arrhythmia database. IEEE Eng. Med. Biol. Mag. **20**(3), 45–50 (2001)
33. Nolle, F., Badura, F., Catlett, J., Bowser, R., Sketch, M.: CREI-GARD, a new concept in computerized arrhythmia monitoring systems. Comput. Cardiol. **13**, 515–518 (1986)
34. Our World in Data: Life expectancy. [Online Link]
35. Policy Advice: The state of healthcare industry—statistics for 2021. [Online Link]
36. Porumb, M., Stranges, S., Pescapè, A., Pecchia, L.: Precision medicine and artificial intelligence: a pilot study on deep learning for hypoglycemic events detection based on ECG. Sci. Rep. **10**(1), 1–16 (2020)
37. Prabakaran, B.S., Akhtar, A., Rehman, S., Hasan, O., Shafique, M.: BioNetExplorer: architecture-space exploration of biosignal processing deep neural networks for wearables. IEEE Internet Things J. **8**(17), 13251–13265 (2021)
38. Preuer, K., Klambauer, G., Rippmann, F., Hochreiter, S., Unterthiner, T.: Interpretable deep learning in drug discovery. In: Explainable AI: Interpreting, Explaining and Visualizing Deep Learning, pp. 331–345. Springer, Berlin (2019)
39. Rajkomar, A., Oren, E., Chen, K., Dai, A.M., Hajaj, N., Hardt, M., Liu, P.J., Liu, X., Marcus, J., Sun, M., et al.: Scalable and accurate deep learning with electronic health records. NPJ Digit. Med. **1**(1), 1–10 (2018)
40. Ramsundar, B., Eastman, P., Walters, P., Pande, V.: Deep Learning for the Life Sciences: Applying Deep Learning to Genomics, Microscopy, Drug Discovery, and More. O'Reilly Media (2019)
41. Reeves, C., Rowe, J.E.: Genetic Algorithms: Principles and Perspectives: A Guide to GA Theory, vol. 20. Springer, Berlin (2002)
42. Rifaioglu, A.S., Atas, H., Martin, M.J., Cetin-Atalay, R., Atalay, V., Doğan, T.: Recent applications of deep learning and machine intelligence on in silico drug discovery: methods, tools and databases. Briefings Bioinform. **20**(5), 1878–1912 (2019)
43. Sahiner, B., Pezeshk, A., Hadjiiski, L.M., Wang, X., Drukker, K., Cha, K.H., Summers, R.M., Giger, M.L.: Deep learning in medical imaging and radiation therapy. Med. Phys. **46**(1), e1–e36 (2019)
44. Sastry, K., Goldberg, D., Kendall, G.: Genetic algorithms. In: Search Methodologies, pp. 97–125. Springer, Berlin (2005)
45. Shickel, B., Tighe, P.J., Bihorac, A., Rashidi, P.: Deep EHR: a survey of recent advances in deep learning techniques for electronic health record (EHR) analysis. IEEE J. Biomed. Health Inform. **22**(5), 1589–1604 (2017)
46. Solares, J.R.A., Raimondi, F.E.D., Zhu, Y., Rahimian, F., Canoy, D., Tran, J., Gomes, A.C.P., Payberah, A.H., Zottoli, M., Nazarzadeh, M., et al.: Deep learning for electronic health records: a comparative review of multiple deep neural architectures. J. Biomed. Inform. **101**, 103337 (2020)
47. United Nations: World population ageing. [Online Link]
48. Van Laarhoven, P.J., Aarts, E.H.: Simulated annealing. In: Simulated Annealing: Theory and Applications, pp. 7–15. Springer, Berlin (1987)
49. Wang, F., Casalino, L.P., Khullar, D.: Deep learning in medicine—promise, progress, and challenges. JAMA Internal Med. **179**(3), 293–294 (2019)

50. Xiao, C., Choi, E., Sun, J.: Opportunities and challenges in developing deep learning models using electronic health records data: a systematic review. J. Am. Med. Inform. Assoc. **25**(10), 1419–1428 (2018)
51. Zitzler, E., Laumanns, M., Thiele, L.: Spea2: improving the strength pareto evolutionary algorithm. In: TIK-Report, vol. 103 (2001)

Robust Machine Learning for Low-Power Wearable Devices: Challenges and Opportunities

Ganapati Bhat, Dina Hussein, and Nuzhat Yamin

1 Introduction

Wearable devices that integrate multiple sensors, processors, and communication technologies have the potential to transform multiple facets of human life: public health, fitness, and the way we interact with the environment. For instance, wearable devices are being widely used by the general public to monitor their activity levels and steps. Wearable devices are being also used to track vital signs and rehabilitation in patients with chronic disorders [24, 60]. More broadly, Internet of Things (IoT) devices are being deployed to enable interesting applications such as smart cities, environmental monitoring, digital agriculture, and wide area sensing [4, 10, 37, 88, 95]. Overall, the promise shown by wearable devices has led to increased research attention from a number of communities in both academia and industry.

Wearable devices typically collect sensor data at runtime and process the data locally or in an edge node to fulfill user and application requirements. For example, data from motion sensors are processed at runtime to identify the activities of the users. Early implementations of wearable devices used statistical or analytical models to identify and monitor the parameters of interest [6]. However, their limited processing capability restricted the complexity of applications implemented on wearable devices. Recent advances in machine learning, low-power sensors, and embedded microprocessors have enabled wearable devices to perform higher degree of processing at the edge [8, 40]. As a result, wearable devices are being used to implement interesting and high-impact applications such as fall detection, arrhythmia monitoring, and movement disorder diagnosis [24, 60, 77]. The applications collect sensor data at runtime and process it on the device using machine learning

G. Bhat (✉) · D. Hussein · N. Yamin
School of EECS, Washington State University, Pullman, WA, USA
e-mail: ganapati.bhat@wsu.edu; dina.hussein@wsu.edu; nuzhat.yamin@wsu.edu

S. Pasricha, M. Shafique (eds.), *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing*, https://doi.org/10.1007/978-3-031-40677-5_3

algorithms. Processing the data on the device ensures that the raw data is not transmitted outside the device, thus ensuring the privacy of the user data. At the same time, wearable devices must operate under tight energy budgets due to their small battery capacities. Moreover, the applications on wearable devices must be robust to changes in user data patterns and other uncertainties in the environment. To this end, there is a need to ensure that the machine learning algorithms employed at the edge on wearable devices are reliable while satisfying the low power requirements.

Edge machine learning algorithms in wearable devices face a number of challenges in real-world usage. The first major challenge is that the distribution of sensor data during real world usage may not match the distribution of the data during training. This is because the sensor data patterns of the users available during training may be different from the users in the real-world deployment. Even for the same set of users, the activity patterns (e.g., walking style) may change with time. The orientation of the sensor data may also change with time due to long-term usage or user error. For instance, the user may mount a wearable sensor facing the front, whereas the machine learning algorithm is trained with the sensor mounted facing the back. As a result, the distribution of the sensor data changes, which may reduce the accuracy of the application. Second, due to energy constraints in wearable devices, the applications may experience missing data samples. In particular, for wearable devices that employ energy harvesting, the stochastic nature of ambient energy may result in sensors turning off due to lack of energy, which in turn leads to missing samples. The missing samples will cause the accuracy to drop since typical machine learning on wearable devices are not trained to handle missing data. Therefore, recent research has focused on developing approaches that are able to handle sensor data shift and missing data to provide reliable applications on wearable devices.

The goal of this book chapter is to first summarize the state of the art in edge machine learning for wearable devices. Specifically, we present the typical flow in edge machine learning development for wearable devices and review recently implemented applications on wearable devices. Next, we review the robustness challenges for edge machine learning algorithms in wearable devices including energy limitations, sensor data shift, and missing data. We also present a summary of recent approaches for handling the robustness challenges in wearable devices followed by future opportunities in reliable machine learning for wearable devices. Moreover, we present a case study with a human activity recognition (HAR) application to demonstrate the effects of missing sensor data on the application accuracy. Finally, we perform a design space exploration with generative adversarial networks to develop a HAR classifier that is robust to missing data.

The rest of this chapter is organized as follows. Section 2 describes the typical flow of implementing edge machine learning in wearable devices and reviews some representative applications. Next, we present the requirements of robust machine learning and recent approaches for robustness in Sect. 3. Finally, Sect. 4 presents our HAR case study before concluding in Sect. 5.

2 Edge Machine Learning in Wearable Devices

Wearable technology is being used to implement a number of interesting applications for health and activity monitoring. It is also being used in gesture recognition and human–computer interaction applications. The general wearable application development goes through four major stages: data collection, feature design, and machine learning algorithm design, as shown in Fig. 1. In the following, we describe each of these steps briefly before providing the details of edge and on-device machine learning.

Data Collection Collection of sensor data that is representative of the application requirements is one of the first steps in the development of wearable applications. The data collection is typically performed in laboratory settings where subjects wear the sensors and perform the activities of interest. For instance, in a fitness monitoring application, sensor data are collected when users run, walk, and stand [112]. The laboratory settings make it easy to label the data and create a dataset for supervised learning. At the same time, obtaining data in controlled settings and labeling the data is time consuming and expensive. Therefore, it is also common to collect unsupervised data in free-living environments for long-term monitoring. For example, the Actitracker [58] dataset uses a smartphone application to collect activity data outside laboratory settings. Indeed, for some applications, such as arrhythmia monitoring, where the occurrences of arrhythmia events are rare, it is crucial to monitor the data over a long period to ensure that the rare events are captured. Data collection is an important step in wearable devices because the quality of the data determines the generalizability and effectiveness of the machine learning algorithms for the wearable applications.

Feature Design The next step after data collection is to design features that capture the behavior of the activities and the parameters of interest to implement

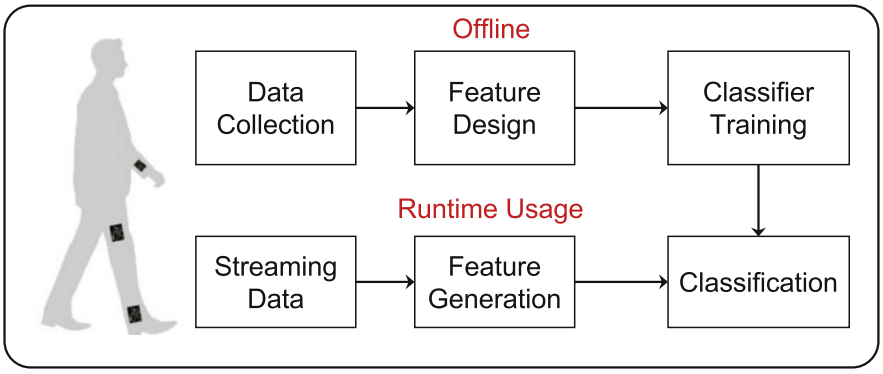


Fig. 1 Typical data flow and steps in designing machine learning algorithms in edge architectures for wearable devices

the wearable applications. Specifically, the features and the supervised labels are used to train the edge machine learning algorithms in the wearable device. Early implementations of wearable applications used handcrafted features as a function of parameters being monitored. For example, the activity recognition application uses the mean, kurtosis, and entropy as features for training a support vector machine classifier [2]. Similarly, the approach in [78] uses time and frequency domain features to train the machine learning algorithm to identify activity intensity. Recently, neural networks are being widely deployed on wearable devices to process the sensor data. The neural network approaches typically use the raw sensor data as the input and implicitly generate the features. For example, the approach in [53] directly uses the data from accelerometers in neural networks to recognize human activities. This ensures that the application designers do not have to handcraft features for the application.

Machine Learning Algorithm Design The last step in the design of applications for using wearable devices is the training of machine learning algorithms. This is a critical step since the accuracy and quality of service of the application depend on the effectiveness of the machine learning algorithms. Wearable devices present some unique challenges to the development of effective machine learning algorithms due to their processing, power consumption, and memory capabilities. To this end, the following sections detail the general model for edge machine learning for wearable devices, challenges for edge machine learning, and recent approaches to address the challenges. Then, we describe some recent examples of edge machine learning in wearable devices.

2.1 Edge Machine Learning Architectures

Edge machine learning architectures in wearable devices can be broadly classified into three categories, as shown in Fig. 2. The first category performs all the computations on the wearable sensors or a microcontroller that is integrated on the device. This model ensures that the raw data from the sensors are not transferred outside the wearable device. At the same time, the microcontrollers on the device may not be able to perform complex computations, and therefore, the second model uses a host device, such as a smartphone or a laptop, to process the sensor data. The wearable device wirelessly transfers the data to the smartphone so that further processing can be performed as per the application requirements. The high processing capability in smartphones or laptops enables the application to run more complex algorithms with higher computational requirements. Finally, in cases where the capabilities of the host device are not sufficient, the wearable device transfers the data to a cloud server for processing, as shown in Fig. 2c. This architecture is typically used for online training or model updates where new sensor data are sent to the cloud server to update the machine learning algorithms. For instance, when a new user starts using the device, their data are uploaded to the cloud

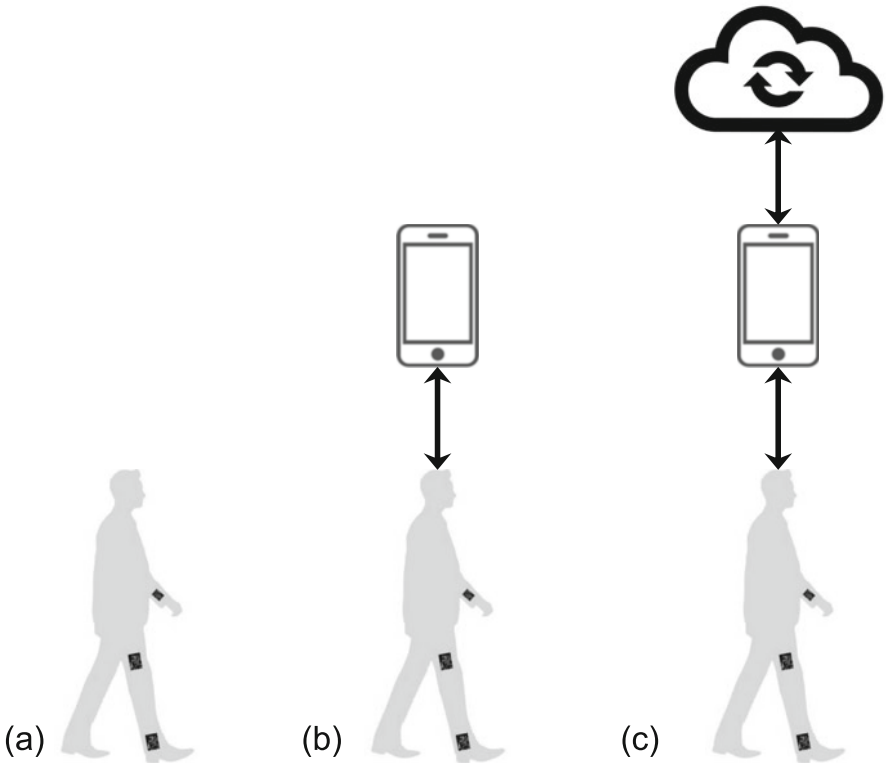


Fig. 2 Illustration of three edge computing architectures for wearable devices (**a**) local computing, (**b**) local computing with a mobile host, (**c**) hierarchy of computing with a cloud server and mobile host

server to train the machine learning model. Overall, processing on a host device or cloud servers is useful and allows wearable devices to employ more complex algorithms. At the same time, these architectures transfer raw data from the device onto external devices, which impacts the privacy of the users [71]. Indeed, data privacy is one of the primary concerns expressed by users when considering the use of wearable devices. Therefore, in the rest of this chapter, we will focus on edge machine learning in the on-device computation architecture, shown in Fig. 2a.

2.2 Edge Machine Learning Algorithms

A number of machine learning algorithms have been employed to enable on-device processing in wearable devices. This section briefly describes the commonly used machine learning algorithms in wearable devices.

2.2.1 Tree-Based Machine Learning Algorithms

Tree-based algorithms, such as decision trees and random forest, are commonly used on wearable devices due to their simplicity and low computational complexity [14, 35, 65]. For example, the approach in [35] uses a decision tree to classify human activities, while the approach in [14] uses random forests to enable gesture recognition. Tree-based algorithms typically use a set of handcrafted features and supervised labels to train the model. The models usually consist of comparison of features at each level of the tree until the leaf node is reached. Decision trees employ a single tree while random forests employ a collection of trees to improve the accuracy of the algorithm. Recent work has also proposed tree-based algorithms tailored for resource-constrained wearable devices [83]. Specifically, the Bonsai [30] approach learns a sparse tree to reduce the size of the model, thus making it suitable for small memories in wearable devices. The Bonsai decision trees are able to achieve accuracy that is comparable with state-of-the-art methods while consuming only few kilobytes of memory.

2.2.2 Support Vector Machines

Support vector machine (SVM) is another class of supervised learning algorithms that are widely used in wearable applications. The traditional SVM algorithms distinguish between two classes by learning a hyperplane that separates the two classes. At runtime, SVM algorithms perform a linear combination of the features and the weights to determine the class of the new data. In multi-class scenarios, multiple SVM classifiers are trained to distinguish between the classes. SVMs have also been used widely in wearable devices due to their ease of implementation. For example, the approaches in [35, 51, 52] use SVM for activity recognition and gesture recognition, respectively. They have also been used in health applications such as Parkinson's disease diagnosis [24, 60].

2.2.3 Neural Networks and Deep Learning

Recent success of neural networks in image processing, natural language processing, and computer vision has prompted their use in wearable devices as well. Deep learning models include multiple layers of neurons between the input and the output. Depending on the type of the model, the layers may be fully connected, convolutional, or recurrent. Recurrent layers are typically used in time series data to reveal the time dependence of the sensor data. Similarly, convolutional layers are used in the initial part of a network to extract features from the data. Then, features are passed through fully connected layers to obtain the final output. Neural network models have been successfully employed for a number of wearable applications including activity recognition, gesture recognition, and arrhythmia detection [11, 52, 76]. Deep learning models are also suitable for online updates

since their parameters can be updated through gradient descent algorithms as new data become available. Many recent studies have employed reinforcement learning with deep learning models to perform online updates so that the models provide high accuracy for new, unseen, users [11, 52, 76]. Due to these advantages, deep learning models are being widely used in wearable devices to process sensor data and enable interesting applications.

2.3 *Challenges for On-Device Edge Machine Learning in Wearable Devices*

Despite the privacy and security advantages of on-device machine learning in wearable devices, they face several challenges in their effective implementation. This section describes the major technical challenges for on-device machine learning in wearable devices.

Limited Computational Capacity Wearable devices typically integrate low-power processors with few hundred megahertz of operating frequency. Wearable processors also do not include graphics processing units, which are one of the most commonly used processing methods for deep learning models. Moreover, microcontrollers included in wearable devices may not include advanced memory management features such as multiple cache levels. Consequently, the machine learning models deployed on wearable devices must be computationally lightweight so that they can be executed on the low-power processors with minimal latency.

Memory Capacity Wearable devices and processors typically have limited memory capacities due to their small size. For instance, the Texas Instruments (TI) CC2652R processor, one of the commonly used processors in wearable devices, has only 80 KB of main memory. The small memory size makes it impossible to deploy popular deep learning models like AlexNet on wearable devices. Therefore, edge machine learning models must have a small memory footprint to ensure that they can be deployed seamlessly on devices with limited memory capacities.

Energy and Power Constraints The battery capacities of wearable devices are limited due to their small size [18]. Integrating larger batteries is challenging because they make the devices bulky and uncomfortable for users. For instance, a battery with 1000 mAh capacity can weigh 20–30 grams, which is more than the weight of other wearable device components. Smaller batteries create two major constraints for the wearable devices. First, it limits the energy budgets available for executing the machine learning algorithms, thus limiting the number of computations that can be performed. Second, it limits the peak processing power for the wearable device. This means that even if an algorithm operates under low energy budgets, it must also have low peak power consumption to operate within the electrical limits of the device. Therefore, edge machine learning algorithms must

be able to operate under extremely low energy budgets while minimizing the peak power consumption.

2.4 Solutions to Address On-Device Learning Challenges

A number of approaches have been recently explored to address the challenges of on-device machine learning. We briefly review some of the approaches below while referring the readers to the surveys in [23] and [21] for more details.

2.4.1 Quantization

Machine learning algorithms typically use floating-point numbers to perform computations. However, floating-point operations consume additional resources for computation, which increase both latency and power consumption. To this end, recent approaches have proposed quantizing the model parameters and inputs into integers so that integer processing units can be used for computations. For example, the approach in [10, 38] uses 16-bit or 8-bit representations of model parameters to reduce the computations in the algorithm. Machine learning models with more aggressive quantization, such as binary neural networks, have also been proposed [56, 111]. Binary neural networks encode all parameters to zeros and ones, which significantly simplifies the computations. Early quantization approaches typically train the machine learning models in floating-point and then convert the parameters to integers. While this is useful, it can impact the accuracy of the machine learning models [34]. Therefore, recently proposed approaches perform quantization-aware training to ensure that the accuracy of the models does not drop with quantization. Quantization-aware training approaches typically use reduced precision data during training so that the need for converting the parameters to integers after training is eliminated [25, 46]. A recent approach proposed in [72] applies quantization to majority of the data during training while handling a small part of data with large values in high precision. This ensures that the quantization error is minimized while also ensuring high accuracy for the models. Quantization has been successfully used in human activity recognition to reduce the model size while maintaining the accuracy [105–107]. Overall, model quantization ensures that low-power and low-complexity integer computations are used on the wearable device for computations.

2.4.2 Model Pruning

Many machine learning models, especially well-known neural networks such as ImageNet [47], are not feasible to be deployed on wearable devices with limited memory capacities. Model pruning is an effective approach being used to reduce

the size of the models so that they run on the wearable devices [113]. A common approach to achieve smaller networks is to first train a large model and then prune parameters that have small magnitudes [20, 33]. Studies have also proposed methods to achieve structured sparsity through the use of theoretical techniques or regularization techniques [59, 67, 92]. More recently, the lottery ticket hypothesis has emerged as an effective method to achieve model pruning in neural networks [27]. The lottery ticket hypothesis states that as opposed to training a large network and then pruning, we can train smaller networks that achieve similar test accuracy as the original, large network. The advantage of the lottery ticket hypothesis is that training the smaller network from scratch with proper initialization provides accuracy similar to the original network. This process also eliminates the need to post-process the trained networks for pruning. Examples of pruning applied to human activity recognition include [17, 31, 91]. In summary, model pruning is a promising approach to ensure that the machine learning models trained for wearable devices are able to execute on the low-power wearable devices.

2.4.3 Energy Harvesting

Energy harvesting has emerged as a promising solution to alleviate the problem of small battery capacities and frequent recharging of wearable devices. There has been extensive research on how harvesting energy from ambient sources can aid in the smooth operation of wearable devices in the state-of-the-art literature [70, 94]. Commonly used ambient energy sources include solar, wind, radio-frequency, and body heat and motion. Recent work has shown that photovoltaic (PV) cells can harvest 0.1–100 $\mathrm{mW/cm^2}$ of power [41, 94] in indoor and outdoor conditions, respectively. Similarly, body motion can provide about 15 μW with a 23.8 $\mathrm{cm^2}$ piezoelectric patch when walking [90]. An antenna with -10 dBm gain will yield about 10 μW using radio frequency harvesting [68]. Indeed, energy harvesting has been used in multiple human activity recognition studies [36, 44, 64]. At the same time, harvesting energy from ambient sources is stochastic in nature. For instance, solar energy experiences seasonal and diurnal variations, whereas wind energy is unpredictable in nature. In addition, body motion is factored by human activities [28]. Therefore, the need for an accurate energy prediction model and efficient management of the harvested energy is critical.

2.5 Edge Machine Learning in Health Applications

This section provides some recent examples of edge machine learning being used in health applications. Specifically, we provide representative studies of edge machine learning being used for Parkinson's disease diagnosis, vital sign monitoring, and human–computer interaction.

2.5.1 Parkinson's Disease Diagnosis

Parkinson's disease is typically diagnosed with clinical assessment of motor and non-motor symptoms in a clinical environment [43]. However, the clinical assessments can be subjective and have variations across different patients. Therefore, wearable devices and edge machine learning are being used in the diagnosis of Parkinson's disease [77, 109]. Woods et al. [101] use data from a smartphone in an SVM algorithm to diagnose Parkinson's disease. Similarly, the work in [1] uses a random forest algorithm to perform gait analysis so that Parkinson's disease can be detected in patients. Specifically, the authors use a ground reaction force sensor worn under the foot to perform the gait analysis. Non-motor symptoms have also been studied using edge machine learning. For instance, Tsanas et al. [89] process audio recordings using SVM and random forest methods to detect the presence of Parkinson's disease symptoms. Overall, these studies show that edge machine learning can lead to standardization and objective measures for Parkinson's disease. We refer the interested readers to the survey in [48] for more detailed analysis of edge machine learning in Parkinson's disease.

2.5.2 Vital Sign Monitoring

Vital sign monitoring is another high-impact application that uses edge machine learning in wearable devices. Common vital signs monitored using wearable devices include heart rate, blood pressure, oxygen levels, and respiration rates. The continuous monitoring of vital signs can aid clinicians in getting a deeper understanding of the patient's health as opposed to periodic measurements [100]. A recent study in [99] showed that machine learning approaches with wearable sensors can provide accuracy that is equivalent to clinical methods. The approach uses random forest and Lasso models to predict clinical laboratory test results using readings from wearable sensors. Similarly, the work in [12] presents the uses of wearable sensors along with SVMs to monitor the risk of cardiovascular disease. In summary, wearable sensors along with edge machine learning algorithms provide a powerful method for vital sign monitoring of individuals in free living environments.

2.5.3 Human–Computer Interaction

Human–computer interaction (HCI) is another important application where wearable devices can make an impact. Specifically, HCI using wearables can enable natural interactions between humans and computers [62]. For instance, using wearable devices and gesture recognition to control a gaming device or computer provides a natural method of interaction with the applications. To this end, the recent research has developed a number of edge machine learning approaches for HCI using wearable devices. The study in [73] uses data from an accelerometer sensor in a neural network to recognize five gestures. Similarly, Ferrone et al. [26] use strain

sensors to recognize gestures using linear discriminant analysis and SVM classifiers. The strain sensors are beneficial because they have low power consumption due to being passive. Overall, these studies show that wearable HCI using edge machine learning can enable new and novel methods of interacting with computers. We refer the readers to the survey in [39] for a more detailed overview of HCI using wearable devices.

3 Robustness in Wearable Applications

Reliability and robustness are an important criterion for wearable devices and edge machine learning algorithms to ensure that the users receive a high quality of service. However, this is a challenging problem for wearable devices due to the dynamic nature of their use as well as the potential for user errors. Furthermore, the limited battery capacity of wearable sensors can lead to interruptions in the sensor data. Therefore, wearable devices must take measures to ensure that edge machine learning algorithms are able to handle the reliability challenges. This section first introduces the three major challenges to reliability of machine learning algorithms in wearable devices and then reviews the current state of the art to handle the challenges.

3.1 Reliability Challenges for Wearable Devices

3.1.1 Sensor Shifts and Disturbances

Machine learning algorithms are typically trained with data collected in controlled environments where the sensor position and parameters are carefully optimized. The data collection procedure also ensures that the sensors and processors are operating at their optimal frequencies. For instance, if the sampling rate of the sensor is not stable during the data collection, the experiment is repeated to ensure that the data is clean. While this setup is useful to train accurate machine learning models, the real-world data distribution may differ from the distribution during training. The shift in the sensor data distribution can occur due to one or more of the following reasons:

- The sensor locations may change with long-term usage, as illustrated in Fig. 3. For example, if the sensor is mounted using a sleeve or a belt, it may slip or move with usage. This will result in a change in the distribution of the sensor data. The data distribution may also change due to user errors, such as incorrect orientation of the sensors.
- The sampling frequencies and calibration of the sensors may have variations from one device to another, which alters the distribution of the observed data. Indeed,

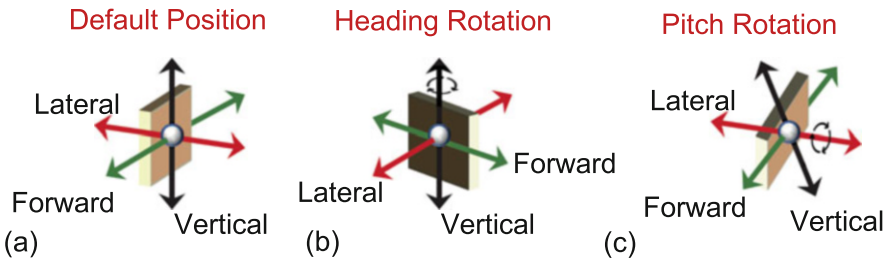


Fig. 3 Example of sensor disturbances in wearable devices, (**a**) default position of sensor, (**b**) sensor directions with heading rotation, (**c**) sensor directions with pitch rotation

a recent study analyzed the sampling rates of accelerometers from 13 devices and observed that there is a wide variation across devices [85]. Consequently, if a single classifier is trained assuming a fixed sampling frequency, it may suffer from accuracy reduction when tested on a new device.

- The activity or application patterns of users in the real world may not match the users used during training. Even for the same user, the data distribution may change with time. For instance, if a user is injured for a period of time, the activity patterns will be different from when the user was healthy.

The shifts in the sensor data distribution must be handled carefully for reliable performance of the edge machine learning algorithms. Therefore, recent research has focused on ensuring that machine learning algorithms can handle the sensor disturbances. We review some of these methods later in this section.

3.1.2 Missing Sensor Data

Edge machine learning algorithms for wearable devices are also trained with the assumption that all the sensors and devices will be operating perfectly at runtime. However, this assumption may not hold true in many real-world application scenarios. For example, due to energy constraints, one or more sensors can turn off, thus leading to missing data. The missing data can also occur due to sensor malfunction or communication issues. Specifically, the wearable sensors are connected to the processor through serial buses, which may drop packets in some instances due to bandwidth limitations. The missing data, in turn, leads to significant reduction in the accuracy of the machine learning models because the models are trained with the assumption that data from all the sensors are available. Indeed, our case study with the activity recognition application shows that missing sensor data leads to 30–40% accuracy drop. Training individual classifiers for each combination of sensors is also not feasible due to the exponential increase in the number of combinations with sensors. Therefore, recent research has focused on developing methods to handle missing data in edge machine learning algorithms.

3.1.3 Energy and Power Constraints

Energy and power constraints pose challenges [41] to reliable operation of edge machine learning algorithms in addition to the on-device edge machine learning. The small battery capacities of wearable devices not only limit the complexity of edge machine learning models but also affect the reliability of the models. Specifically, lack of energy can lead to sensors being turned off during important activities, which leads to a reduction in the quality of service to the user. The energy limitations may also force the device to operate in low-power mode with reduced precision of sensing or computations. This leads to a reduction in the accuracy of the edge machine learning models. Energy harvesting from ambient sources is a promising technology to alleviate the energy limitations in the wearable devices. However, just integrating energy harvesting technologies into wearable devices is not sufficient as the harvested energy must be carefully managed to ensure that there is sufficient energy available even when ambient energy is not available. To this end, recent approaches have focused on optimal energy harvesting and management, as we outline in the next section.

3.2 State-of-the-Art Methods for Robust Edge Machine Learning in Wearable Devices

Approaches to achieve reliable machine learning have been developed for classification, clustering, regression, and data imputation [49, 50, 63, 66, 93, 97, 97]. In this section, we describe recent approaches for robust and reliable edge machine learning in wearable devices for each of the challenges in the previous section.

3.3 Approaches to Address Sensor Shifts and Disturbances

A number of approaches to address sensor orientation disturbances have been proposed in the literature [49, 50, 66, 93, 97]. These approaches can be broadly divided into two classes. The first class of approaches detects the sensor disturbances at runtime and corrects the data before processing it further. For instance, the approaches in [49, 50] use principal component analysis (PCA) to determine the orientation of an accelerometer sensor in an activity recognition application. Specifically, PCA is used when there is heading rotation change in the direction of motion. For example, if one of the accelerometer axes is pointing in the direction of motion, heading rotation will result in the projection of the motion acceleration on the other two axes. To solve this issue, user movement calibration is done first to detect the correct direction of motion offline, and then PCA component is calculated for the reference and new movement in order to calculate the angle

between the original and the new motion direction. After the angle is calculated, it is used to recover the sensor heading direction. Then, the orientation information is used to correct the sensor data in case of any variations from the training setup. This approach is useful when there are changes in the direction of motion for the accelerometer in activity monitoring applications. Similarly, the approach in [66] uses the gravitational force of the Earth to determine the orientation of motion sensors and apply corrections, if needed. This is useful in ensuring that any motion sensor disturbances with respect to the gravitational axis are corrected before processing by machine learning applications.

The second class of methods to resolve sensor data shift and disturbances uses data augmentation [63, 93, 97]. Specifically, some examples of data with shifts or disturbances are included in the training data to ensure that the edge machine learning algorithms learn the sensor data disturbances as well. For instance, in [93], the authors use a convolutional neural network (CNN) for data augmentation of wearable sensor data for Parkinson's disease monitoring. The training of CNN model requires large dataset, and therefore, data augmentation is a way to increase the input data while still preserving the labels. However, any disturbance or scaling in the sensor data may lead to incorrect label. Therefore, the authors use label-preserving data augmentation techniques such as jittering, cropping, magnitude-warping, and time-warping to increase the non-disturbed data instances for better CNN training. In summary, these methods allow the edge machine learning to provide reliable performance in the presence of sensor data disturbances.

3.4 Missing Data Recovery Algorithms

Development of algorithms to recover missing data at runtime is critical to ensure that the wearable applications provide reliable quality of service. The data recovery algorithms impute any missing sensor values before the machine learning algorithms process the data, thus ensuring that the application does not suffer from accuracy reductions. A number of approaches to recover missing data at runtime have been proposed in the literature [75, 76, 86]. One of the most popular approaches for missing data imputation is the k-nearest neighbor (k-NN) algorithm [75, 76]. Specifically, whenever missing data are detected, the algorithm identifies other data windows that are close to the missing value. Then, the neighboring values are used to estimate the missing sample by taking the mean of the available samples. However, k-NN methods are typically not suitable for wearable devices as they need to store the training data on the device, which imposes memory overhead on the wearable devices.

To handle the limitation of the data storage requirements of k-NN algorithms, the recent research has proposed using autoencoders and generative networks for imputing missing data [79, 84, 110]. For instance, Saeed et al. propose an adversarial autoencoder model to handle missing sensory features and samples before classification. The adversarial autoencoder uses a traditional autoencoder architecture in

addition to a discriminator network to force the encoder output to match a specific target distribution. This additional discriminator network transforms the traditional autoencoder into a generative model to recover the missing data. More recently, the success of generative adversarial networks (GANs) in the image recognition domain [22] has prompted their use in the recovery of sensor data for wearable devices as well. For instance, conditional GANs are being used to generate data for specific labels or scenarios in a wearable application [84]. GANs have also been used to impute sensor data when some portions of the data are missing. Specifically, the generative adversarial imputation network (GAIN) [110] is a network that is specifically designed to recover missing samples in time series data. GAIN takes the available data with missing samples as input, in addition to a vector that denotes the missing samples' locations. The generator of GAIN is used then to recover the missing data by learning the distribution of the data and using the available data to impute the missing samples. We present a concrete instantiation of the GAIN in our HAR case study. In summary, the goal of the data imputation approaches is to recover the missing data at runtime so that a single edge machine learning is able to provide reliable operation to the user and the application.

3.4.1 Energy Management in Wearable Devices

As indicated in Sect. 2.4.3, energy harvesting is a promising solution to help with the battery energy limitations of wearable devices. However, energy harvesting from ambient sources may be unreliable due to its stochastic nature. Therefore, developing accurate energy forecast models is critical. Furthermore, energy management of the harvested energy is required for optimal energy utilization in these devices. We evaluate some of the possible solutions that have been discussed in the literature.

Energy Harvesting Prediction Methods A number of approaches have been explored in the literature to design and develop accurate energy harvesting prediction models. Particularly, there has been a growing interest in predicting solar energy harvesting due to the high magnitude of solar energy. Kansal et al. proposed the Exponentially Weighted Moving-Average (EWMA) algorithm [42], which is a frequently used solar energy prediction scheme. It employs an exponentially moving average filter to take advantage of the diurnal pattern of energy harvesting throughout the day. However, because of the variability of sunny and overcast days, it suffers from high prediction errors. To address this issue, some authors proposed energy prediction algorithms based on energy profiles. Noh et al. [69] improve the EWMA model by keeping track of previous solar energy profiles based on a scaling factor. Piorno et al. suggested a Weather Conditioned Moving Average (WCMA) forecast model in [74] that considers previous days' energy harvesting. It does, however, include a weighting component to measure how the current day's weather conditions differ from the preceding days. Similarly, in [15], Cammarano et al. proposed another profile-based prediction model, Pro-Energy, to predict solar and wind energy harvesting. Pro-energy predicts future energy availability at short

(few minutes to half an hour) and medium (a few hours) predictions horizons. They further extend the predictions with variable length intervals in [16].

In recent years, the authors are taking advantage of machine learning techniques to predict future energy availability. Sharma et al. established a model for predicting solar energy using multiple regression techniques in [81]. The authors proposed a hierarchical machine learning model to classify the energy harvesting magnitude of the day and accurately predict solar energy based on the classification in [104]. Similarly, the authors propose energy prediction models based on Recurrent Neural Networks (RNNs) in [54, 108]. In summary, the energy prediction approaches provide estimates of the future energy to the management algorithms so that effective energy management can be performed. We refer the readers to the survey in [19] for a more comprehensive analysis of the energy harvesting and prediction methods.

Energy Management Approaches It is essential to have an efficient energy management algorithm to allocate the harvested energy in wearable devices effectively. Many energy management algorithms have been suggested in the literature—most approaches aim to achieve energy neutral operation, as suggested by Kansal et al. [9, 42, 57]. Energy neutral operation (ENO) refers to enabling the device's functioning using harvested energy only. Energy management algorithms, particularly, aim to establish ENO and achieve maximum utility to the device.

There are several approaches for energy management [42] presented in the literature. The first class of algorithms utilizes convex optimization and dynamic programming to allocate the harvested energy. The approaches proposed in [9, 42, 96] use various optimization techniques for the effective energy allocation of wearable devices. Furthermore, the authors in [102, 103] focus on the dynamic trading of energy between multiple resources, which can be adapted to allocate energy between multiple devices on the body. Despite being effective, these approaches rely on dynamic optimization at runtime to account for differences in energy harvesting. This can lead to significant execution time and memory overhead. Therefore, techniques that do not require optimization at runtime are also being studied to minimize energy consumption for the IoT device.

Another class of solutions switches to different modes of operation to adjust the energy consumption based on available energy. Task scheduling [55] and duty cycling schemes [13, 57] are among those solutions. However, these approaches are heavily dependent on the harvested energy fluctuations. Therefore, some machine learning-based energy management approaches have been proposed in the literature. Among the approaches, [3, 7] use reinforcement learning (RL) to manage the energy in IoT devices. To summarize, the literature has explored different directions for effective energy management in wearable devices. We refer the survey in [80] for a more detailed exposition of energy management methods.

3.5 Future Opportunities

In spite of recent approaches to handle sensor disturbances and missing data in wearable devices, several challenges and opportunities remain to fully realize the potential of reliable edge machine learning for wearable devices. Specifically, we present the following research directions and opportunities for future researchers (see Table 1).

- Current approaches for sensor data recovery are typically integrated before the edge machine learning pipeline for wearable devices. While this is useful, it introduces additional execution time and energy overheads on already resource-constrained devices. Therefore, there is a need for future research approaches that eliminate the extra step of data recovery while enabling reliable edge machine learning applications.
- The energy management approaches for wearable devices are generally unable to account for sudden changes in the user activities, such as falls or injuries. Instead, they rely on typical activity patterns and make energy management decisions based on the trend of activity patterns. Adapting the energy management algorithms for sudden events will improve the reliability of the edge machine learning algorithms.
- State-of-the-art approaches for missing data recovery are well suited for isolated missing data instances. However, many algorithms face challenges when there are longer sequences of missing data. Developing approaches for recovery of long missing data sequences will further improve the reliability of the wearable devices.

Next, we present a case study with the human activity recognition application to explore the impacts of missing data on the accuracy and then show that missing data can be effectively recovered with generative networks.

Table 1 Challenges and opportunities in wearable health applications

Challenge	Potential solutions
Missing data from sensors	Generative methods to recover data at runtime
Sensor data shift	Efficient methods to generate additional training examples to obtain reliable classifiers
Energy limitations	Ambient energy harvesting, prediction, and optimal management
Monitoring of critical activities in health applications	Adaptive energy management that can predict future activities of users

4 Human Activity Recognition Case Study

Human activity recognition (HAR) has increased in popularity due to the recent development of low-power wearable devices that integrate multiple sensors, microprocessors, and communication capabilities [8, 24, 52]. HAR algorithms are being used for fitness monitoring, rehabilitation, and in knowing the activities of movement disorder patients [51, 61].

HAR algorithm development and validation typically involve four crucial steps: training data collection and labeling, activity segmentation, classifier training, and testing on new users [52, 82, 98]. The data collection is done in a controlled environment where the developers ensure that the sensors are in perfect working condition, and they are mounted in a known position. The collected data are then divided into distinct segments and labeled by one or more experts. The labeled data are passed through a feature generation algorithm to obtain features that distinguish between the activities of interest. The features and labels are input to a supervised learning algorithm to train an activity classifier. Finally, the trained classifier is used at runtime to identify the activities of one or more new users.

State-of-the-art HAR algorithms in the literature generally assume that all the sensor data are available perfectly without any missing samples. However, in the real-world usage, the wearable devices can encounter scenarios where there are missing samples either due to user error, sensor malfunction, or energy limitations. The missing data, in turn, lead to a significant drop in the accuracy. For instance, the accuracy drops from 95% to 50% with only 10% of the samples missing. Therefore, there is a strong need to develop robust HAR classifiers that provide high accuracy in the presence of missing data.

One of the common approaches to develop robust classifiers is to include examples with missing samples during training. However, including missing samples during training does not yield significant benefits in accuracy. For instance, including examples with 10% missing data in training increases the accuracy to only about 65%. Furthermore, including missing samples during training does not account for all possible scenarios at runtime. Therefore, in this chapter, we develop a runtime approach to recover missing samples in HAR. Specifically, we leverage recently proposed generative adversarial imputation networks (GAINs) to recover missing data at runtime [110]. GAIN is an adaptation of the general generative adversarial networks that is well suited for recovering missing samples in time series data [29]. Starting with a baseline GAIN with a large number of weights, we perform a design space exploration of various GAIN structures to analyze the trade-off between accuracy and overhead. This is important because wearable devices are generally constrained by their limited batteries. As such, it is critical to obtain the highest accuracy while minimizing the execution time and energy. After recovering the missing samples with GAIN, the robust HAR classifier generates the features and identifies the activity.

We validate the robust HAR classifier using w-HAR, a publicly available HAR dataset [11]. The dataset does not include any missing data by default. Therefore, we first use missing data in the dataset and then use GAIN to recover the missing

data. We also implement the GAIN algorithm on the TI CC2652 microcontroller to measure the energy overhead [87]. Overall, the experiments show that GAIN is able to effectively recover missing data and enable accurate activity classification.

4.1 HAR Background

As detailed in Sect. 2, robust HAR involves segmentation, feature generation, and classification. In this section, we provide a brief summary of each step in HAR.

Segmentation and Data Recovery Sensors used in HAR produce streaming data at a pre-configured sampling frequency. The streaming data must be divided into distinct activity segments so that each segment contains a single activity. The segmentation step achieves this by utilizing variable-length or fixed-length segmentation algorithms [52, 82, 98]. Next, the data recovery algorithm checks for any missing samples and recovers them if it detects missing samples.

Feature Generation The activity segments are processed by the feature generation block to calculate the features required for classification. In this chapter, we reuse the features provided in the w-HAR dataset [11].

Classification The features are finally used to identify the activity being performed by the user. Any supervised learning algorithm can be used to identify the activities. In this work, we utilize the neural network structure proposed in the w-HAR dataset.

4.2 Generative Adversarial Imputation Networks

Generative adversarial networks (GANs) [29] have been used recently to obtain synthetic data for a number of applications including image recognition, speech recognition, and natural language processing [5, 32, 45]. GANs are useful in generating completely new data examples, but they are not well suited to recover missing data when partial information is available. To address this limitation, the work in [110] proposed a variation of GAN called as GAIN. Following the general structure of GANs, GAIN consists of a generator and a discriminator. The goal of the generator in GAIN is to accurately recover missing data by utilizing the available samples, while the job of the discriminator is to distinguish between recovered synthetic data and observed data. The generator is trained to minimize the error between the generated data and the actual observed data. By minimizing this error, the generator can generate accurate samples for instances where the observed data are not available. The generator is also trained to maximize the misclassification error in the discriminator so that the discriminator is unable to distinguish between synthetic and observed data. When fully trained, the GAIN generator is able to accurately recover the missing samples and the discriminator

is unable to distinguish the synthetic data. This means that the data generated by GAIN follow the distribution of the actual data. Next, we illustrate the accuracy gains achieved by GAIN when the sensor data are missing samples.

4.3 Experiments and Results

4.3.1 Experimental Setup

Wearable Device We use the TI-CC2652R as the primary processor for our wearable device model [87]. The TI-CC2652R processor consists of ARM Cortex processor which does the segmentation, data recovery, feature generation, and classification. The robust HAR classifier is implemented on the TI-CC2652 processor to measure the overhead.

HAR Dataset We use the w-HAR dataset to perform our experiments. The dataset provides stretch sensor and accelerometer readings for 22 users, while they are performing eight activities: jump, lie down, sit, stand, walk, stairs up, stairs down, and transition. Missing data implementation: We introduce missing data for each 3-second interval of the w-HAR dataset by randomly choosing indices in the 3-second window. To evaluate the performance of GAIN for various missing data lengths, we use the following configurations of missing data percentages: 2%, 5%, 10%, 20%, and 30%.

GAIN Structure We use a GAIN generator neural network with two hidden layers and one output layer. We vary the number of neurons in the hidden layers to perform our design space exploration. The hidden layers in the generator use the ReLU activation. The output layer contains one unit for each sample in the input data and uses the sigmoid activation. It is important to note that the generator produces data even for observed samples. In our implementation, we discard those data and use the generated samples for only the missing data.

4.3.2 Design Space Exploration for GAIN

In this section, we vary the number of neurons in the hidden layers with 2 hidden layer and 3 hidden layer architectures and evaluate the accuracy achieved by the robust HAR classifier with GAIN. We also compare the memory requirements of each configuration to understand the overhead of GAIN. The memory requirements, in turn, correlate with the execution time and energy overheads on the wearable device. For instance, if the number of weights doubles, we see an increase in the execution time and energy required for using the GAIN to generate data. We use this methodology to quantify the overhead because all GAIN configurations are not feasible for the TI-CC2652R processor due to its limited memory. At the same time, this analysis provides a useful comparison point for future wearable with higher memory.

Table 2 Details of GAIN architectures

	1st hidden layer	2nd hidden layer	3rd hidden layer
2 hidden layer models	4–183	4–183	_____
	Acc: 91.04–96.8%	Acc: 91.04–96.8%	
3 hidden layer models	4–22	11–91	22–91
	Acc: 92.37–88.52%	Acc: 92.37–88.52%	Acc: 92.37–88.52%

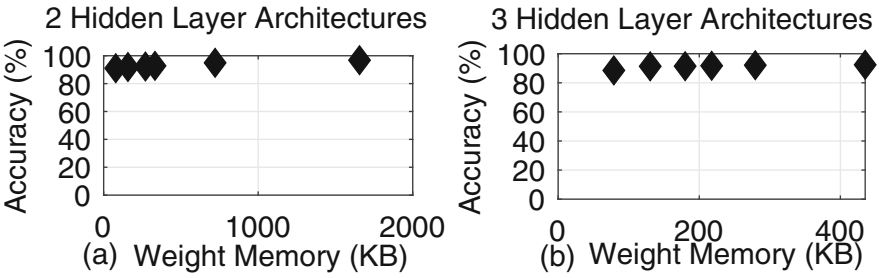


Fig. 4 Comparison of accuracy achieved by HAR with GAIN for data recovery

Table 2 provides details on the hidden layer configurations we use for the GAIN. As shown in the table, for the 2 hidden layer models, we vary the number of neurons in both first and second layers from 4 to 183. We also work with 3-layer models and vary the number of neurons in the first hidden layer from 4 to 22, while for the second hidden layer we vary from 11 to 91, and lastly, we vary the number of neurons in the third layer from 22 to 91 neurons. For each configuration, we use a uniform mix of missing data percentages and obtain the accuracy.

Figure 4 shows the overall recognition accuracy for each configuration as a function of the number of weights. Increasing the number of weights leads to higher accuracy while incurring a higher overhead. Therefore, designers must choose an architecture that provides the optimal trade-off between the accuracy, device memory constraints, and the overhead. To this end, we implement a three hidden layer architecture with 4 neurons in the first layer, 11 neurons in the second layer, and 45 neurons in the third layer. We use this architecture since it fits in the memory of the TI CC2652R processor while providing a high accuracy. Our measurements on the TI CC2652R processor show that GAIN takes 9 ms for the stretch sensor and 15 ms for each accelerometer direction to recover the data. Moreover, the energy consumption is 99 μJ and 168 μJ for stretch and accelerometer, respectively. These overhead values amount to less than 10% overhead for each activity.

Summary HAR is an essential component of personalized healthcare and activity monitoring. However, most classifiers designed for HAR do not consider missing data in their training process, which can lead to reduced accuracy in real-world usage. To address this issue, we proposed an adaptive imputation algorithm that recovers missing sensor data before activity segmentation. Experiments with three publicly available datasets showed that the proposed algorithm effectively recovers

sensor data and achieves accuracy that is comparable to the accuracy with clean data while using the same classifier weights.

5 Conclusion

Wearable devices are being increasingly used for health monitoring, rehabilitation, and fitness applications. These applications are enabled by the advances in edge machine algorithms that are able to accurately process sensor data to fulfill user needs. However, the edge machine learning models face a number of robustness challenges including sensor data shift, missing samples, and energy constraints. This chapter reviewed recent methods to address the reliability challenges and presented some future research opportunities to improve the reliability of edge machine learning algorithms for wearable devices. We also presented a case study on the HAR application to highlight the reliability challenges as well as demonstrate the effectiveness of the recently proposed GAIN approach for recovering missing data. Our case study shows that recovery of missing data at runtime using GAIN is a promising approach to enable reliable HAR classifications.

References

1. Açıcı, K., Erdaş, Ç.B., Aşuroğlu, T., Toprak, M.K., Erdem, H., Oğul, H.: A random forest method to detect Parkinson's disease via gait analysis. In: International Conference on Engineering Applications of Neural Networks, pp. 609–619. Springer, Berlin (2017)
2. Anguita, D., Ghio, A., Oneto, L., Parra, X., Reyes-Ortiz, J.L.: Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. In: International Workshop on Ambient Assisted Living, pp. 216–223. Springer, Berlin (2012)
3. Aoudia, F.A., Gautier, M., Berder, O.: RLMan: An energy manager based on reinforcement learning for energy harvesting wireless sensor networks. IEEE Trans. Green Commun. Netw. **2**(2), 408–417 (2018)
4. Atzori, L., Iera, A., Morabito, G.: The Internet of Things: a survey. Comput. Netw. **54**(15), 2787–2805 (2010)
5. Bao, J., Chen, D., Wen, F., Li, H., Hua, G.: CVAE-GAN: fine-grained image generation through asymmetric training. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2745–2754 (2017)
6. Bao, L., Intille, S.S.: Activity recognition from user-annotated acceleration data. In: Int. Conf. on Pervasive Comput., pp. 1–17 (2004)
7. Basaklar, T., Tuncel, Y., Ogras, U.Y.: tinyMAN: lightweight energy manager using reinforcement learning for energy harvesting wearable IoT devices (2022). arXiv preprint arXiv:2202.09297
8. Bhat, G., Deb, R., Chaurasia, V.V., Shill, H., Ogras, U.Y.: Online human activity recognition using low-power wearable devices. In: Proc. of Int. Conf. on Comput. Aided Design, pp. 72:1–72:8 (2018). https://doi.org/10.1145/3240765.3240833
9. Bhat, G., Park, J., Ogras, U.Y.: Near-optimal energy allocation for self-powered wearable systems. In: Proc. Int. Conf. on Comput.-Aided Design, pp. 368–375 (2017)
10. Bhat, G., Tuncel, Y., An, S., Lee, H.G., Ogras, U.Y.: An ultra-low energy human activity recognition accelerator for wearable health applications. ACM Trans. Embedd. Comput. Syst. **18**(5s), 1–22 (2019)

11. Bhat, G., et al.: w-HAR: an activity recognition dataset and framework using low-power wearable devices. Sensors **20**(18), 5356 (2020)
12. Boursalie, O., Samavi, R., Doyle, T.E.: M4cvd: mobile machine learning model for monitoring cardiovascular disease. Proc. Comput. Sci. **63**, 384–391 (2015)
13. Buchli, B., Sutton, F., Beutel, J., Thiele, L.: Dynamic power management for long-term energy neutral operation of solar energy harvesting systems. In: Proc. Conf. on Embedd. Network Sensor Syst., pp. 31–45 (2014)
14. Camgöz, N.C., Kindiroglu, A.A., Akarun, L.: Gesture recognition using template based random forest classifiers. In: European Conference on Computer Vision, pp. 579–594. Springer, Berlin (2014)
15. Cammarano, A., Petrioli, C., Spenza, D.: Pro-energy: a novel energy prediction model for solar and wind energy-harvesting wireless sensor networks. In: Int. Conf. on Mobile Ad-Hoc and Sensor Syst., pp. 75–83 (2012)
16. Cammarano, A., Petrioli, C., Spenza, D.: Online energy harvesting prediction in environmentally powered wireless sensor networks. IEEE Sensors J. **16**(17), 6793–6804 (2016)
17. Cao, J., Li, W., Ma, C., Tao, Z.: Optimizing multi-sensor deployment via ensemble pruning for wearable activity recognition. Inform. Fusion **41**, 68–79 (2018)
18. Chen, Y.K.: Challenges and Opportunities of Internet of Things. In: ASPDAC, pp. 383–388 (2012)
19. Chong, Y.W., Ismail, W., Ko, K., Lee, C.Y.: Energy harvesting for wearable devices: a review. IEEE Sensors J. **19**(20), 9047–9062 (2019)
20. Collins, M.D., Kohli, P.: Memory bounded deep convolutional networks (2014). arXiv preprint arXiv:1412.1442
21. Covi, E., Donati, E., Liang, X., Kappel, D., Heidari, H., Payvand, M., Wang, W.: Adaptive extreme edge computing for wearable devices. Front. Neurosci. **15** (2021)
22. Creswell, A., White, T., Dumoulin, V., Arulkumaran, K., Sengupta, B., Bharath, A.A.: Generative adversarial networks: an overview. IEEE Signal Process. Mag. **35**(1), 53–65 (2018)
23. Dhar, S., Guo, J., Liu, J., Tripathi, S., Kurup, U., Shah, M.: A survey of on-device machine learning: an algorithms and learning theory perspective. ACM Trans. Internet Things **2**(3), 1–49 (2021)
24. Espay, A.J., et al.: Technology in Parkinson's disease: challenges and opportunities. Movt. Disorders **31**(9), 1272–1282 (2016)
25. Fan, A., Stock, P., Graham, B., Grave, E., Gribonval, R., Jegou, H., Joulin, A.: Training with quantization noise for extreme model compression (2020). arXiv preprint arXiv:2004.07320
26. Ferrone, A., Maita, F., Maiolo, L., Arquilla, M., Castiello, A., Pecora, A., Jiang, X., Menon, C., Colace, L.: Wearable band for hand gesture recognition based on strain sensors. In: 2016 6th IEEE International Conference on Biomedical Robotics and Biomechatronics (BioRob), pp. 1319–1322. IEEE, Piscataway (2016)
27. Frankle, J., Carbin, M.: The lottery ticket hypothesis: Finding sparse, trainable neural networks (2018). arXiv preprint arXiv:1803.03635
28. Geisler, M., et al.: Human-motion energy harvester for autonomous body area sensors. Smart Mater. Struct. **557**(1), 012024 (2017)
29. Goodfellow, I., et al.: Generative adversarial nets. In: NeurIPS, pp. 2672–2680 (2014)
30. Gope, D., Dasika, G., Mattina, M.: Ternary hybrid neural-tree networks for highly constrained IoT applications. Proc. Mach. Learn. Syst. **1**, 190–200 (2019)
31. Gupta, S., Jain, S., Roy, B., Deb, A.: A tinyML approach to human activity recognition. In: Journal of Physics: Conference Series, vol. 2273, p. 012025. IOP Publishing (2022)
32. Haidar, M., Rezagholizadeh, M., et al.: TextKD-GAN: Text generation using knowledge distillation and generative adversarial networks. In: Canadian Conference on Artificial Intelligence, pp. 107–118. Springer, Berlin (2019)
33. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in Neural Information Processing Systems, vol. 28 (2015)

34. Hashemi, S., Anthony, N., Tann, H., Bahar, R.I., Reda, S.: Understanding the impact of precision quantization on the accuracy and energy of neural networks. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017, pp. 1474–1479. IEEE, Piscataway (2017)
35. Hossain, T., Inoue, S.: A comparative study on missing data handling using machine learning for human activity recognition. In: 2019 Joint 8th International Conference on Informatics, Electronics & Vision (ICIEV) and 2019 3rd International Conference on Imaging, Vision & Pattern Recognition (icIVPR), pp. 124–129. IEEE, Piscataway (2019)
36. Huang, H., Li, X., Liu, S., Hu, S., Sun, Y.: Tribomotion: a self-powered triboelectric motion sensor in wearable Internet of Things for human activity recognition and energy harvesting. IEEE Internet Things J. **5**(6), 4441–4453 (2018)
37. Hwang, G.T., et al.: Self-powered cardiac pacemaker enabled by flexible single crystalline PMN-PT piezoelectric energy harvester. Adv. Mater. **26**(28), 4880–4887 (2014)
38. Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., Kalenichenko, D.: Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2704–2713 (2018)
39. Jiang, S., Kang, P., Song, X., Lo, B., Shull, P.B.: Emerging wearable interfaces and algorithms for hand gesture recognition: a survey. IEEE Reviews in Biomedical Engineering **15**, 85–102 (2021)
40. Jin, X., Li, L., Dang, F., Chen, X., Liu, Y.: A survey on edge computing for wearable technology. Digit. Signal Process. **125**, 103146 (2021)
41. Jokic, P., Magno, M.: Powering smart wearable systems with flexible solar energy harvesting. In: 2017 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–4. IEEE, Piscataway (2017)
42. Kansal, A., Hsu, J., Zahedi, S., Srivastava, M.B.: Power management in energy harvesting sensor networks. ACM Trans. Embedd. Comput. Syst. **6**(4), 32 (2007)
43. Kassubek, J.: Diagnostic procedures during the course of Parkinson's disease. Basal Ganglia **4**(1), 15–18 (2014)
44. Khalifa, S., Lan, G., Hassan, M., Seneviratne, A., Das, S.K.: Harke: Human activity recognition from kinetic energy harvesting data in wearable devices. IEEE Trans. Mobile Comput. **17**(6), 1353–1368 (2017)
45. Kostak, M., Berger, A., Slaby, A.: Migration of artificial neural networks to smartphones. In: International Conference on Computational Science and Its Applications, pp. 845–858. Springer, Berlin (2020)
46. Krishnamoorthi, R.: Quantizing deep convolutional networks for efficient inference: a whitepaper (2018). arXiv preprint arXiv:1806.08342
47. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, vol. 25 (2012)
48. Kubota, K.J., Chen, J.A., Little, M.A.: Machine learning for large-scale wearable sensor data in Parkinson's disease: concepts, promises, pitfalls, and futures. Movement Disorders **31**(9), 1314–1326 (2016)
49. Kunze, K., Lukowicz, P.: Sensor placement variations in wearable activity recognition. IEEE Pervasive Comput. **13**(4), 32–41 (2014)
50. Kunze, K., Lukowicz, P., Partridge, K., Begole, B.: Which way am i facing: inferring horizontal device orientation from an accelerometer signal. In: 2009 International Symposium on Wearable Computers, pp. 149–150. IEEE, Piscataway (2009)
51. Kwapisz, J.R., et al.: Activity recognition using cell phone accelerometers. ACM SigKDD Explorations Newslett. **12**(2), 74–82 (2011)
52. Lara, O.D., et al.: A survey on human activity recognition using wearable sensors. IEEE Commun. Surv. Tut. **15**(3), 1192–1209 (2012)
53. Lee, S.M., Yoon, S.M., Cho, H.: Human activity recognition from accelerometer data using convolutional neural network. In: 2017 IEEE International Conference on Big Data and Smart Computing (BigComp), pp. 131–134. IEEE, Piscataway (2017)

54. Li, G., Wang, H., Zhang, S., Xin, J., Liu, H.: Recurrent neural networks based photovoltaic power forecasting approach. Energies **12**(13), 2538 (2019)
55. Lin, X., Wang, Y., Chang, N., Pedram, M.: Concurrent task scheduling and dynamic voltage and frequency scaling in a real-time embedded system with energy harvesting. IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. **35**(11), 1890–1902 (2016)
56. Lin, X., Zhao, C., Pan, W.: Towards accurate binary convolutional neural network. In: Advances in Neural Information Processing Systems, vol. 30 (2017)
57. Liu, S., Lu, J., Wu, Q., Qiu, Q.: Harvesting-aware power management for real-time systems with renewable energy. IEEE Trans. Very Large Scale Integr. Syst. **20**(8), 1473–1486 (2011)
58. Lockhart, J.W., Weiss, G.M., Xue, J.C., Gallagher, S.T., Grosner, A.B., Pulickal, T.T.: Design considerations for the WISDM smart phone-based sensor mining architecture. In: Proceedings of the Fifth International Workshop on Knowledge Discovery from Sensor Data, pp. 25–33 (2011)
59. Louizos, C., Welling, M., Kingma, D.P.: Learning sparse neural networks through l_0 regularization (2017). arXiv preprint arXiv:1712.01312
60. Maetzler, W., Domingos, J., Srulijes, K., Ferreira, J.J., Bloem, B.R.: Quantitative wearable sensors for objective assessment of Parkinson's disease. Mov. Disord. **28**(12), 1628–1637 (2013)
61. Maetzler, W., et al.: A clinical view on the development of technology-based tools in managing Parkinson's disease. Mov. Disord. **31**(9), 1263–1271 (2016)
62. Mann, S.: Wearable computing: Toward humanistic intelligence. IEEE Intell. Syst. **16**(3), 10–15 (2001)
63. Mathur, A., Zhang, T., Bhattacharya, S., Velickovic, P., Joffe, L., Lane, N.D., Kawsar, F., Lió, P.: Using deep data augmentation training to address software and hardware heterogeneities in wearable and smartphone sensing devices. In: 2018 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), pp. 200–211. IEEE, Piscataway (2018)
64. Mayer, P., Magno, M., Benini, L.: Energy-positive activity recognition-from kinetic energy harvesting to smart self-sustainable wearable devices. IEEE Trans. Biomed. Circuits Syst. **15**(5), 926–937 (2021)
65. McCarthy, M.W., James, D.A., Lee, J.B., Rowlands, D.D.: Decision-tree-based human activity classification algorithm using single-channel foot-mounted gyroscope. Electron. Lett. **51**(9), 675–676 (2015)
66. Mizell, D.: Using gravity to estimate accelerometer orientation. In: Seventh IEEE International Symposium on Wearable Computers, 2003. Proceedings, pp. 252–252. Citeseer (2003)
67. Molchanov, D., Ashukha, A., Vetrov, D.: Variational dropout sparsifies deep neural networks. In: International Conference on Machine Learning, pp. 2498–2507. PMLR (2017)
68. Nguyen, S., Amirtharajah, R.: A hybrid rf and vibration energy harvester for wearable devices. In: 2018 IEEE Applied Power Electronics Conference and Exposition (APEC), pp. 1060–1064. IEEE, Piscataway (2018)
69. Noh, D.K., Kang, K.: Balanced energy allocation scheme for a solar-powered sensor system and its effects on network-wide performance. J. Comput. Syst. Sci. **77**(5), 917–932 (2011)
70. Odema, M., Rashid, N., Al Faruque, M.A.: Energy-aware design methodology for myocardial infarction detection on low-power wearable devices. In: ASPDAC, pp. 621–626 (2021)
71. Ozanne, A., Johansson, D., Hällgren Graneheim, U., Malmgren, K., Bergquist, F., Alt Murphy, M.: Wearables in epilepsy and Parkinson's disease—a focus group study. Acta Neurol. Scand. **137**(2), 188–194 (2018)
72. Park, E., Yoo, S., Vajda, P.: Value-aware quantization for training and inference of neural networks. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 580–595 (2018)
73. Park, J., Bhat, G., Geyik, C.S., Ogras, U.Y., Lee, H.G.: Energy per operation optimization for energy-harvesting wearable IoT devices. Sensors **20**(3), 764 (2020)
74. Piorno, J.R., Bergonzini, C., Atienza, D., Rosing, T.S.: Prediction and management in energy harvested wireless sensor nodes. In: Int. Conf. on Wireless Comm., Vehicular Tech., Info. Theory and Aerospace & Electron. Syst. Tech., pp. 6–10 (2009)

75. Pires, I.M., Hussain, F., Garcia, N.M., Zdravevski, E.: Improving human activity monitoring by imputation of missing sensory data: experimental study. Fut. Internet **12**(9), 155 (2020)
76. Prabowo, O.M., Mutijarsa, K., Supangkat, S.H.: Missing data handling using machine learning for human activity recognition on mobile device. In: 2016 International Conference on ICT for Smart Society (ICISS), pp. 59–62. IEEE, Piscataway (2016)
77. Raethjen, J., Govindan, R., Muthuraman, M., Kopper, F., Volkmann, J., Deuschl, G.: Cortical correlates of the basic and first harmonic frequency of parkinsonian tremor. Clin. Neurophysiol. **120**(10), 1866–1872 (2009)
78. Reiss, A., Stricker, D.: Creating and benchmarking a new dataset for physical activity monitoring. In: Proceedings of the 5th International Conference on PErvasive Technologies Related to Assistive Environments, pp. 1–8 (2012)
79. Saeed, A., Ozcelebi, T., Lukkien, J.: Synthesizing and reconstructing missing sensory modalities in behavioral context recognition. Sensors **18**(9), 2967 (2018)
80. Sharma, A., Kakkar, A.: A review on solar forecasting and power management approaches for energy-harvesting wireless sensor networks. Int. J. Commun. Syst. **33**(8), e4366 (2020)
81. Sharma, N., Sharma, P., Irwin, D., Shenoy, P.: Predicting solar generation from weather forecasts using machine learning. In: 2011 IEEE International Conference on Smart Grid Communications (SmartGridComm), pp. 528–533. IEEE, Piscataway (2011)
82. Shoaib, M., et al.: A survey of online activity recognition using mobile phones. Sensors **15**(1), 2059–2085 (2015)
83. Shoaran, M., Haghi, B.A., Taghavi, M., Farivar, M., Emami-Neyestanak, A.: Energy-efficient classification for resource-constrained biomedical applications. IEEE J. Emerging Sel. Topics Circuits Syst. **8**(4), 693–707 (2018)
84. Smith, K.E., Smith, A.O.: Conditional GAN for timeseries generation (2020). arXiv preprint arXiv:2006.16477
85. Stisen, A., Blunck, H., Bhattacharya, S., Prentow, T.S., Kjærgaard, M.B., Dey, A., Sonne, T., Jensen, M.M.: Smart devices are different: Assessing and mitigating mobile sensing heterogeneities for activity recognition. In: Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems, pp. 127–140 (2015)
86. Tang, J., Deng, C., Huang, G.B.: Extreme learning machine for multilayer perceptron. IEEE Trans. Neural Netw. Learn. Syst. **27**(4), 809–821 (2015)
87. Texas Instruments Inc.: CC2652R Microcontroller (2018). https://www.ti.com/product/CC2652R. Accessed 1 Nov 2020
88. Tokognon, C.A., Gao, B., Tian, G.Y., Yan, Y.: Structural health monitoring framework based on internet of things: a survey. IEEE Internet Things J. **4**(3), 619–635 (2017)
89. Tsanas, A., Little, M.A., McSharry, P.E., Spielman, J., Ramig, L.O.: Novel speech signal processing algorithms for high-accuracy classification of Parkinson's disease. IEEE Trans. Biomed. Eng. **59**(5), 1264–1271 (2012)
90. Tuncel, Y., Bandyopadhyay, S., Kulshrestha, S.V., Mendez, A., Ogras, U.Y.: Towards wearable piezoelectric energy harvesting: Modeling and experimental validation. In: Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design, pp. 55–60 (2020)
91. Uddin, M.H., Ara, J.M.K., Rahman, M.H., Yang, S.: Neural network pruning: an effective way to reduce the initial network for deep learning based human activity recognition. In: 2021 International Conference on Electronics, Communications and Information Technology (ICECIT), pp. 1–4. IEEE, Piscataway (2021)
92. Ullrich, K., Meeds, E., Welling, M.: Soft weight-sharing for neural network compression (2017). arXiv preprint arXiv:1702.04008
93. Um, T.T., Pfister, F.M., Pichler, D., Endo, S., Lang, M., Hirche, S., Fietzek, U., Kulić, D.: Data augmentation of wearable sensor data for Parkinson's disease monitoring using convolutional neural networks. In: Proceedings of the 19th ACM International Conference on Multimodal Interaction, pp. 216–220 (2017)
94. Valenzuela, A.: Energy Harvesting for No-Power Embedded Systems (2008). https://bit.ly/3fnA6Vm. Accessed 28 Mar 2021

95. Vasisht, D., et al.: Farmbeats: an Iot platform for data-driven agriculture. In: USENIX NSDI, pp. 515–529 (2017)
96. Vigorito, C.M., Ganesan, D., Barto, A.G.: Adaptive control of duty cycling in energy-harvesting wireless sensor networks. In: Proc. Conf. on Sensor, Mesh and Ad Hoc Comm. and Networks, pp. 21–30 (2007)
97. Volpi, R., Namkoong, H., Sener, O., Duchi, J.C., Murino, V., Savarese, S.: Generalizing to unseen domains via adversarial data augmentation. In: Advances in Neural Information Processing Systems, vol. 31 (2018)
98. Wang, A., et al.: A comparative study on human activity recognition using inertial sensors in a smartphone. IEEE Sensors J. **16**(11), 4566–4578 (2016)
99. Weenk, M., Bredie, S.J., Koeneman, M., Hesselink, G., van Goor, H., van de Belt, T.H., et al.: Continuous monitoring of vital signs in the general ward using wearable devices: randomized controlled trial. J. Med. Internet Res. **22**(6), e15471 (2020)
100. Weenk, M., van Goor, H., Frietman, B., Engelen, L.J., van Laarhoven, C.J., Smit, J., Bredie, S.J., van de Belt, T.H., et al.: Continuous monitoring of vital signs using wearable devices on the general ward: pilot study. JMIR mHealth uHealth **5**(7), e7208 (2017)
101. Woods, A.M., Nowostawski, M., Franz, E.A., Purvis, M.: Parkinson's disease and essential tremor classification on mobile device. Pervasive Mobile Comput. **13**, 1–12 (2014)
102. Xiao, Y., Niyato, D., Han, Z., DaSilva, L.A.: Dynamic energy trading for energy harvesting communication networks: a stochastic energy trading game. IEEE J. Sel. Areas Commun. **33**(12), 2718–2734 (2015)
103. Xiao, Y., Niyato, D., Wang, P., Han, Z.: Dynamic energy trading for wireless powered communication networks. IEEE Commun. Mag. **54**(11), 158–164 (2016)
104. Yamin, N., Bhat, G.: Online solar energy prediction for energy-harvesting Internet of Things devices. In: 2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), pp. 1–6. IEEE, Piscataway (2021)
105. Yang, J., Liu, W., Yuan, J., Mei, T.: Hierarchical soft quantization for skeleton-based human action recognition. IEEE Trans. Multimedia **23**, 883–898 (2020)
106. Yang, Z., Raymond, O.I., Zhang, C., Wan, Y., Long, J.: DFTerNet: towards 2-bit dynamic fusion networks for accurate human activity recognition. IEEE Access **6**, 56750–56764 (2018)
107. Ye, J., Li, K., Qi, G.J., Hua, K.A.: Temporal order-preserving dynamic quantization for human action recognition from multimodal sensor streams. In: Proceedings of the 5th ACM on International Conference on Multimedia Retrieval, pp. 99–106 (2015)
108. Yona, A., Senjyu, T., Funabashi, T.: Application of recurrent neural network to short-term-ahead generating power forecasting for photovoltaic system. In: 2007 IEEE Power Engineering Society General Meeting, pp. 1–6 (2007). https://doi.org/10.1109/PES.2007.386072
109. Yoneyama, M., Kurihara, Y., Watanabe, K., Mitoma, H.: Accelerometry-based gait analysis and its application to Parkinson's disease assessment—part 2: a new measure for quantifying walking behavior. IEEE Trans. Neural Syst. Rehabil. Eng. **21**(6), 999–1005 (2013)
110. Yoon, J., Jordon, J., Schaar, M.: GAIN: missing data imputation using generative adversarial nets. In: ICML, pp. 5689–5698 (2018)
111. Yu, S., Li, Z., Chen, P.Y., Wu, H., Gao, B., Wang, D., Wu, W., Qian, H.: Binary neural network with 16 mb RRAM macro chip for classification and online training. In: 2016 IEEE International Electron Devices Meeting (IEDM), pp. 16–2. IEEE, Piscataway (2016)
112. Zhang, M., Sawchuk, A.A.: USC-HAD: a daily activity dataset for ubiquitous activity recognition using wearable sensors. In: Proc. of the Conf. on Ubiquitous Comput., pp. 1036–1043 (2012)
113. Zhu, M., Gupta, S.: To prune, or not to prune: exploring the efficacy of pruning for model compression (2017). arXiv preprint arXiv:1710.01878

Efficient Deep Vision for Aerial Visual Understanding

Rafael Makrigiorgis, Shahid Siddiqui, Christos Kyrkou, Panayiotis Kolios, and Theocharis Theocharides

1 Introduction

Embedded visual AI is a growing trend in applications requiring low latency, real-time decision support, increased robustness and security [14, 15]. An example of this is the increasing use of Unmanned Aerial Vehicles (UAVs) for a number of applications as a remote sensing platform, such as road traffic monitoring [19, 23], search and rescue [25], and precision agriculture [24]. Recent technological advances such as the integration of camera sensors with onboard processing provide the opportunity for new UAV applications such as (i) detecting and classifying infrastructure faults during routine inspections, (ii) identifying and tracking people of interest, locating objects, and flagging unusual situations, (iii) locating people who are lost, etc.

The aforementioned capabilities are enabled by recent advances in deep learning-based scene understanding, and convolutional neural networks (CNNs) in particular, that provide remarkable opportunities for computer vision-based applications. In such scenarios, the vision algorithms need to deploy on resource-constrained embedded devices on UAVs that support high-resolution cameras for applications beyond photography, such as environmental and infrastructure monitoring. However, deep learning algorithms are computationally very intensive and not suitable for onboard processing on small devices such as UAVs due to battery/power constraints and are usually processed via the cloud. For certain scenarios, however, where the system operates in remote areas with limited connectivity, this can result in unwanted response latency which can degrade performance, a potentially catastrophic scenario in safety-critical applications. Thus, processing information

R. Makrigiorgis · S. Siddiqui · C. Kyrkou · P. Kolios · T. Theocharides (✉)
KIOS Research and Innovation Center of Excellence, University of Cyprus, Nicosia, Cyprus
e-mail: makrigiorgis.rafael@ucy.ac.cy; siddiqui.muhammad-shahid@ucy.ac.cy; kyrkou.christos@ucy.ac.cy; kolios.panayiotis@ucy.ac.cy; ttheocharides@ucy.ac.cy

S. Pasricha, M. Shafique (eds.), *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing*, https://doi.org/10.1007/978-3-031-40677-5_4

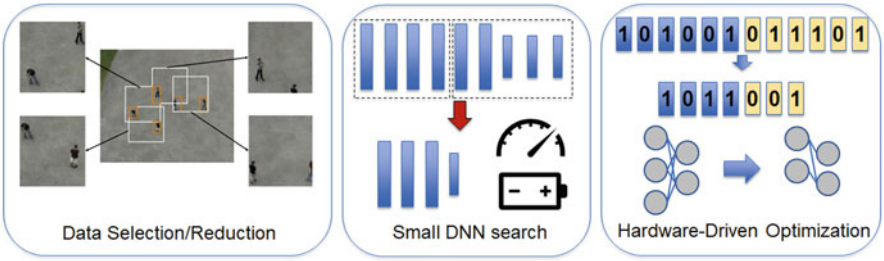


Fig. 1 Efficient visual understanding for UAVs requires optimizations at different levels, from data reduction to deep neural network architecture exploration, and hardware-driven model adjustments

at the edge can not only eliminate unwanted lag issues but also partially handle security problems since the data are not transmitted and sensitive data cannot be intercepted.

A lot of research has been conducted to address the challenges of visual perception using UAV imagery. A few notable examples are detecting vehicles for traffic monitoring scenarios in [13] and disaster management in [1]. Processing visual data information has seen significant accuracy and performance improvements due to advancements in deep learning and technologies in Graphical Processing Units (GPUs). However, relying on hardware improvements alone is not by itself sufficient to provide the optimal power/performance trade-offs necessary for edge and IoT applications. Hence, in this chapter, we highlight a body of work that aims to explore different techniques that in tandem with embedded hardware improvements can lead to better efficiency for edge applications. The described techniques as depicted in Fig. 1 provide a way toward a more holistic optimization by considering all aspects from the data to the AI model, as well as hardware aspects. These techniques are presented in the subsequent chapters where improved performance and efficiency are demonstrated compared to standard approaches.

2 Domain-Specific Small ConvNets for UAV Applications

Typically, we use hardware accelerators to speed up compressed and quantized versions of well-known models because the underlying operations are highly parallel. What if we also try and optimize the model directly for the application by only using the right amounts and types of each operation? In this section, we will demonstrate some use cases where a small neural network is capable of providing adequate performance.

Small deep neural networks have many desirable properties and advantages that we can see across their lifetime cycle and make them more easily deployable on embedded processors where computation and even more so memory are at a

premium. In addition, being able to store the model on-chip saves on power as it avoids off-chip memory access which consumes order of magnitudes more power. In addition, to inference, small neural networks facilitate faster training iterations and more easily updatable over the air (OTA). So whenever a remote system encounters a situation where it is not confident in its predictions or may require retraining, then we can send less data even from a lower bandwidth network. Finally, when using smaller neural networks, it is easier for multiple vision tasks to run on the same platform, e.g., object detection and classification.

But it is not only the computational and technical characteristics that need to be considered, but many of the operational characteristics also change when considering edge applications. In most cases, these are narrow-domain applications that need to recognize a few classes compared to the generic models trained on ImageNet [6], and they have requirements for real-time processing so that a decision or action can be taken in reasonable time and need to operate under limited energy and resources. So, for many applications, the full capacity of ImageNet pretrained models could be unnecessary which provides opportunities to explore smaller deep learning models in edge applications.

The main principles behind the design of smaller models, as shown in Fig. 2, are the balance between the number of parameters, the downsampling rate, and the operations performed within the network. The number of parameters usually impacts how many convolutions we have in the model, and this can affect the FLOP demands. Aggressive downsampling can harm the accuracy of a model even though it makes it faster. While the number of channels and the type of convolution also impact the performance. Next, two use cases for UAV applications are presented where smaller neural networks with architectural modifications are capable of providing competitive performance compared to traditional deep learning models.

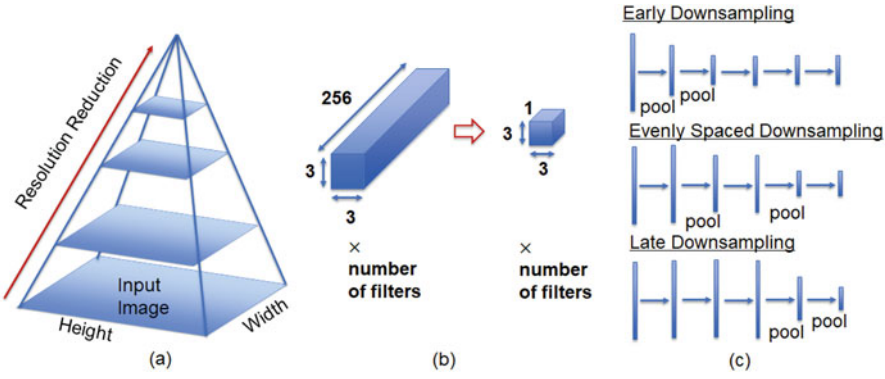


Fig. 2 Main approaches for producing smaller and more efficient neural networks. (**a**) Reducing input image resolution. (**b**) Using efficient operators like depthwise convolution if it is implemented efficiently on the underlying hardware. (**c**) Early downsampling of feature maps can improve processing speed

2.1 Disaster Classification

The first use case will demonstrate the use of UAVs for patrolling and automated recognition of disaster events. When a disaster strikes, there is limited time to act, and hence, the objective is to develop an automated platform for rapid deployment and large area coverage. Coupled with an automated path planning software, a UAV can navigate a monitored area and automatically recognize different events through onboard processing of its camera feed. Since connectivity is not guaranteed in such cases, this should be done in a way that is suitable for the embedded hardware of UAVs and in real time.

2.1.1 Network Design

For this application, a small neural network is designed that features a balance between the number of parameters, the downsampling rate, and the operations performed within the network. First, we design a basic block that is suited for processing images where the object/area appears at varying resolutions such as in UAV imagery. Specifically, this block is referred to as **A**trous **C**onvolutional **F**eature **F**usion (ACFF) block (shown in Fig. 3) that relies on depthwise atrous/dilated convolutions to aggregate context information at multiple resolutions. Each dilated convolution is factored into depthwise convolution that performs lightweight filtering by applying a single convolutional kernel per input channel to reduce the computational complexity. The intuition is to take advantage of the different dilation rates since each path may peek up features at different object/region resolution due to changes in altitude. Another advantage of using dilated convolutions is that the same number of parameters and computations are needed regardless of the size of the kernels in the block. Prior to processing the input feature map, a 1×1 convolution filter is applied to reduce its size and then expand it after the fusion of the dilated convolutions.

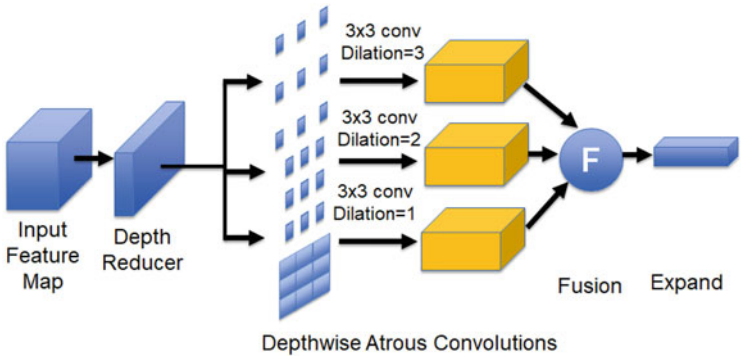


Fig. 3 Atrous Convolutional Feature Fusion Block

Other properties of the network geared toward more efficient processing on edge devices are reduced number of 16 channels at the first layer and downsampling with strided convolutions. This is particularly important in achieving better performance since the image resolution at this stage is at the highest. This first convolutional block is a standard convolution. Then, the network follows a canonical architecture of ACFF blocks with a progressive reduction of spatial resolution with an increase in depth with up to 256 channels at the last layer. To further reduce the number of parameters, the fully connected layers that are usually employed in classification networks are replaced with global pooling operations. Finally, the network depth is maintained at 7 blocks since deeper networks resulted in saturating performance. Additionally, modifications were made to the activation function in order to make it more amicable to quantization. First, the leaky variant of the ReLU activation is used that permits for the gradient to flow for negative input values. In addition, the maximum activation is capped to the value of 255 to allow for easier quantization to 8 bits. Finally, we have two operating modes; during training, the full range of the capped ReLU is used, while during inference, the negative values are zeroed. These minor modifications allow optimizing for lower precision execution. We refer to this architecture as *EmergencyNet* [18].

2.1.2 Experiments and Results

In this section, the experimental evaluation of small neural network is discussed with results from the experimental evaluation of the approach on an actual embedded platform. First, the improvements over existing networks are validated on the developed dataset from [17], and results of the effectiveness are presented. Real experiments have also been conducted in two different settings: (i) onboard embedded processing, where all computations are performed onboard the resource-constrained UAV device, and (ii) remote processing, in which the UAV transmits the captured video to the controller ground station for processing on an Android tablet that controls the UAV. Herein, we focus on the former. It is worth noting that the primary interest is in single image processing speed and as such the evaluation phase is carried out with a batch size of 1 since this is common in real-time streaming applications where the camera outputs each frame sequentially (Table 1).

The small neural network is compared with some standard networks, and results are summarized in Table 1. First, with regard to the accuracy of the pretrained models, it is observed that *VGG16* outperforms all of them with a 96.4% F1-score with *ResNet50* closely following with 96.1%. However, both networks have very high demands for computational and storage requirements making them unsuitable for resource constraint systems and real-time use. The latest iteration of MobileNet, i.e., V3, achieves the highest accuracy among the MobileNet family of networks with a score of 95.3%. However, it requires an order of magnitude more parameters and memory. Other MobileNet versions (V1 and V2) demonstrate similar score with the V2 version resulting in a slightly higher FPS. *EfficientNet* provides a high accuracy of 96.0% due to its elaborate architecture. However, the parameter count

Table 1 Comparison with existing approaches

Model	Parameters	Memory (MB)	F1 Score (%)	FPS (1/s)
VGG16 [30]	14,849,349	59.39	96.4	1.1
ResNet50 [8]	24,113,541	96.4	96.1	2.9
MobileNet V1 [10]	3,492,549	13.9	95	7.9
MobileNet V2 [29]	2,587,205	10.3	95.2	9.3
MobileNet V3 [9]	3,046,037	12.1	95.3	9.0
EfficientNet (B0) [32]	4,378,785	17.5	96.0	10.4
SqueezeNet [11]	698,917	2.7	91.5	6.6
ShuffleNet [22]	4,282,425	17.1	91.1	4.5
Xception [5]	21,387,309	85.549	95.3	2.3
Fire_Net [33]	5,235,860	5.2	90.5	9.8
EmergencyNet [18]	90,892	0.368	95.7	24.3

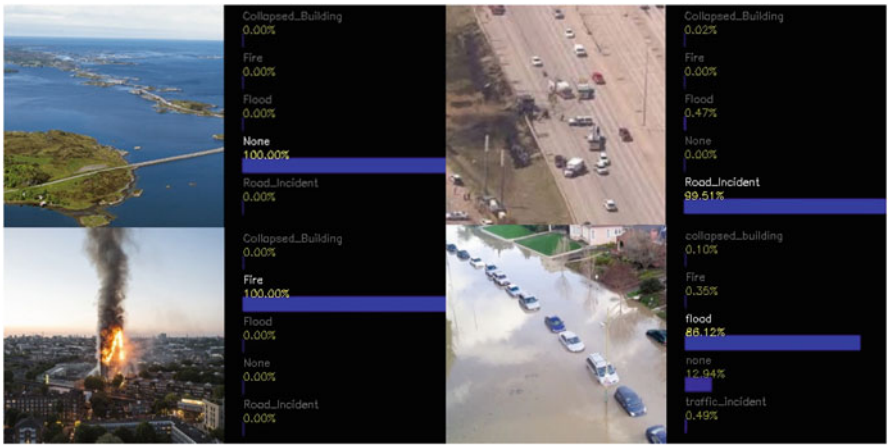


Fig. 4 Example of classification results on sample images

and memory requirements are higher than EmergencyNet. Other networks fail to provide adequate accuracy and require a higher number of parameters. It is clear from this analysis that it is worth investigating specifically tailored solutions for resource-constrained applications in order to provide an improvement across all design aspects.

Furthermore, some classification results are shown in Fig. 4. It is interesting to note that similarly appearing patterns between images do not confuse the network. For instance, the presence of cars, which is more often associated with *traffic incidents*, does not cause the network to fail which outputs the correct classification as *flood*. Overall, these results are promising since a small network manages to match or at least be very close to other more general networks while the processing speed and subsequent frame-rate improvements provide adequate trade-offs for edge applications.

2.2 Vehicle Detection

In the second use case, we describe how the problem of top view vehicle detection from UAV aerial imagery is tackled through the exploration of small convolutional neural networks. While there has been quite some research on reducing the complexity of well-studied ConvNet models in the form of parameter compression and quantization, there has been little effort on developing specialized solutions for resource-constrained embedded vision systems such as UAVs. This section briefly outlines the end-to-end investigation of different single-shot CNN detectors for drone-based vehicle detection.

2.2.1 Network Design and Approach

Our approach in [16] focuses on the exploration of an efficient and lightweight network by investigating different network design considerations. The goal is to keep the accuracy as high as possible but design a faster model. We develop several models to explore the effect of a different number of parameters. The models are trained to detect only 1 class, which, in our case, is vehicles viewed from top. We explore the impact on performance by changing the structure of the network such as the number of filters, the number of layers, image size, the number of convolution, and the pooling layers. The design approaches considered are the following: (1) number and size of filters. Firstly, we use pooling layers sparingly and use a small number of filters in each layer in order to get a smaller network which leads to a faster detection. The largest number of filters in a network is 256 for the final convolutional layer. As the network goes deeper, we double the number of filters. In total, there are 9 convolutional and 4 max-pooling layers. (2) Input Image Size: Input size is a parameter that can increase accuracy but decreases the detection speed since a deeper network will be required to reach the final bounding box regression size. Moreover, a larger input image implies more convolutions per feature map and in some cases more bounding boxes to account for.

Four network architectures are derived (SmallYoloV3, TinyYoloVoc, TinyYoloNet, and DroNet). These have different parameters including the layers and the type of each layer (conv, maxpool, detection) together with the configuration of the layers in terms of the number of filters, the size of filters in each layer, and the input/output size of the feature maps. A common theme, however, is the use of 3×3 and cheaper 1×1 convolutional filters, as well as the progressive reduction of the feature maps size by a factor of 2 and use of a lower number of filters at early layers. Finally, feature shortcut connections are used to further improve accuracy for small objects.

To find the CNN that optimizes both accuracy and computation cost, a custom metric is employed. Given a model instance, it captures both the detection accuracy and the achieved runtime on the target hardware platform. It is defined as a composite linear combination metric that combines various performance indicators.

By following this methodology, the resulting CNN model yields the highest performing balance between detection accuracy and faster execution.

$$Score(w) = w_1 \times FPS + w_2 \times IoU + w_3 \times Sensitivity + w_4 \times Precision \quad (1)$$

The score is parameterized with respect to a vector of weights which sums to one, where each weight captures the application-level importance of each metric and is normalized between [0-1]. Since real-time performance is desired, the FPS is prioritized with a weight of 0.4 over other three accuracy-related metrics which are equally weighted with 0.2.

2.2.2 Results

In this section, we present a comprehensive quantitative evaluation of the four CNN architectures. The basic network models are trained and tested for various input sizes using the constructed vehicle dataset as shown in Fig. 5. Table 2 shows the performance comparison of these models. In our test set, with 386 × 386 as input resolution, TinyYoloNet achieves 10× higher performance than TinyYoloVoc with decreased detection sensitivity and precision by 20% and 10%, respectively, and an IoU drop of 0.11. The network SmallYoloV3, with 386 × 386 resolution, achieves the highest frame rate of 23 FPS among all network designs. Nevertheless, the substantial reduction in the number of weights leads to a decrease in sensitivity

Fig. 5 Example of detection results on sample images

Table 2 Comparison of the different models

Model	TinyYOLOVoc	TinyYOLONet	SmallYOLOv3	DroNet
FPS	1.2	12	23	22
IoU	0.36	0.25	0.11	0.22
Sensitivity(%)	87.8	68.2	34.7	69.4
Precision (%)	90.5	80.6	69.1	76.5
Size (MB)	63.1	6.4	0.115	0.283
Score	0.62	0.68	0.69	**0.83**

which is 53% lower and prohibits us from using it for robust vehicle detection. There are significant performance gains starting from TinyYoloVoc to TinyYoloNet followed by SmallYoloV3 and then to DroNet. For example, comparing these models for the same input size of 386, the performance of DroNet is 30× faster compared to TinyYoloVoc with a minimal drop of 0.08 on the IoU. Moreover, there is a limited drop of 2% and 6% for the detection sensitivity and precision, respectively.

Comparing DroNet with different models demonstrates the effect of a composite score function. In particular, with respect to the smaller smallYoloV3 network, it provides lesser FPS while being more accurate. On the other hand, compared with the largest tinyYoloVOC, it is less accurate but much faster. The model size is also suitable for edge applications with limited resources. Overall, it provides the best trade-off between accuracy and performance.

The analysis is performed on the Odroid-XU4 with an Octa-core Samsung Exynos-5422 CPU which is lightweight and capable of being powered by the UAV platform. Overall, the DroNet network maintains a performance of 10 FPS with the accuracy maintained around 95% for 512×512 image resolution.

3 Processing Aerial Images with Tiling

The previous section dealt with design of small, domain-specific deep neural networks. However, optimizing the network alone might not be enough to obtain a high performing system. In many cases, when a UAV flies at a high altitude (e.g., 500 meters) and covers a wide field of view, the image resolution must be large enough to recognize targets. High-resolution images, however, imply an exponential increase in the amount of data processed, and most of the times there is a lot of redundant data. Hence, even a small neural network would require significant time to process such high-resolution images. Thus, it is necessary to investigate intelligent data reduction techniques. Some works have attempted to improve UAV imagery detection utilizing hybrid approaches such as combining deep learning with SVM [2] or two-stage Faster R-CNN [4]. However, embedded devices still do not have the necessary processing power to process such high-resolution frames in real time. Recent works have also focused on developing low-power hardware or designing CNNs architectures to retain as much accuracy as possible with real-time processing capabilities. However, such optimizations may require a fixed input size that is usually multiple times smaller than a desired high-resolution image.

From a UAV's perspective, objects may appear too small and hence challenging to detect. Therefore, approaches beyond CNN architectural design should be implemented to maximize the efficiency of such applications by reducing the data to be processed without losing significant information from the input images.

The work presented herein focuses on optimizing the accuracy and performance of onboard UAV object detection. Several approaches have been developed and tested using tiling methods. Tiling is a technique, where, upon receiving an image

Fig. 6 Proposed tiles for processing base on the selection process. *This figure was taken from our previous work in [27]*

frame, overlapping slices of the image are created and processed individually or in parallel (e.g., Fig. 6). Using this technique, we avoid resizing the original image, keeping all the information and features of the image at its original resolution. For this purpose, we propose the *EdgeNet* framework [27]. *EdgeNet* framework can be utilized with predefined architectures even on higher resolution images, and it is an intelligent way to minimize processing data while increasing accuracy and system performance. *EdgeNet* framework consists of three main stages, Initial Position Estimation, Tiling and CNN Selection, and Tracking using Optical Flow.

3.1 Approach

The presented approach, referred to as the EdgeNet framework, aims to simultaneously improve the overall accuracy and performance while reducing the power consumption when processing high-resolution images on an edge device. The EdgeNet framework's main idea is to utilize a pool of CNN detectors and select the optimal one based on the needs of each frame to be processed. Specifically, by scheduling different processing stages, where different CNNs are used, and allocating a varying number of times steps to each stage, we can obtain a good trade-off between accuracy and speed. An evaluation of multiple algorithmic configurations and parameters is possible through which to identify the time allocation for each stage in the framework and accordingly analyze the accuracy and the performance of each configuration. The EdgeNet framework is comprised of three-stage pipeline as depicted in Fig. 7. Following is the description of each individual stage.

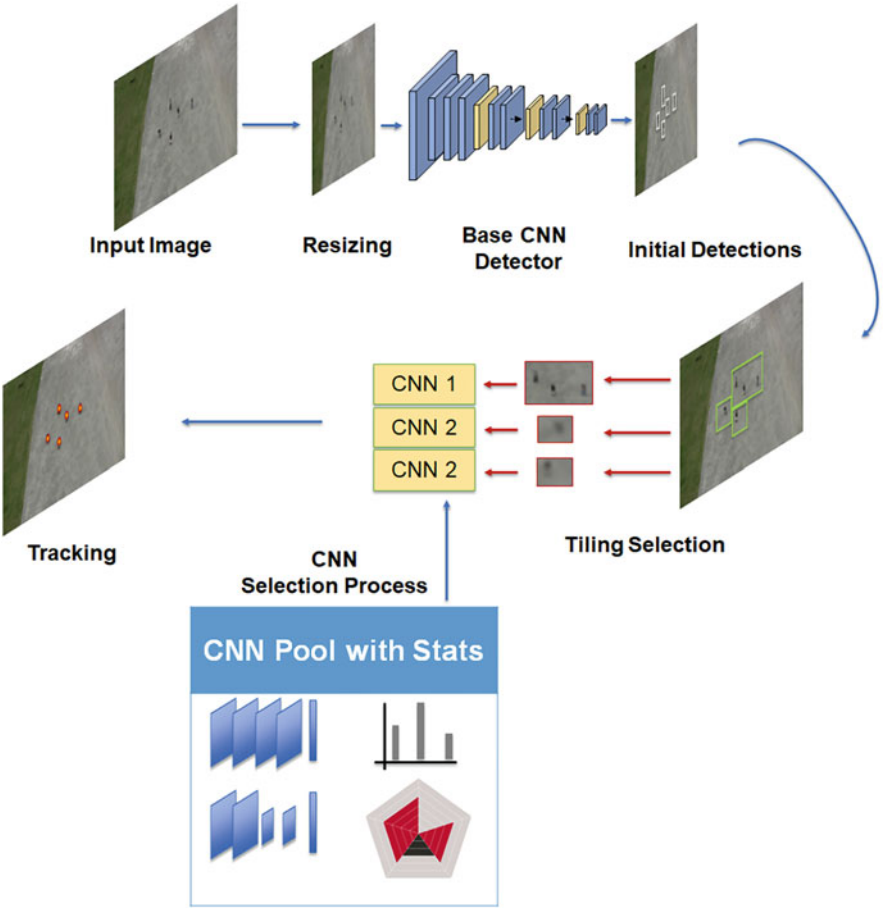


Fig. 7 EdgeNet Framework Pipeline adapted from [27]

3.1.1 Initial Position Estimation

The purpose of this stage is to help the framework choose the necessary method for the following stages by constructing the initial positions of the objects in a frame. For this task, the efficiency of a Convolutional Neural Network is needed, and hence, *DroNet* which is aforementioned is utilized. However, the structure of DroNet is extended by utilizing upsample feature maps from previous layers, to achieve detection of objects in multiple sizes. This method sufficiently improves the detector's accuracy [28] for small objects, such as pedestrians. From now on, this network is referred to as $DroNet\_V3$. For this stage, a more traditional way is utilized, where the input frame is resized and then passed through $DroNet\_V3$ producing the bounding boxes of the detected objects in the image.

3.1.2 Tiling and CNN Selection

The purpose of the second stage of the framework is to effectively reduce the data that needs to be processed. To achieve this, distinct tiles around initial target positions are selected to figure out the minimum region of the image that needs processing. The selection method uses the detected positions of the objects that were previously estimated. For illustration purposes, the proposed DroNet is utilized because it can operate on multiple input sizes depending on the tile size, ranging from 128 to 512, and will be referred to as $DroNet\_Tile$. In order to select an appropriately sized model, we need to first perform a profiling and benchmark CNNs with the distinct input sizes, as shown in Table 3. The best CNNs out of the pool will be chosen, for each time period, to guarantee the least processing time.

Furthermore, the amount of objects that are included in a tile is utilized as a guiding factor for the selection. The candidate tiles that cover each object must be identified by this procedure. Hence, a number of tiles are created for each of the detected boxes, as seen by the first stage in Fig. 7. The detected objects are positioned at the corners of the tiles as shown in Fig. 8. Then, different sizes of tiles are also generated, more specifically five total sizes are used: 128, 256, 352, 416, and 512, which matches the different sizes in the CNN pool. A total of 20 tiles are generated for each proposed object. Then, each tile is evaluated using a selection

Table 3 Processing time of $DroNet\_V3$ and $DroNet_Tile$ for distinct input sizes

CNN	Input size (pixels)	Processing time (sec.)
DroNet_V3	512	0.08
DroNet_Tile	512	0.03
DroNet_Tile	416	0.02
DroNet_Tile	352	0.014
DroNet_Tile	256	0.008
DroNet_Tile	128	0.002

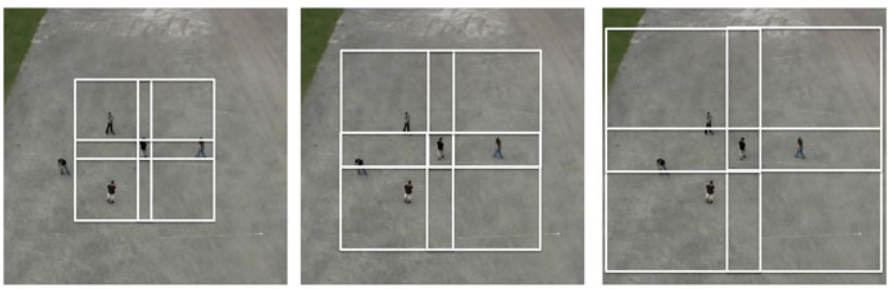


Fig. 8 Different tile proposals, with respect to the position and size of the tiles, for an object in the image. (**a**) 128 × 128, (**b**) 256 × 256, and (**c**) 352 × 352 adapted from [27]

process based on the objects it covers and its associated processing time. Also, an Effective Processing Time (EPT) is calculated for each one of the 20 tiles for each object. Basically, EPT is the number of object covered, divided by its corresponding processing time that can be found in Table 3. The selected tile is the one that achieves the minimum EPT. Then, we combine all the tiles that are created, discard those which cover duplicate or with fewer objects and are finally left with those with the minimum EPT.

An example is depicted in Fig. 6 where 128×128 is selected from the selection process. $DroNet$ with 128 size is used to process each tile. The total processing time for this particular example is $4 \times 0.002 = 0.008$ s. Compared to $DroNet\_V3$, which takes 0.05 s to detect, it shows significant performance boost.

3.1.3 Optical Flow-Based Tracker

This is the third and the last stage of the EdgeNet framework. This tracker was selected due to its fast execution time even with a huge amount of tracked points in an image. It uses Lucas–Kanade [21] optical flow tracker, which solves the basic optical flow equations for all the pixels by the least squares criterion. Its principle is that the motion of objects in sequential frames is almost constantly relative to the given object. Apart from this selection though, any other tracker can be utilized on this stage, but it will affect the accuracy or performance, respectively, depending on the tracker and the requirements of the application. The purpose of this stage is (1) to simultaneously track objects along with stages 1 and 2 and cooperatively be used to verify the location of the objects and (2) to reduce the overall processing time of EdgeNet by utilizing the tracking for a few iterations prior to needing to look at the whole image again. The tracker operates on each of the detected boxes by calculating a centered point for each. All of these points are also utilized as the starting points of the tracker along with the corresponding image. A time-slot combination is selected that determines the amount of times that each process will be executed based on the application requirements and the processing specifications of the device the framework runs on.

3.2 Evaluation of EdgeNet Framework

In this section, an extensive evaluation of EdgeNet's framework using different configurations is presented. The configuration varies in the number of frames, which correlates to the amount of time, that is allocated for each stage. More specifically, we are using the notations N_S_1 - N_S_2 - N_S_3, to present the number of frames that can be allocated to each stage. For instance, if a time-slot combination of $1-5-5$ is selected, it means that the first stage will run for 1 frame, and the second and third stages will run for 5 frames each. Furthermore, we provide comparison and an extensive evaluation of three single-shot models $DroNet\_V3$, $DroNet\_Tile$, and

$TinyYoloV3$. Using this method, we show that any approach that does not utilize tiling or some form of dynamic selection has decreased accuracy performance due to the reduced image resolution. Also, we perform comparisons on three different devices that facilitate different use cases. The same dataset is used to train and test all the CNNs and initially compared on a low-end laptop without a GPU[1] and also tested on two embedded devices, an Odroid device[2] and a Raspberry Pi3. The dataset used consisted of 198 sequential images, captured from an aerial-view perspective and having 988 pedestrians in total.

3.2.1 Metrics

The following metrics are used to analyze and evaluate different approaches on the same test dataset:

Sensitivity (SEN) This metric is usually used to represent accuracy as it returns the percentage of the correctly classified objects. To calculate it, we divide the True Positives (T^{pos}) and False Negatives (F^{neg}) of the detected objects.

$$SEN = \frac{T^{pos}}{T^{pos} + F^{neg}} \tag{2}$$

Average Processing Time (APT) As the name suggests, this metric calculates the average time needed to process a single image from a sequence of images. To calculate it, we get the average processing time from all the $N^{test\_samples}$ test images, where t_i is the processing time for image i.

$$APT = \frac{1}{N^{test\_samples}} \times \sum_{i=1}^{N^{test\_samples}} t_i \tag{3}$$

Average Power Consumption (APC) This metric represents the input energy (measured in Watts) that is needed to process a single image from a sequence of images for a specific platform. It can be calculated by summing up the power consumption of each image divided by the total number of test images, where p_i is the energy consumption for an image i.

$$APC = \frac{1}{N^{test\_samples}} \times \sum_{i=1}^{N^{test\_samples}} p_i \tag{4}$$

[1] Quad Core 1.2 GHz Broadcom 64bit CPU.

[2] Samsung Exynos-5422 Cortex—A15 2 Ghz and Cortex—A7 Octa-core CPUs with Mali-T628 MP6 GPU.

3.2.2 Configuration Analysis

In this section, we first investigate how different configurations affect the overall performance and accuracy of each stage of the framework for the application of people detection. Since stage 1 is the most time-consuming one, we set its time allocation to 1 frame. For the remaining stages 2 and 3, the time allocation can vary. Figure 9 depicts the average sensitivity and processing time for multiple configurations for our in-house person test dataset. By analyzing this figure, we can notice that stage 3 impacts the performance of the framework as its time allocation increases. However, there is no drawback for sensitivity. On the other hand, when increasing the time spend on stage 2, the processing time decreases, but sensitivity decreases as well. In general, when increasing the time spend on stages 2 and 3, we delay the use of stage, 1 which utilizes $DroNet\_V3$ for full frame processing, and therefore, the overall processing time decreases. Thus, since stages 2 and 3 utilize initial target positions from stage 1, this also leads to a decreased sensitivity. As a result, choosing the right values for each frame is really important in order to have the optimal solution in terms of performance and accuracy. In our case, we want to have the highest possible accuracy with the higher possible performance, and therefore, we chose $EdgeNet1-3-5$ as the best configuration to perform analysis on edge platforms.

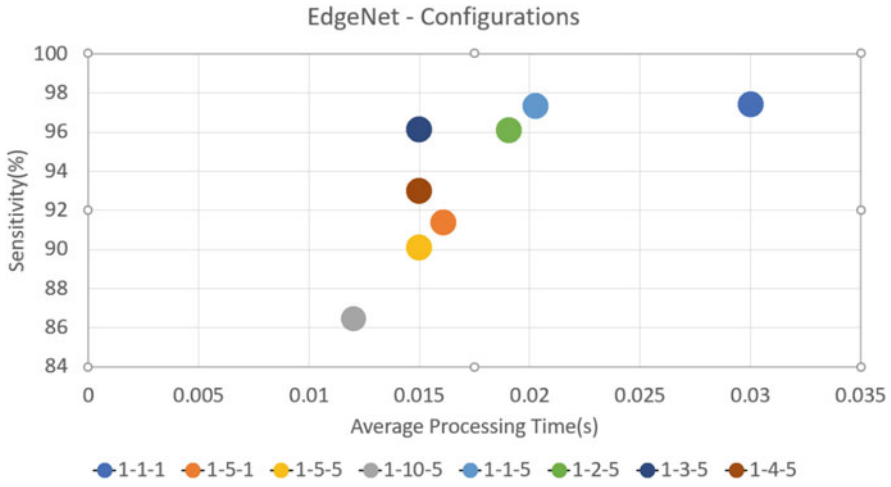


Fig. 9 Comparison of average processing time (CPU) and sensitivity between different EdgeNet configurations for different time frames for each stage. *This figure was taken from our previous work in [27]*

3.2.3 Performance Analysis on CPU, Odroid, and Raspberry Platforms

In this section, we evaluate the average processing time, power consumption, and sensitivity of $EdgeNet$ on multiple platforms, by comparing three single-shot CNNs, i.e., $DroNet\_Tile$, $DronNet\_V3$, and $Tiny-YoloV3$. The platforms used, as aforementioned, are a Raspberry Pi 3, an ODROID device,[3] and a low-end laptop CPU.[4]

Tables 4 and 5 demonstrate the average power consumption and average processing time results of the CNNs and $EdgeNet$ for all three platforms. As seen in the tables, the selected $EdgeNet$ configuration is leading in both inference speed and power consumption. As a result, with inference speeds below 0.06 s and consumption power ranging from 1.5 to 9 Watts, it achieves the goal of having a real-time framework for low-powered edge devices. Also, it is interesting to note that $DroNet\_Tile$ always comes off second best, while models such as $Tiny-YoloV3$, which are pretrained on massive datasets, perform worst with regard to power and processing time.

Moreover, upon observing the results from the sensitivity, as seen in Fig. 10, we can see that $DroNet\_Tile$ even though it comes off second best in terms of power consumption and inference speed, and it comes last in terms of sensitivity, while $Tiny-YoloV3$ comes second. On the other hand, $EdgeNet$ achieves a 6% higher sensitivity than $Tiny-YoloV3$, which is the largest model of all of them. $EdgeNet-1-3-5$ sensitivity keeps the accuracy close to 96% compared to others due to the fact that single-shot models lose object resolution and features when resizing the image that leads to a decreased accuracy. Overall, by reducing the processed data, having smaller input-sized CNNs and utilizing the tracker, it directly impacts the processing time which leads to reduction of computation power

Table 4 Average Power Consumption measured in Watts

	EdgeNet-1-3-5	DroNet_Tile	DroNet_V3	Tiny-YoloV3
RP3	1.5	1.7	2	2.4
ODROID	3.6	3.8	4	5
CPU	9	10	15	23

Table 5 Average Processing time on different platforms

	EdgeNet-1-3-5	DroNet_Tile	DroNet_V3	Tiny-YoloV3
RP3	0.06	0.09	0.22	0.85
ODROID	0.05	0.1	0.3	1
CPU	0.02	0.03	0.08	0.59

[3] Samsung Exynos-5422 Cortex—A15 2 Ghz and Cortex—A7 Octa-core CPUs with Mali-T628 MP6 GPU.

[4] Quad Core 1.2 GHz Broadcom 64-bit CPU.

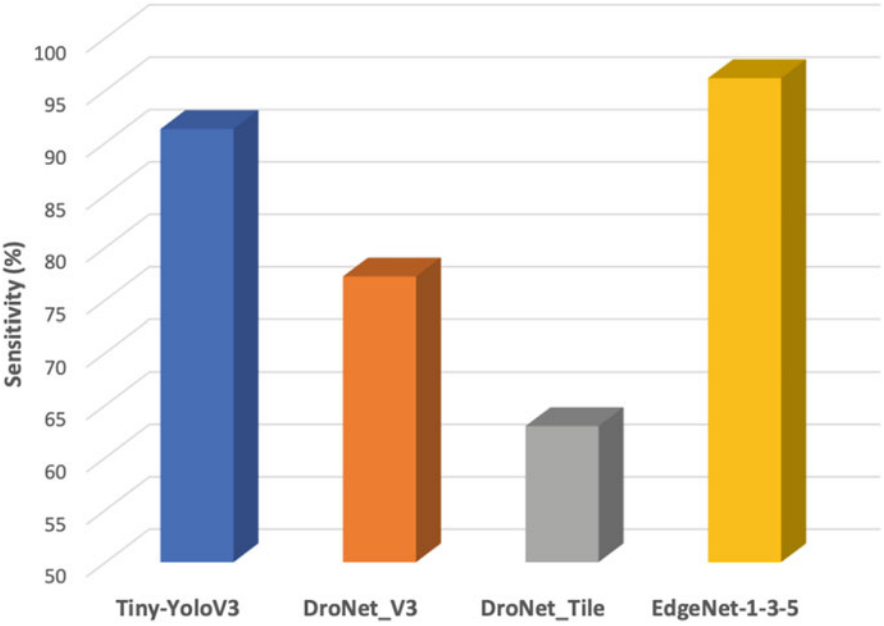


Fig. 10 Sensitivity of Tiny YoloV3, $DroNet\_V3$, $DroNet\_Tile$ and EdgeNet on different platforms. *This figure was taken from our work in [27]*

on all platforms. Therefore, $EdgeNet$ is a framework capable of providing real-time processing pipeline for mobile/edge devices in terms of accuracy, inference speed, and power consumption.

4 Combining Tiling with Quantization

Embedded deployment of neural networks (NNs) are constrained by limited amount of memory, compute, power budget, and real-time latency requirements. As such, relying on small models and data reduction techniques might not be enough. Hence, a range of hardware-driven neural network optimization techniques have been proposed in the literature for efficient deployment. Among those, quantization of neural networks is of special interest since many embedded platforms support integer only arithmetic with INT8 quantization [12]. These include but are not limited to some variants of ARM Cortex-M, RISC-V GAP-8 a system-on-chip, and Google's Edge TPU. Quantization, in general, is a method to map from input values in a large (often continuous) set to output values in a small (often finite) set, e.g., rounding and truncation [7]. In the context of neural networks (NNs), quantization allows network weights and activation functions to be converted from floating point operations to either fixed point or mixed precision operations, hence

reducing model's memory footprint and RAM consumption and improving latency and power consumption. Using TVM quantization library [3, 20] has shown 3.89×, 3.32×, and 5.02× speed-up for ResNet50 [8], VGG-19 [30], and inceptionV3 [31], respectively.

Existing object detectors increase inference efficiency by operating on aggressively downsized, lower resolution images which causes significant information loss and hence performance degradation for small object detection. In addition, quantizing an already resized image further reduces detection accuracy. Therefore, in addition to using quantization for faster inference, an additional mechanism for maintaining high image resolution is desired for highly accurate multi-scale object detection. Our work in Sect. 3 proposes a mechanism to intelligently select and process more relevant portions (tiles) of a high-resolution image, hence maintaining high detection accuracy while still improving the inference times. Thus, combining the aforementioned techniques of small network design, with image data reduction, and network quantization can potentially lead to further improvements.

4.1 Quantization Techniques

The way an NN is quantized can vary in many aspects. A quantization is said to be uniform if all quantization levels are equally spaced and non-uniform otherwise. Similarly, a quantization is said to be symmetric if the clipping range of the signal is centered at zero and asymmetric otherwise. Moreover, if the clipping range of NN activations is pre-computed, the quantization is said to be static. Dynamic quantization is also possible and generally results in higher accuracy but has higher computational overhead because activation clipping range is calculated for each input during real-time inference. Quantization can further be categorized into layerwise, groupwise, and channelwise depending upon what set of parameters is being used to estimate the clipping range of the activations.

Quantization is also classified by "when" it is performed. The most widespread trend is post-training quantization where the neural network is first trained till convergence and quantized only when ready for deployment. Recently, notable accuracy gains have been reported by using quantization-aware training where a network is first trained till convergence, and then some of its layers are quantized and the network is fine-tuned with either mixed precision or integer-only weights. This is intended to make network already aware of it being quantized and fine-tune accordingly, hence named "quantization aware training." For a detailed review of quantization, we refer the reader to [7].

4.2 Approach

In this section, we investigate a case study for object detection using quantization and selective tiling which was discussed in the previous section. In addition, the

model under consideration is DroNetV3, which was also introduced in the previous section. Hence, we study the impact of quantization and tiling techniques on detection accuracy and latency.

As discussed before, aggressive image downsizing leads to poor detection performance in case of smaller objects of interest. In our previous work, [26] shows how to split a larger image into smaller tiles and process only the relevant ones. Since distributing the tiles uniformly across image leads to a drastic increase in the number of sub-images to be processed by the CNN, two statistical techniques are proposed to select and process only relevant tiles while also keeping track of non-active tiles.

The first technique is making use of Intersection over Union (IoU) metric, where each newly detected object's bounding box (bbox) is compared to detected boxes in the previous frames and classified as either new or old object. The position of the objects in previous frames is maintained in a memory buffer. The second technique uses statistical metrics such as the number of objects detected in each tile and prioritizes tiles with a higher number of detected objects as well as those that have not been selected recently to be processed in the subsequent frames. These techniques combined allow an intelligent selection of image regions to be fed into a smaller and faster object detector thus keeping higher resolution intact leading to higher accuracy and real-time performance due to selective processing.

A post-training quantization of 8 bits is applied for both the input and the selected convolutional neural network detector. With the use of quantization, multiply–add operations are transformed into lower precision operations which lead to large computational gains and higher performance. This implementation is based on the Darknet framework and CUDA-based neural network framework. The combined tiling and quantization approach can then be analyzed with respect to resulting accuracy and processing time for the application of people detection from UAV.

4.3 Experimental Results

In case of YOLOV3 and Tiny-Yolov3, speed-ups up to $1.4 - 1.7\times$ are observed. However, with DroNet and DroNetV3, i.e., relatively smaller networks, the speed-up is limited to $1 - 1.4\times$. This is due to significant performance overhead caused by the additional processing time for input image quantization as compared to network inference time. The rest of the experiments are conducted using DroNetV3 which achieved highest speed-up using input resolution of 352×352.

Next, we evaluate combinations of quantization with different tiling schemes and report accuracy, IoU, and average processing time (APT) metrics. All networks are trained on a UAV-based people detection dataset consisting of 1500 images and a total of 60,000 people.

Table 6 shows the impact of resizing, processing a single, all or only selected tiles [26], and applying quantization to all base cases. Resizing the input causes 20% accuracy drop and quantization on top drops another 8%. This result is expected

Table 6 Results of quantization and tiling with resizing, single tile selection, all tiles selection, and intelligent tile selection

	DroNetV3	DroNetV3 + Resizing	DroNetV3 + Single Tile	DroNetV3 + All Tiles	DroNetV3 + Selective Tiling
Baseline Models					
Accuracy	98.929	79.643	92.143	99.464	96.071
APT	0.432	0.089	0.087	0.680	0.132
IoU	0.643	0.415	0.496	**0.650**	0.563
Quantized models					
Accuracy	**100**	71.429	94.643	98.750	98.571
APT	0.315	0.071	**0.068**	0.574	0.098
IoU	0.648	0.312	0.494	0.619	0.562

due to information loss of input image and model weights, respectively. However, resizing speeds up inference significantly, i.e., an APT of 0.432 s to 0.089 s. Since tiling maintains high resolution, all techniques based on tiling improve accuracy as compared to DroNetV3 with resizing. Even when models are quantized, accuracy increases from 71% to 94% – 98%, i.e., an increase of 23% to 27%. Since the selective tile processing mechanism is processing $2-3$ per frame, the processing time is higher than either resizing or single tiling. However, quantizing this network improves latency from 0.132 s to 0.098 s without accuracy degradation. A similar trend has been observed for IoU metric, i.e., avoiding resizing the image in general leads to higher IoU. Quantization reduces the IoU from 0.415 to 0.312 for DroNet with resizing and its quantized version. However, combining quantization and tiling does not degrade IoU. Overall, using a combination of quantization and tiling allows processing higher resolution images, hence improving latency as compared to original full precision detector and accuracy as compared to using resizing.

5 Conclusion

Deep learning and computer vision are increasingly being utilized in edge applications to provide real-time intelligence. This chapter has demonstrated various techniques and approaches to apply at various stages of visual processing in order to make algorithms more suitable for the unique demands of aerial image understanding with UAVs and embedded application domains. The exploration of small neural network design considerations and architectures can be effective in deploying such models on resource-constrained devices. Coupled with search reduction strategies and hardware-directed optimizations, the performance of such small models can be further improved leading to real-time power efficient solutions. As a future work, we aim to further improve the accuracy performance trade-off by investigating adaptive computation schemes through networks with multiple

learnable exit layers, as well as dynamically changing the computation scheme based on context and state variables.

Acknowledgments The project is co-financed by the European Regional Development Fund and the Republic of Cyprus through the Cyprus Research & Innovation Foundation ("RESTART 2016-2020" Program) (Grant No. INTEGRATED/0918/0056) (RONDA). This work was also supported by the European Union's Horizon 2020 research and innovation program under grant agreement No. 739551 (KIOS CoE) and from the Government of the Republic of Cyprus through the Directorate General for European Programs, Coordination, and Development.

Christos Kyrkou gratefully acknowledge the support of NVIDIA Corporation with the donation of the RTX A6000 GPU.

References

1. Allison, R., Johnston, J., Craig, G., Jennings, S.: Airborne optical and thermal remote sensing for wildfire detection and monitoring. Sensors **16**(8), 1310 (2016)
2. Chen, L.C., Papandreou, G., Schroff, F., Adam, H.: Rethinking atrous convolution for semantic image segmentation. arXiv preprint arXiv:1706.05587 (2017)
3. Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E.Q., Shen, H., Cowan, M., Wang, L., Hu, Y., Ceze, L., Guestrin, C., Krishnamurthy, A.: TVM: An automated end-to-end optimizing compiler for deep learning. In: OSDI (2018)
4. Chen, X., Xiang, S., Liu, C.-L., Pan, C.-H.: Vehicle detection in satellite images by parallel deep convolutional neural networks. In: 2013 2nd IAPR Asian conference on pattern recognition, pp. 181–185. IEEE, New York (2013)
5. Chollet, F.: Xception: Deep learning with depthwise separable convolutions. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1800–1807 (2017)
6. Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., Fei-Fei, L.: ImageNet: A large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition, pp. 248–255 (2009)
7. Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M.W., Keutzer, K.: A survey of quantization methods for efficient neural network inference. CoRR, abs/2103.13630 (2021)
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778 (2016)
9. Howard, A., Sandler, M., Chu, G., Chen, L.C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q.V., Adam, H.: Searching for mobilenetv3. In: The IEEE International Conference on Computer Vision (ICCV) (2019)
10. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: MobileNets: Efficient convolutional neural networks for mobile vision applications. CoRR, abs/1704.04861 (2017)
11. Iandola, F.N., Moskewicz, M.W., Ashraf, K., Han, S., Dally, W.J., Keutzer, K.: SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1 mb model size. CoRR, abs/1602.07360 (2016)
12. Ignatov, A., Malivenko, G., Timofte, R.: Fast and accurate quantized camera scene detection on smartphones, mobile ai 2021 challenge: Report. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, pp. 2558–2568 (2021)
13. Kim, E.J., Park, H.C., Ham, S.W., Kho, S.Y., Kim, D.K.: Extracting vehicle trajectories using unmanned aerial vehicles in congested traffic conditions. J. Adv. Transp. **2019**, 1–16 (2019)
14. Kyrkou, C.: C^3Net: End-to-end deep learning for efficient real-time visual active camera control. J. Real-Time Image Proc. **18**(4), 1421–1433 (2021)

15. Kyrkou, C., Christoforou, E.G., Timotheou, S., Theocharides, T., Panayiotou, C., Polycarpou, M.: Optimizing the detection performance of smart camera networks through a probabilistic image-based model. IEEE Trans. Circuits Syst. Video Technol. **28**(5), 1197–1211 (2018)
16. Kyrkou, C., Plastiras, G., Theocharides, T., Venieris, S.I., Bouganis, C.S.: DroNet: Efficient convolutional neural network detector for real-time UAV applications. In: 2018 Design, Automation Test in Europe Conference Exhibition (DATE), pp. 967–972 (2018)
17. Kyrkou, C., Theocharides, T.: Deep-learning-based aerial image classification for emergency response applications using unmanned aerial vehicles. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pp. 517–525 (2019)
18. Kyrkou, C., Theocharides, T.: EmergencyNet: Efficient aerial image classification for drone-based emergency monitoring using atrous convolutional feature fusion. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing **13**, 1687–1699 (2020)
19. Kyrkou, C., Timotheou, S., Kolios, P., Theocharides, T., Panayiotou, C.G.: Optimized vision-directed deployment of UAVs for rapid traffic monitoring. In: 2018 IEEE International Conference on Consumer Electronics (ICCE), pp. 1–6 (2018)
20. Lin, W.: Automating optimization of quantized deep learning models on CUDA (2019)
21. Lucas, B.D., Kanade, T., et al.: An iterative image registration technique with an application to stereo vision. Vancouver (1981)
22. Ma, N., Zhang, X., Zheng, H.T., Sun, J.: ShuffleNet V2: Practical guidelines for efficient CNN architecture design. In: Computer Vision—ECCV 2018, pp. 122–138. Springer International Publishing, Cham (2018)
23. Makrigiorgis, R., Hadjittoouli, N., Kyrkou, C., Theocharides, T.: AirCamRTM: Enhancing vehicle detection for efficient aerial camera-based road traffic monitoring. In: 2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), pp. 3431–3440 (2022)
24. Murugan, D., Garg, A., Singh, D.: Development of an adaptive approach for precision agriculture monitoring with drone and satellite data. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing **10**(12), 5322–5328 (2017)
25. Petrides, P., Kyrkou, C., Kolios, P., Theocharides, T., Panayiotou, C.: Towards a holistic performance evaluation framework for drone-based object detection. In: 2017 International Conference on Unmanned Aircraft Systems (ICUAS), pp. 1785–1793 (2017)
26. Plastiras, G., Kyrkou, C., Theocharides, T.: Efficient convnet-based object detection for unmanned aerial vehicles by selective tile processing. In: Proceedings of the 12th International Conference on Distributed Smart Cameras (ICDSC '18), New York, NY, USA (2018). Association for Computing Machinery, New York
27. Plastiras, G., Kyrkou, C., Theocharides, T.: EdgeNet: Balancing accuracy and performance for edge-based convolutional neural network object detectors. In: Proceedings of the 13th International Conference on Distributed Smart Cameras, pp. 1–6 (2019)
28. Redmon, J., Farhadi, A.: Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767 (2018)
29. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 4510–4520 (2018)
30. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: International Conference on Learning Representations (2015)
31. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2818–2826 (2016)
32. Tan, M., Le, Q.V.: EfficientNet: Rethinking model scaling for convolutional neural networks. In: Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9–15 June 2019, Long Beach, California, USA. Proceedings of Machine Learning Research, vol. 97, pp. 6105–6114. PMLR, New York (2019)
33. Zhao, Y., Ma, J., Li, X., Zhang, J.: Saliency detection and deep learning-based wildfire identification in UAV imagery. Sensors **18**(3), 712 (2018)

Edge-Centric Optimization of Multi-modal ML-Driven eHealth Applications

Anil Kanduri, Sina Shahhosseini, Emad Kasaeyan Naeini, Hamidreza Alikhani, Pasi Liljeberg, Nikil Dutt, and Amir M. Rahmani

1 Introduction

Smart eHealth applications deliver critical digital healthcare services such as disease diagnostics, clinical decision support, forecasting health status, pro-active and preventive healthcare decisions, alerts for emergency intervention, etc. [20]. eHealth applications improve the reach and quality of healthcare services, timeliness, and accuracy of clinicians decisions and reduce the burden on healthcare professionals and overall medical expenditure [72]. Smart eHealth systems integrate remote sensing, continuous monitoring, wireless transmission, data analytics, and machine learning to deliver intelligent patient-centric digital healthcare and well-being services [36]. eHealth applications are particularly effective for managing chronic patients through continuous monitoring, extracting clinically relevant data with minimal intrusion [18].

A. Kanduri (✉) · P. Liljeberg
Department of Computing, University of Turku, Turku, Finland
e-mail: spakan@utu.fi; pasi.liljeberg@utu.fi

S. Shahhosseini · E. K. Naeini · H. Alikhani · N. Dutt
Department of Computer Science, University of California, Irvine, CA, USA
e-mail: sshahos@uci.edu; ekasaeya@uci.edu; hamidra@uci.edu; dutt@uci.edu

A. M. Rahmani
Department of Computer Science, University of California, Irvine, Irvine, CA, USA

School of Nursing, University of California, Irvine, CA, USA
e-mail: a.rahmani@uci.edu

S. Pasricha, M. Shafique (eds.), *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing*, https://doi.org/10.1007/978-3-031-40677-5_5

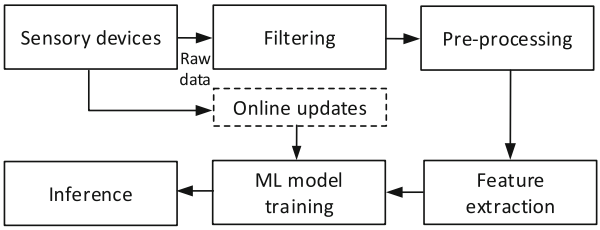


Fig. 1 ML-driven eHealth application pipeline

1.1 ML in Smart eHealth Applications

eHealth systems continuously monitor patients using wearable sensors for acquiring physiological parameters [17]. In addition to the bio-signals, eHealth applications also track behavioral and environmental parameters to contextualize the patients' current situation [39]. Thus, smart eHealth applications generate huge volumes of heterogeneous input data, combining multiple streams of inputs from physiological, behavioral, and environmental parameters [20]. Analyzing such continuous stream of heterogeneous multi-modal raw data for predicting potential threats, accurate clinical decisions, and diagnostics requires broader support from the AI domain [25]. Smart eHealth systems are increasingly using ML algorithms for analyzing multi-modal input sensory data, to provide intelligent digital healthcare and well-being services [36]. State-of-the-art eHealth applications have applied different ML algorithms for analyzing input data, and predicting results on diagnostics, potential and health status [16]. ML-driven eHealth systems work in a pipeline of data acquisition, filtering and pre-processing, data analysis, training, inference for predictive results, followed by notification to the clients [20]. Figure 1 shows the workflow of typical ML-driven eHealth applications, where raw data acquired by sensory devices is filtered and preprocessed to remove noisy components, motion artifacts, and anomalies. This input data is then used for extracting relevant features and training suitable ML models. Predictive results are achieved by inferencing the trained ML model, while the trained model is periodically updated with evolving input data.

1.2 Collaborative Edge Computing for Smart eHealth Applications

eHealth applications rely on traditional cloud infrastructure for training, storage, and updating of ML models and inference tasks. However, rapidly increasing volumes of sensory input data and uncertain network conditions imposes limitations on the efficacy of running eHealth applications on the cloud layer. Edge computing

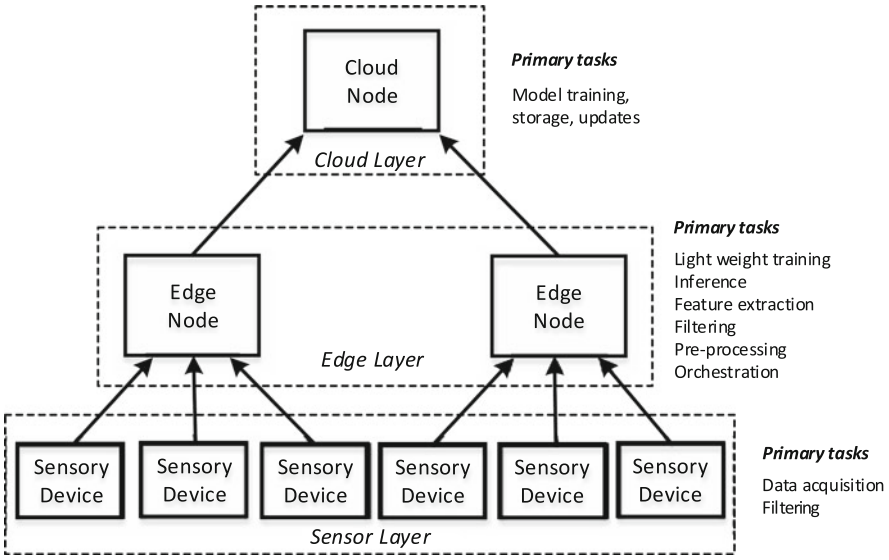


Fig. 2 Sensor–edge–cloud architecture

paradigm brings computational intelligence closer to the sources of input data, minimizing the reliance of smart eHealth applications on cloud infrastructure [74]. Edge computing architectures have been widely adopted for deploying ML-driven smart eHealth applications, simultaneously handling input data, compute intensity, and network constraints [55]. Figure 2 shows an overview of the hierarchical multi-layered sensor–edge–cloud architecture [5]. The *sensor layer* comprises sensory devices such as wearable sensors, smart biosensors, and sensors deployed on mobile and IoT devices. The *sensor layer* primarily acquires raw data from different devices, performs lightweight tasks such as filtering, and transmits relevant inputs to the resourceful layers in the hierarchy. The *edge layer* receives input data from the sensory devices and executes intensive tasks such as data preprocessing, feature extraction, lightweight ML model training, and inference tasks. More importantly, the *edge layer* also handles orchestration functionalities such as application-level and system-level monitoring, application partitioning, compute placement, and resource allocation. The *cloud layer* handles heavy computational tasks such as ML model training, updating, and storage and notifications to edge nodes on model updates.

1.2.1 Example Scenario

We demonstrate the pipeline of a quintessential edge-centric ML-driven eHealth application through the example of arrhythmia detection application [20]. Figure 3 shows the data and control flows of the arrhythmia detection application [20]

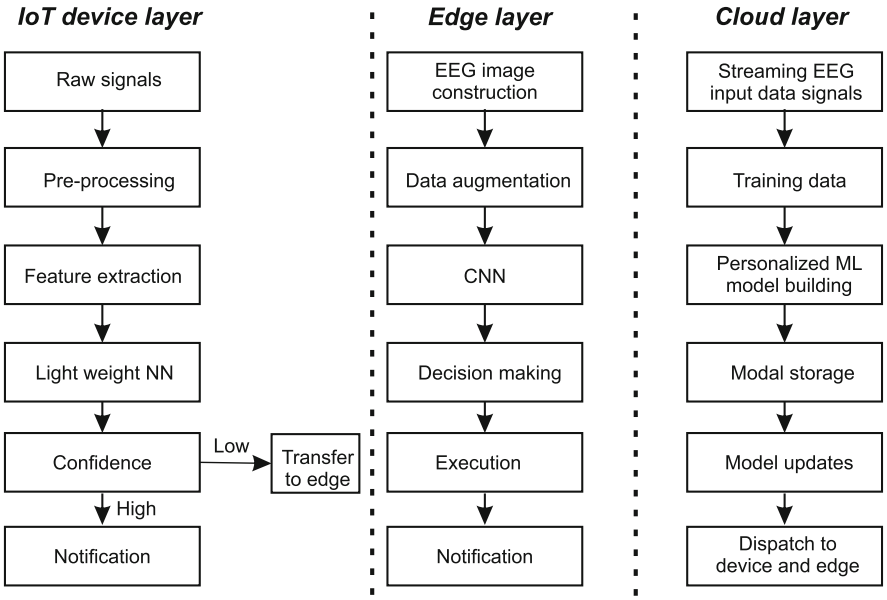


Fig. 3 Pipeline of arrhythmia detection [20]

across the device–edge–cloud layers. Initially, raw electroencephalogram (EEG) signals are acquired by wearable sensory devices. The raw signals are preprocessed for noise removal to extract relevant features from the raw data for training an ML model. Considering the limited compute resources of the IoT device layer, a lightweight neural network (NN) model is employed to predict/detect arrhythmia. The predictions are notified to the client if the NN model has a higher confidence on prediction accuracy, while forwarding the input data to edge layer when the model confidence is lower. In this application, the edge layer uses the reconstructed EEG images for data aggregation. With relatively higher compute resources, the edge layer consists of a moderately intensive convolutional neural network (CNN) model to train on input data. The CNN model is inferred in the execution phase for arrhythmia detection/prediction, and the client is notified with the result. The cloud layer collects streaming inputs from the device and edge layers to train appropriate ML model, store the model, and update the model periodically with evolving input data. The cloud layer transmits the updated model parameters to the edge and device layers for running local inference tasks. Furthermore, the cloud layer performs advanced data analytics to generate personalized decisions for each client. It should be noted that the compute capabilities and thus tasks vary at each of the device–edge–cloud layers. Optimizing ML-driven eHealth systems requires such understanding on input data flows, computational requirements, and accuracy and performance of ML models.

1.3 Summary

Implementing ML-driven smart eHealth applications presents different challenges on sensory data acquisition, understanding application-level requirements, handling compute intensities of ML models, and energy and network constraints. At the same time, deploying such applications on multi-layered sensor–edge–cloud platforms exposes opportunities for selective processing through input data quality awareness, choice of compute placement among edge–cloud nodes exploring energy–performance trade-offs, and understanding algorithmic nature of applications to explore accuracy–performance–energy trade-offs. Collaborative sensor–edge–cloud platforms enable layer-wise partitioning of smart eHealth application pipeline, to synergistically improve the quality of the services. In the subsequent sections, we present different edge-centric optimizations for ML-driven smart eHealth applications on collaborative sensor–edge–cloud platforms.

1.3.1 Organization

Section 2 presents an exemplar case study of pain assessment application, describing a sensor–edge–cloud framework integrating different edge-centric optimizations. This case study is then used in the following chapters to demonstrate some of the optimization techniques. Section 3 presents techniques for improving performance metrics of edge-centric ML workloads through efficient compute placement and exploration of accuracy–latency trade-offs. Section 4 presents techniques for improving resilience of ML-driven eHealth applications, through sense–compute co-optimization. Section 5 concludes with key insights and open research directions.

2 Exemplar Case Study of Edge-ML-Driven Pain Assessment Application

In this section, we present an exemplar case study of pain assessment application, describing application characteristics, nature of input data, and challenges in typical edge-centric ML-driven smart eHealth application. We also present a modular framework, iHurt, for deploying ML-driven eHealth applications (pain assessment in this case study) on collaborative sensor–edge–cloud platforms. The iHurt framework serves as a generic platform for prototyping smart eHealth applications for processing sensory data using ML models. Furthermore, iHurt platform provides a testbed for evaluating the edge orchestration, compute placement, RL agent-based offloading, and sense–compute co-optimization techniques presented in Sects. 3 and 4.

2.1 Pain Assessment

Pain is a complex phenomenon, associated with several illnesses [70]. Pain is defined as “an unpleasant sensory and emotional experience associated with actual or potential tissue damage, or described in terms of such damage” [47]. The pain assessment “gold standard" relies on a patient’s self-report of their pain intensity on a scale of 0–10, where 0 refers to no pain and 10 represents the most severe pain. Pain assessment is done through tools such as Numerical Rating Scale (NRS), Visual Analogue Scale (VAS), and Verbal Rating Scale (VRS). There is a high demand for objective tools to assess patients’ pain in the clinical context. Tools are needed especially when the patient’s own opinion is difficult to obtain. Assessment of pain is particularly difficult when the ability of a patient to communicate is limited (e.g., during critical illness, in infants and preverbal toddlers, or in patients under sedation or anesthesia, with intellectual disabilities, and at the end of life) [10]. Inadequately treated pain has major physiological, psychological, economic, and social ramifications for patients, their families, and society [4]. Undertreatment of pain could result in many adverse effects and other complications and may evolve into chronic pain syndromes. It could also cause delayed discharge or prolonged recovery, which may incur higher health care costs and more patient suffering [66]. Overtreatment of pain, on the other hand, may result in unintended adverse consequences such as acute respiratory complications or in long-term complications such as opioid addiction. These issues are particularly pronounced for non-communicative patients who are unable to articulate their experience of pain [8].

We demonstrate an abstract overview of the pain assessment application [31] in Fig. 4. The pain assessment application is implemented as a pipeline of sensing, data processing, model training for predictive results, and inference for pain-level classification. In the context of this case study, we use input data from EMG, ECG, PPG, and EDA sensors for estimation of pain. Sensory data is preprocessed for filtering qualitative inputs, followed by feature extraction. Relevant learning models

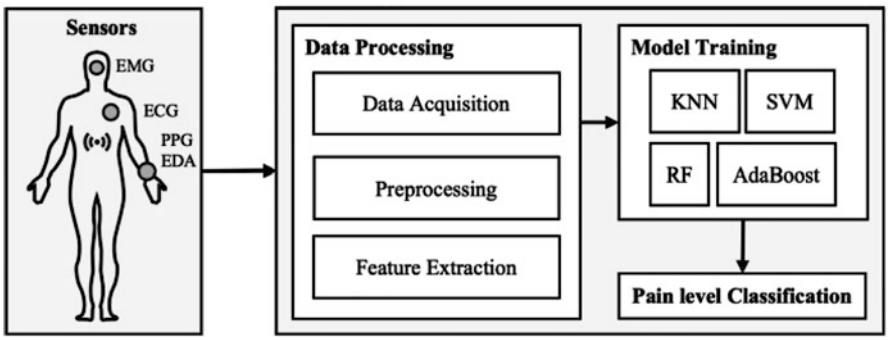


Fig. 4 Overview of pain assessment application

are trained with the multi-modal input data sets for classifying pain level. Different phases of the pain assessment application are detailed in the following.

2.2 Sensory Data Acquisition

Objective pain assessment application collects raw data through continuous monitoring in both clinical and everyday settings [20]. Raw data collected from the sensory data acquisition phase is used for training the ML models for accurate prediction of pain level. In this subsection, we describe the nature and characteristics of sensory input data used in the pain assessment application.

2.2.1 Types of Signals

There are various types of signals that influence the accuracy of monitoring and assessing the affective states in pain assessment. These indicators are extracted through different forms of behavioral, physiological, and contextual/environmental sensor modalities via facial expression, speech, full-body motion, text, and physiological signals. Both behavioral and physiological manifestations of pain can be measured objectively. Behavioral pain indicators include facial expressions, body movements such as rubbing, restlessness, and head movements, and paralinguistic vocalizations such as crying and moaning. Physiological pain indicators are acquired from brain, cardiovascular, and electrodermal activities. Monitoring of these physiological, behavioral, and contextual sensor inputs can also be used in other prominent eHealth applications including emotion recognition and stress monitoring. For instance, stress activates the autonomic nervous system (ANS) which can be detected through monitoring the changes in physiological signals including cardiovascular activity and electrodermal activity, respiration rate, and blood pressure [23]. Furthermore, physiological signals of cardiac function, temperature, muscle electrical activity, respiration, skin conductance, and brain electrical activity can be used to detect human emotions. The multitude of physiological, behavioral, and contextual signals fused together can provide valuable insights for training ML models, specifically in the domain of smart affective computing applications.

2.2.2 Commonly Used Sensors

Recording physiological signals requires people to connect with biosensors. There are contact-based sensors (such as adhesive electrodes and wristbands) or contact-free sensors (such as cameras and microphones) to gather information from patients and analyze them. Widely used contact sensors in eHealth applications record electroencephalogram (EEG, electrical activity of the brain), electrocardiogram

(ECG, heart activity (heart rate (HR) and heart rate variability (HRV))), electrodermal activity (EDA often measured using skin conductance level (SCL), and sometimes the old term "galvanic skin response" (GSR)), surface electromyogram (sEMG, muscle activity), photoplethysmogram (PPG, blood perfusion of the skin for pulse and other measures, also called blood volume pulse or BVP), respiration (RSP), or acceleration (ACC, movement). For pain monitoring, the unidimensional assessment tools have been questioned and debated for their oversimplification and limited applicability in non-communicative patients, since they require interactive communication between patient and caregiver [27]. As a result, physiological sources of data comprising of heart rate (HR), heart rate variability (HRV), SpO_2, skin temperature, electrodermal activity (EDA), and facial expression and frontal muscle activity using computer vision or facial electromyography (EMG) and electroencephalogram (EEG) are prioritized for pain assessment. Accurate calculation of HRV parameters depends on detecting the position of peaks within ECG or PPG signals. Root Mean Square of Successive Differences (RMSSD) is an HRV parameter that is correlated to the short-term variation in the PPG signal. Figure 5 shows a-minute filtered PPG signal illustrating an example, in which less than 5 s of the PPG signal (highlighted in red) are distorted due to hand movements. Such a minor window of corrupted input data in the signal could still affect the

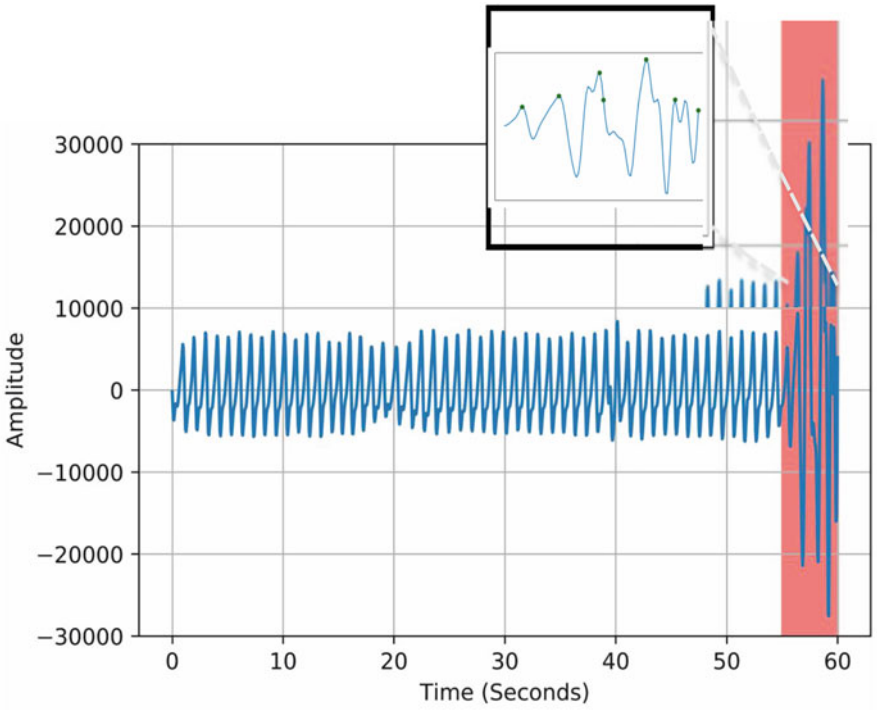


Fig. 5 One-minute windows of filtered PPG signals carrying noise

Table 1 Summary of accuracy results (BL: baseline, PL: pain level, and MM: multi-modal)

	EDA	ECG	RR	MM
BL vs. PL1	63.36	72.04	71.79	77.13
BL vs. PL2	79.24	81.13	82.14	85.64
BL vs. PL3	69.59	69.8	76.64	86.9
BL vs. PL4	63.7	63.41	66.67	74.73
Mean	68.97	71.6	74.31	81.1

eventual accuracy significantly. For instance, in Fig. 5, few peaks are detected incorrectly within the noisy signal part, and thus the RMSSD is not reliable anymore during this window of data. The pain assessment application uses ML models to detect such abnormalities and enable accurately predictions.

2.2.3 Multi-modal Inputs

Objective pain assessment relies on input data from multiple modalities and combinations of physiological, behavioral, and contextual parameters. All these modalities differ in terms of data, noise characteristics, comfort and ease of use, privacy concerns, and energy consumption. Using a single modality versus a combination of multiple modalities effects the computational workloads of the ML models and eventual prediction accuracy. This is demonstrated from a pain monitoring case study on five levels of pain data collection [2, 11, 32, 34]. The accuracy of using each individual input modality of sensory data (EDA, ECG, RR) and multi-modal (MM) inputs for binary classification between no pain/baseline and various pain levels is shown in Table 1. It should be noted that different sensor modalities result in a range of prediction accuracies across different levels of pain. Predictions based on multi-modal input data set have the highest accuracy among each of these cases.

2.3 ML-Driven Objective Pain Assessment

It is imperative to design and develop an objective monitoring tool to improve the well-being and care processes of patients with a more accurate assessment and more timely treatment. While this raises significant technical challenges, requiring a combination of sensing, signal processing, and machine learning skill sets, it also has a tremendous potential to pave the way for next-generation human-modeling methods. Machine learning and deep learning techniques have become immensely popular for classification tasks, as well as for other recognition and pattern matching tasks. ML models can be used for accurate objective pain assessment, using input data from different modalities. The combination of vast amount of multi-modal input data, increased computing power, and more intelligent methods enables fast and automated production of machine learning algorithms able to analyze complex data with accurate results.

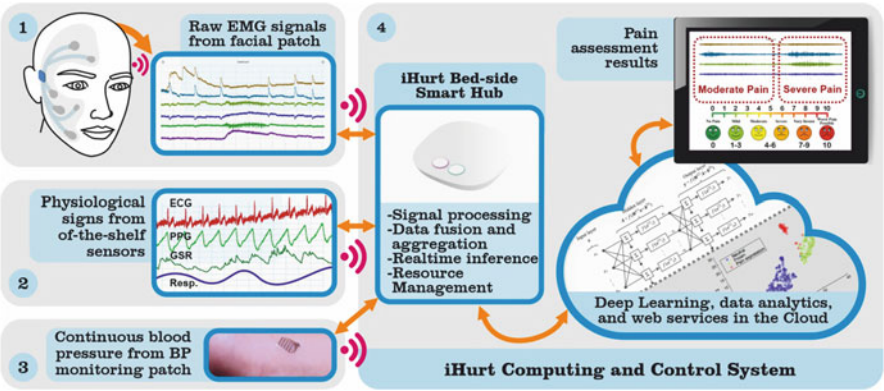


Fig. 6 The objective pain assessment technology

2.3.1 iHurt Platform

Figure 6 presents a complete objective pain assessment technology developed jointly by researchers from University of California, Irvine (UCI) and University of Turku (UTU), Finland. This is an end-to-end system for multi-modal data acquisition, data processing, and analyzing at the gateway. The first step in building a multi-modal system is to process the raw signals collected during trials. While this varies according to the deployed sensor data, a typical preprocessing pipeline consists of the following: filtering (data cleansing, noise reduction, and artifact removal), segmentation (partitioning into time intervals), and normalization (w.r.t. to a baseline signal). These steps are followed by feature extraction to obtain features within various domains. Finally, the processed data can be used to build prediction models using machine learning. Different types of ML models can be used to train over multi-modal input data sets. The choice of the learning model determines the integration and fusion of multi-modal data at feature, decision, or intermediate levels (early, late, and hybrid fusion). Most approaches classify pain using support vector machines (SVMs) [24, 77], random forests (RFs) [30, 76], and nearest neighbors (NNs) [1]. Other widely used models include ADABoost and XGBoost [56], which are ensemble methods to reduce bias and variance in predictive data analysis. The ML models can be computationally fine-tuned differently to pursue various objectives. Sense-making knobs such as early exit, model selection, and input modality selection presented in Sect. 4 can be explored in this context. Figure 7 shows five different classification methods using respiratory signals including ADABoost, XGBoost, RF, SVM, and K-NN classifiers in comparison with a state-of-the-art method, RESP [11]. This case study down sampled pain levels into three levels of pain (pain levels 1–3), besides the baseline of no pain. Note that they achieved higher accuracy compared with the state of the art while using only 88% less features. We can maintain accuracy in presence of noise, or using useful features from the reliable modalities, while also meeting the requirements set on the

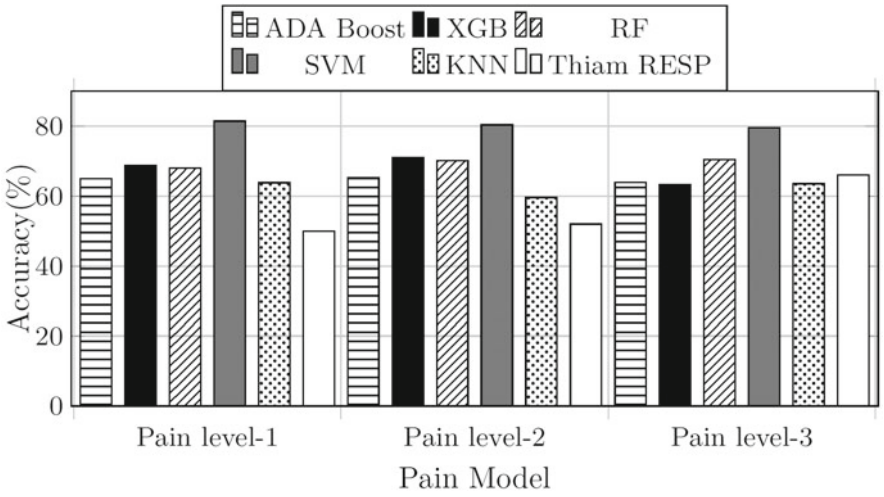


Fig. 7 Validation accuracy of classifiers on top-8 features for different pain levels compared with the baseline

computational performance. The orchestration functionalities presented in Sect. 3 can guide these decisions on exploring accuracy–performance trade-offs.

2.3.2 Other Exemplar eHealth Applications

Emotion recognition and stress monitoring are two other widely used eHealth applications that rely on input data similar to that of the pain assessment application. Emotion recognition uses verbal inputs and cues such as tone of voice, facial expressions, postures, gestures and also through physiological signals [38]. Stress monitoring application detects the existence of stress in each period of time using physiological signals [26]. The Electrodermal Activity (EDA) or Galvanic Skin Response (GSR) is one of the physiological signals generated by the human body, which can be used to detect stimuli in individuals. Fall detection is another exemplar application that uses 3D accelerometer data to detect falls by using classification models [15]. The input data quality assessment, using multi-modal ML models, configuration of ML models, and edge orchestration techniques used for pain assessment can be analogously applied to other similar applications of emotion recognition, stress monitoring, and fall detection.

3 Edge-Centric Optimization of ML-Based eHealth Workloads

Machine learning (ML) is advancing real-time and interactive user services in healthcare domain [57]. ML applications are primarily deployed on cloud infrastructure to meet the compute intensity and storage requirements of ML algorithms and address the resource constraints of user-end wearable sensory devices [7]. However, unpredictable network constraints including variable signal strength and availability of the network affect real-time delivery of cloud services [37]. The edge computing paradigm allows deployment of ML applications closer to the user-end devices, minimizing the latency of service delivery, reducing the total network load, and alleviating privacy concerns.

System Perspective Collaborative sensor–edge–cloud architecture presents multiple execution choices for workload partitioning and compute placement including execution on a single sensor, edge, cloud nodes, and any possible combinations of these devices. Considering the variable accuracy nature of ML algorithms, these execution choices expose a wide range of energy–performance–accuracy trade-off space. Choosing an optimal execution option under varying system dynamics, available energy budget of devices, network constraints, and error resilience of ML workloads is a complex run-time challenge.

Application Perspective Depending on the ML model employed, different applications feature varying compute, data, and communication intensities. At an application level, there is diversity in terms of sensitivities to latency, throughput, infrastructure availability, and accuracy. Furthermore, different application execution choices result in different energy consumption patterns [33]. Considering these application-level variations, the choice of execution of an application is a subject of multiple factors varying at run-time [22].

Edge Orchestration Edge orchestration techniques handle workload partitioning, distribution, and scheduling of ML workloads, considering both applications' and systems' perspectives. Understanding the varying compute intensities of applications, latency and throughput requirements, user-interaction and responsiveness expectations, quality of interconnection network including signal strength, availability, load balancing, compute, and storage capacities of underlying hardware elements put together makes application orchestration a stochastic process [60, 63]. To maximize the efficiency of edge-enabled health care services, run-time solutions that holistically consider both requirements and opportunities vertically across the user, device, application, network, edge node, and platform layers are necessary [59]. In this section, we present state-of-the-art edge orchestration techniques for efficient compute placement with rule-based heuristics and optimized orchestration through reinforcement learning.

3.1 Dynamics of Compute Placement

Computation offloading techniques transfer the execution of an application, or a task within an application, to a resourceful device for improving performance [45]. Compute placement determines the choices on partitioning an application and selection of the external resourceful edge/cloud nodes onto which the partitioned task is to be offloaded [7]. Some of the existing compute placement and offloading strategies do not consider the diversity in applications' compute and communication requirements, eventual performance gain with offloading and compute placement choices, and potential latency penalties incurred with those choices. For optimal performance gains, compute placement techniques have to consider the dynamically varying application and network characteristics, energy budgets, and accuracy–performance trade-offs [19, 61, 69].

We explore the intricacies of running smart eHealth applications on multi-layer sensor–edge–cloud platforms to demonstrate the dynamics of compute placement. We choose stress monitoring [48], fall detection [15], and pain assessment [26] as representative workloads from ML-driven digital healthcare systems. The stress monitoring application uses predictive models to extract statistical features from physiological parameters of Electrodermal Activity (EDA) and Galvanic Skin Response (GSR) and predict stress levels [3]. The fall detection application uses de-noising, feature extraction, and decision tree training for classification of fall and no-fall events [59]. The pain assessment application uses preprocessing, feature extraction, and SVM classification for determining level of pain [41]. In summary, each application typically includes the pipeline of data preprocessing, feature extraction, and classification ML tasks. We execute these workloads on real hardware testbed that emulates a baseline sensor–edge–cloud platform. We consider an edge platform with configurable onboard sensors for sensing at variable sampling rates and connectivity to cloud infrastructure. We consider three compute placement policies with the execution choices of running the ML workloads: (i) fully on the edge device, (ii) fully on the cloud node, and (iii) partially on the edge and partially on the cloud. For evaluation, we define *latency* metric as the time taken to respond to a request including the communication and execution costs.

Figure 8 shows the latency of stress monitoring, fall detection, and pain monitoring applications with different compute placement choices, over different network bandwidths, and sampling rates of input sensory devices. The compute placement choices Local, Cloud, and Partial represent execution of applications on edge node, cloud node, and collaborative edge–cloud nodes. We used three levels of available bandwidths: low, medium, and high. The available bandwidth influences the latency of execution, specifically incurred in transmitting data from edge to cloud nodes. We used two levels of sampling rates: high and low, for input data sensory devices. Sampling rate determines the total volume of data being transmitted from edge to cloud nodes, effecting the latency.

The evaluation shows that network variation in terms of available bandwidth substantially effects the latency and choice of compute placement. For example,

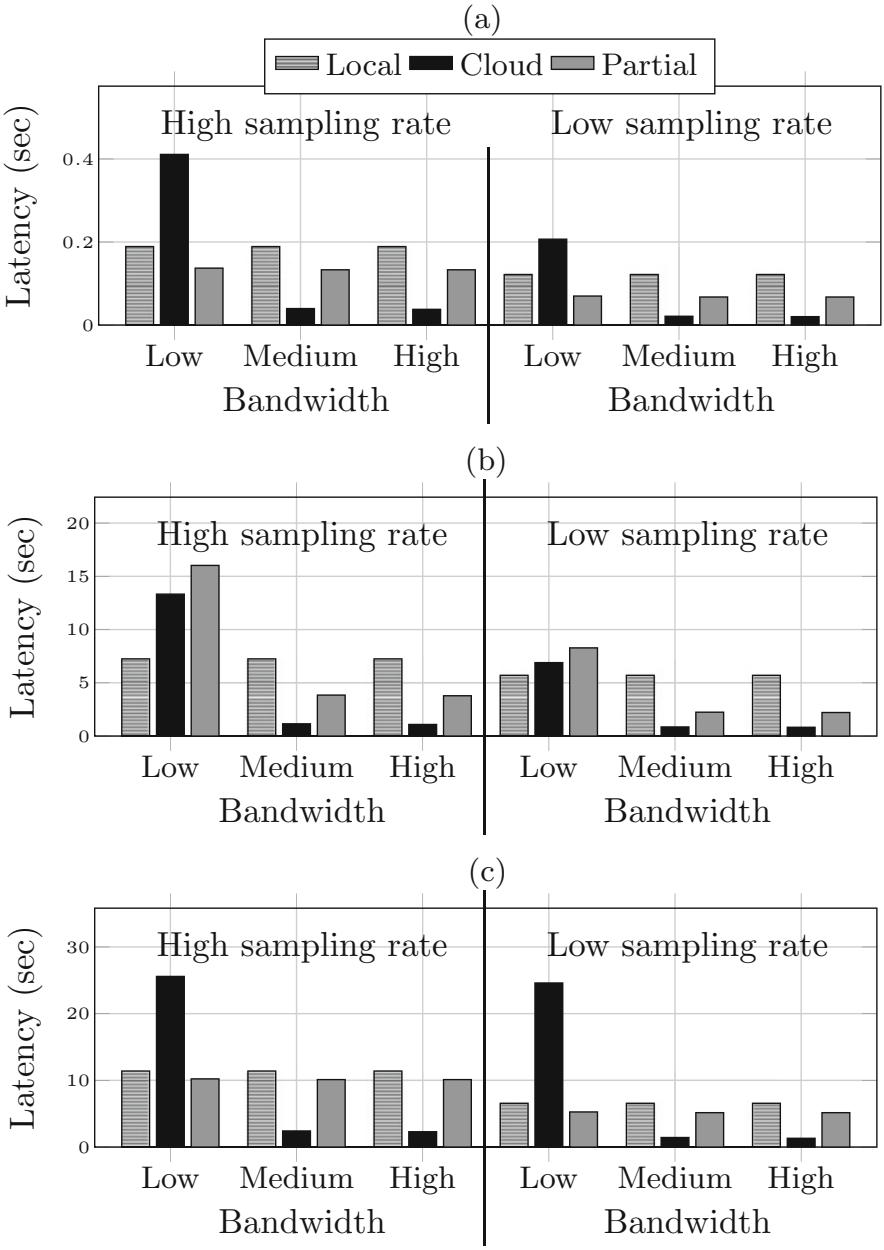


Fig. 8 Latency with different compute placement strategies for eHealth applications under different network bandwidth (low, medium, and high) and sampling rates (high and low). (**a**) Stress monitoring, (**b**) fall detection, and (**c**) pain monitoring

consider the scenario shown in Fig. 8a, under high sampling rate and high bandwidth. In this case, compute placement on the cloud is the optimal choice in comparison with the edge and partial choices. Since there is sufficient network bandwidth, the penalty of transmitting data from edge to cloud is minimal, and the performance gain of executing the application on the cloud is also significant. In contrast, for the same scenario with low bandwidth (LBW), compute placement on the cloud has the highest latency, owing to the higher penalty of transmitting data from edge to cloud under low bandwidth. In this case, the partial (edge–cloud) execution has a better latency.

In addition to the bandwidth availability, the application's nature can substantially influence the choice of compute placement. For instance, consider the scenario in Fig. 8b, under high sampling rate and low bandwidth. The fall detection application's compute–communication ratio and lower bandwidth makes the local execution optimal, as opposed to offloading to the cloud node.

The sensing configuration of an application alters the volume of data generated for transmission and processing. For instance, consider the scenario in Fig. 8c, under low sampling rate and medium bandwidth. As the sampling rate is lower, the penalty incurred in transmitting data to the cloud node is minimized. With lower data volume and availability of either medium (and/or high), bandwidth results in significant improvement in latency.

It should be noted that lowering the sampling rate potentially sacrifices the accuracy, although the latency is improved. While the accuracy loss is subjective to the error resilience of the application, missing insightful input data samples could lead to mis-predictions and critical errors. Mis-predictions can be minimized by setting an upper bound on accuracy requirements and/or bounds on lowering the sampling rates. In such scenarios, the error resilience of an application also influences the compute placement decisions.

3.2 Using RL for Optimization

Finding the optimal orchestration policy for an unknown and dynamic system is critical since dynamicity of environment (e.g., network condition, workload arrival at computing nodes, user traffic, and application characteristics) changes over time. Most current solutions are based on design time optimization, without considerations on varying system dynamics at run-time [6, 12–14, 35, 42, 46, 52, 65, 79, 80]. A complex system that runs a variety of applications in uncertain environmental conditions requires dynamic control to offer high-performance or low-power guarantees [33, 59, 61, 62, 64]. Considering the run-time variation of system dynamics, and making an optimal orchestration choice, requires intelligent monitoring, analysis, and decision-making. Existing heuristic and rule-based orchestration methods require an extensive design space exploration to make optimal compute placement decisions. Furthermore, such a solution based on exhaustive search at run-time becomes practically infeasible for latency critical

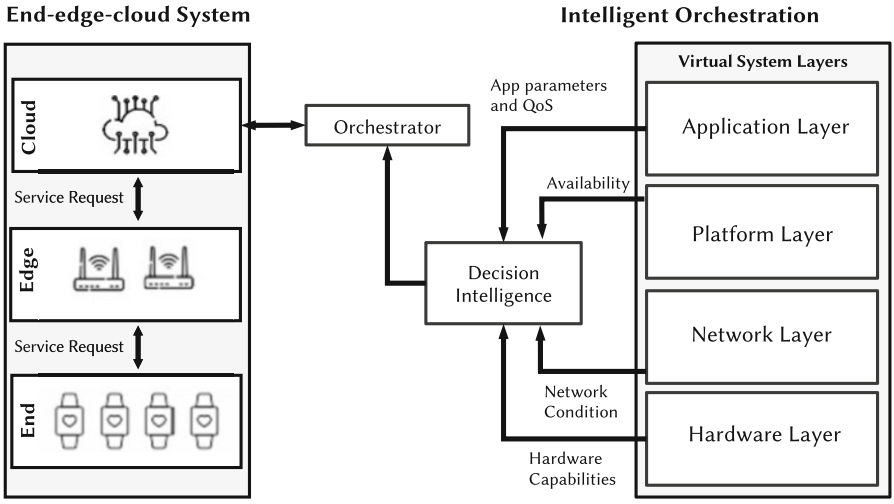


Fig. 9 Overview of intelligent orchestration in end–edge–cloud architectures [64]

services. In this context, different offline and online machine learning models have been adopted for run-time resource management of distributed systems, to handle the complexity of orchestration choices. Among these models, reinforcement learning (RL) approach is effective in developing an understanding and interpreting varying system dynamics [49, 54]. Reinforcement learning enables identification of complex dynamics between influential system parameters and online decision-making to optimize objectives such as response time, energy consumption, and quality of service [67]. The RL approach allows formulating policies at run-time using the input data collected over time. Specifically, the RL approach uses a reward function to quantify the effect of an action on the system state. This allows optimizing orchestration choices over time, considering the system-wide context and objectives. In this subsection, we present the design of reinforcement learning agent for orchestrating ML workloads on sensor–edge–cloud platforms.

3.2.1 Orchestration Framework

Figure 9 shows an overview of generic intelligent orchestration framework for multi-layered end–edge–cloud architectures [64]. This framework uses system-wide information for intelligent orchestration through virtual system layers that include application, platform, network, and hardware layers. Each of the virtual system layers provides inputs for monitoring system and application dynamics such as application adjustment parameters, accuracy requirements, availability of devices for execution, network characteristics, and hardware capabilities. Each execution choice affects the performance and energy consumption of the user end-device, based on the system parameters such as hardware capabilities, network conditions,

and workload characteristics. Each layer exhibits a diverse set of requirements, constraints, and opportunities to trade-off performance and efficiency that vary over time. For example, the application layer focuses on the user's perception of algorithmic correctness of services, while the platform layer focuses on improving system parameters such as energy drain and data volume migrated across nodes. Both application and platform layers have different measurable metrics and controllable parameters to expose different opportunities that can be exploited for meeting overall objectives. The network layer provides connectivity for data and control transfer between different physical devices. In addition, the hardware layer provides hardware capabilities for computing nodes in the system. Run-time system dynamics affect orchestration strategies significantly in addition to requirements and opportunities. Sources of run-time variation across the system stack include workload of a specific computing node, connectivity and signal strength of the network, mobility and interaction of a given user, etc. Information on run-time cross-layer requirements and run-time variations provides necessary feedback to make appropriate decisions on system configurations such as offloading policies.

3.2.2 RL Agent for Orchestration

Making the optimal orchestration choice considering these varying dynamics is an NP-hard problem, while brute force search of a large configuration space is impractical for real-time applications. Understanding the requirements at each level of the system stack and translating them into measurable metrics enables appropriate orchestration decision-making. On the other hand, heuristic, rule-based, and closed-loop feedback control solutions are slow in convergence due to the large state space [67]. To address these limitations, reinforcement learning (RL) approaches have been adapted for the computation offloading problem [58]. RL builds specific models based on data collected over initial epochs and dramatically improves the prediction accuracy [67]. We design an RL agent that monitors system-wide parameters and chooses a suitable action that maximizes the efficiency of orchestration decisions.

Figure 10 shows the workflow of the RL agent for making orchestration decisions. The RL agent component is deployed within the decision intelligence and orchestration blocks in orchestration framework (Fig. 2). The RL agent receives resource information (e.g., processor utilization, available memory, available bandwidth) from the virtual system layers of the orchestration framework (Fig. 2). The RL agent also collects the reward information (response time in this case) from the environment to learn an optimal action that maximizes the reward. The agent builds the Q-Table for Q-Learning algorithm, based on cumulative reward obtained from the environment over time. This signifies the efficacy of a specific orchestration decision (action) in achieving the target of minimizing latency and enables subsequent optimal orchestration decisions.

We demonstrate the efficacy of using the RL agent for orchestrating ML workloads on sensor–edge–cloud platforms. While the focus of this chapter is

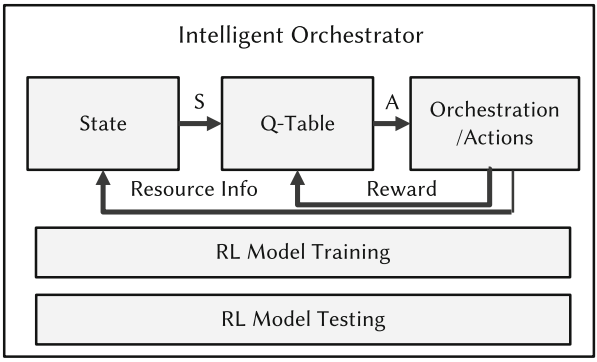


Fig. 10 RL agent for intelligent orchestration

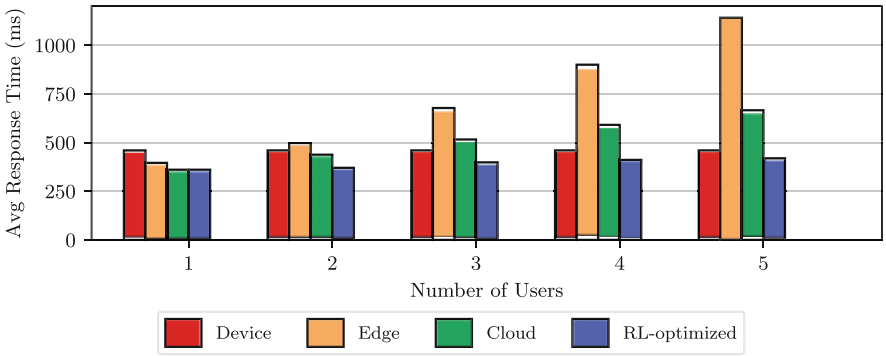


Fig. 11 DL inference orchestration in an end–edge–cloud system which runs an image classification application

optimizing smart eHealth applications, we use image classification task in the experimentation for the purpose of demonstrating multi-user ML workloads. We implement a scenario where a device–edge–cloud architecture serves up to five end-users to execute ML services simultaneously. In this scenario, the users send requests for image classification to an agent located at the cloud. In addition, end-users share their resource availability to the agent. The agent decides to orchestrate the ML tasks based on three static (i.e., device-only, edge-only, and cloud-only) and one RL-optimized strategies. In the device-only strategy, each end-device executes the inference service on a local device. Thus, varying a number of users has no effect on the average response time in this case. In the edge- and cloud-only strategies, simultaneous requests compete for edge and cloud resources. This increases the average response time significantly, as the number of users increases. In the **RL-optimized** strategy, the resource availability is continuously observed and the ML tasks are orchestrated accordingly.

Figure 11 shows the average response time for different numbers of active users for regular network conditions, using different orchestration strategies. The x-axis represents the number of active users. Each bar represents a different orchestration

decision made by using the corresponding orchestration strategy [62, 64]. In RL-optimized approach, the average response time remains constant, while the number of users is less than three. This is due to the orchestration decision of distributing the services across edge and cloud layers. As the number of users increases to three, the services start competing for resources, leading to an increase in the average response time. With the number of users increasing from three to five, the average response time increases, but at a relatively lower rate, exhibiting efficient utilization of the edge and cloud resources. As the number of users increases, the efficiency of the RL-optimized approach over the static strategies is more prominent.

4 Sense–Compute Co-optimization of ML-driven eHealth Applications

In this section, we describe sense–compute co-optimization approaches for improving resiliency of smart eHealth applications. Common use cases of ML models often handle complete and clean input data, with no specific sensing challenges. However, using ML methods in smart eHealth applications on edge devices requires considerations on challenges from the sensory data acquisition phase [73]. With different types of implanted, wearable, on-body, and remote sensors, there is a higher probability of noise, motion artifacts, and missing input data from sensors [44]. For critical healthcare IoT applications, input data perturbation from motion artifacts, physical failure of sensors, network anomalies, and other factors can affect the prediction accuracy of ML models significantly [50]). On the other hand, the sense-making (computation) phase of eHealth applications faces the challenge of limited computational capacity of the edge devices for running ML models [73]. Additionally, the ML models should be resilient to probable sensing anomalies like noisy or missing input data and maintain higher prediction accuracy even with potential garbage input signals [44]. Moreover, some applications (e.g., pain assessment in clinical healthcare monitoring systems) require near real-time response time, emphasizing the need for ML inference performance [33]. Addressing these multitude of challenges necessitates a co-optimization approach that jointly handles sensing and sense-making phases for system-wide exploration of suitable optimizations. A simple schematic for the interaction between sensing and sense-making modules is shown in Fig. 12. Sensing awareness can be developed

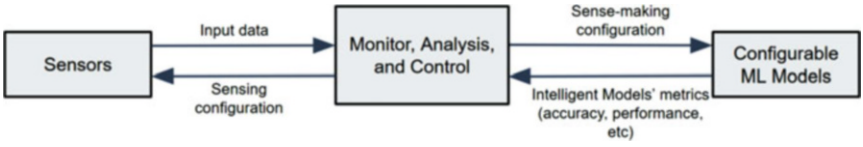


Fig. 12 Sensing and sense-making (compute) modules' interaction with their action knobs in sensor level and intelligent models

by monitoring and analyzing continuous streams of input data. This intelligence can be used to control the sensing and sense-making configurations simultaneously, by fine-tuning ML models to fit input data characteristics.

4.1 Handling Input Data Perturbations

4.1.1 Sensing Phase Knobs

Monitoring input data originating from the sensory devices for anomalies, discrepancies, noisy components, etc. provides insights into quality of input data [43]. Understanding the quality of input data can be exploited for selective sensing, such that unreliable sensors can be down-sampled, to dampen the effect of unreliable input data. When the input data from a specific sensor is noisy, the sampling rate of the sensory device can be decreased so that the garbage data would not waste the computational resources in the edge layer. In acute scenarios, input data from a specific device might be completely unreliable (e.g., when a heartbeat sensor is detached from the human body). In this case, detection of the sensor detachment can guide the computation phase to ignore the input from that specific device and rely on the data from other input modalities [48]. It should be noted that both selective sensing and disabling a sensor modality minimize the network latency penalty with reduced input data volume. In specific scenarios of unreliable network connection, sensing phase knobs can be opportunistically triggered to complement the network delay with reduced input data volume [78].

4.1.2 Sense-Making Phase Knobs

Addressing input data perturbation from multiple modalities requires appropriate and proportional actions in the ML algorithmic phase. For instance, selective or greedy feature selection from the preprocessed input data can reduce the noisy data components fed into the ML models [40]. This approach is effective in reducing unnecessary computation over noisy input data, although potentially affecting the accuracy of the learning models [51]. Selecting an appropriate ML model from a pool of pretrained models is another strategy for handling noisy and missing input data. This requires implementation of a model pool, consisting of different ML models that are trained to handle specific combinations of input modalities [51]. Algorithmic optimizations such as meta-learning [21], fast reinforcement learning [9], and few-shot learning [75] can also be used for providing efficient ML optimizations particularly for edge devices.

4.1.3 Co-optimization Knobs

Besides the presented methods for optimizing sensing and sense-making phases in the edge devices, it is quite important to use co-optimization techniques so that each of these phases complements the other to form a holistic system with efficiency and robustness. To achieve this goal, meaningful interaction between these phases is required. For example, the sensing information from input data can be used as a trigger to adjust specific sense-making knobs so that the computation models can adapt to the recent changes in the input sources. One example of this approach is using the early exit technique [28, 68] in the neural network when the input data has good quality with negligible noise. In this way, an acceptable confidence threshold can be achieved in less time by skipping deeper layers in neural network models. Moreover, the sensing module in the edge layer can send information about input sources with high or low reliability of their data to the sense-making module, and then the machine learning models in the edge layer can adjust importance weights to those input sources by using attention mechanisms inside their architecture [53, 71].

4.1.4 Example Scenarios

We demonstrate the advantages of sense–compute co-optimization through an example of multi-modal pain assessment application [51]. The pain assessment application uses inputs from three modalities: Electrocardiography (ECG), Electrodermal Activity (EDA), and Photoplethysmography (PPG). The ECG, EDA, and PPG modalities have sampling rates of 100, 4, and 64 and generate 52, 42, and 42 features. Figure 13 outlines sensor data acquisition, feature extraction, feature aggregation, and inference task for predicting pain levels under different scenarios. Figure 13a shows the scenario, where the application is executed without modality awareness. In this scenario, data from the ECG sensor is noisy, yet feature vectors from the noisy modality are fed into the ML model, yielding a baseline prediction accuracy of 51%. Figure 13b shows the scenario where modality awareness is considered while executing the application. In this scenario, the application is executed with selective feature aggregation, by selecting fewer features from the noisy ECG modality. This reduces the total number of features from the ECG modality to 12. An appropriate ML model to suit the updated feature vector is selected from a model pool, which comprises pretrained models. Minimizing the features from noisy ECG modality improves the prediction accuracy to 79%, while also reducing the energy consumption and improving the performance, in comparison with the baseline scenario (a). Figure 13c shows the scenario where modality awareness is used to select specific modalities with quality input data. In this scenario, the noisy ECG modality is completely dropped, and data from the EDA and PPG modalities is processed. Similar to the scenario (b), an appropriate ML model that suits EDA and PPG inputs is selected from the model pool. Dropping an entire modality of data significantly reduces the computational effort and energy consumption, in comparison with scenarios (a) and (b). It should be noted that

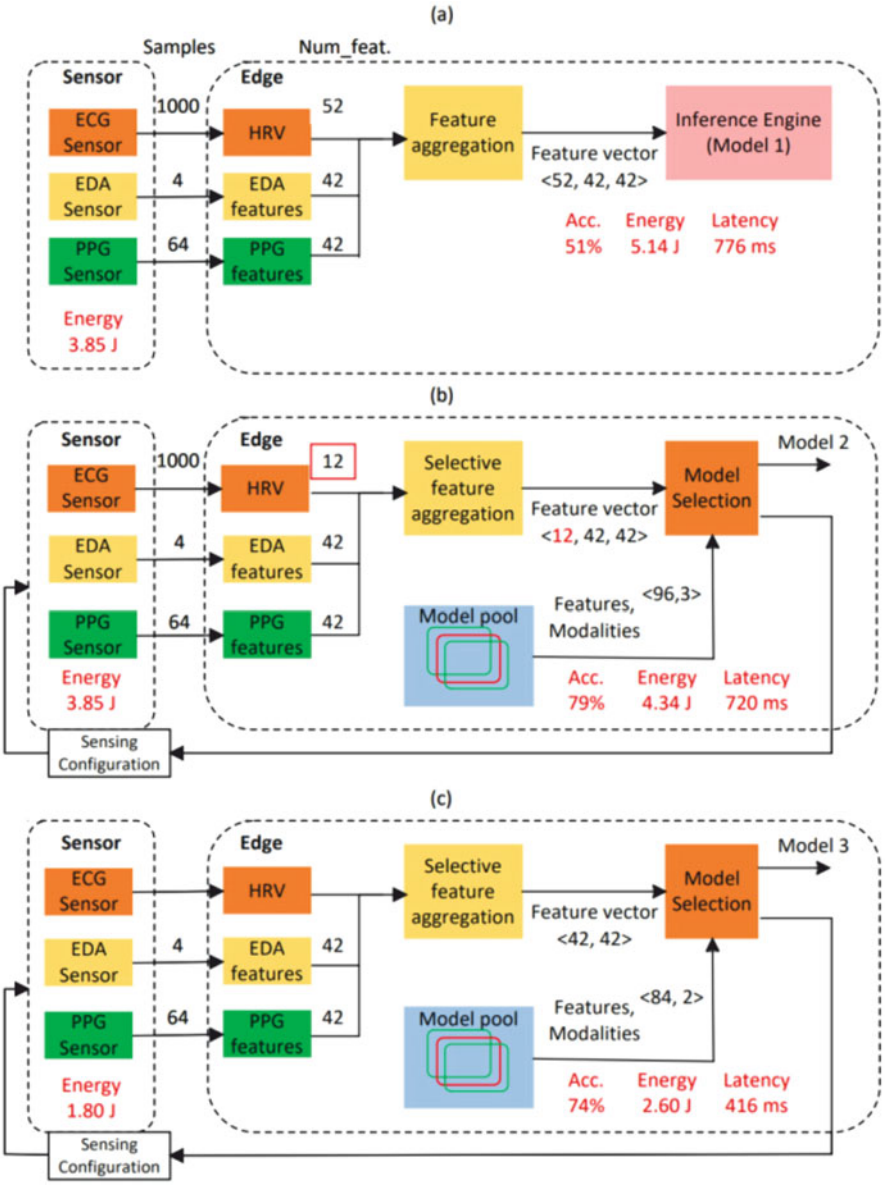


Fig. 13 Motivational scenarios for sense–compute co-optimization [51]. (**a**) Processing noisy sensory input data. (**b**) Processing with selective feature aggregation and model selection. (**c**) Processing with modality selection and model selection

the prediction accuracy with only two modalities is 74%, which is higher than the baseline from scenario (a), while being marginally lower than the prediction accuracy from selective feature aggregation from scenario (b).

4.2 Sense–Compute Co-optimization Framework

We present the conceptual design of sense–compute co-optimization for multi-modal eHealth applications through the AMSER framework [51]. Figure 14 shows an overview of the AMSER framework for sense–compute co-optimization in multi-layered sensor–edge–cloud platforms. The AMSER framework uses run-time monitoring functionalities of signal quality monitoring, discrepancy detection, abnormality detection, and confidence level monitor to understand the quality of input data modalities and confidence of the ML model in predicting results. Insights from the run-time monitoring are used to configure the sampling rates of different sensory devices through the sensing controller. Based on the run-time monitoring, the sense-making (compute) optimizer uses ML configuring knobs—adaptive feature selection, model selection, neural network attention, and early exit mechanisms to configure ML models for current input data sets.

For example, data from a specific modality is labeled as uncertain when the signal quality is below a specific Signal-to-Noise Ratio (SNR). Under such scenarios, the edge-level sensing optimizer feeds the learning models with reliable input modalities, while dropping the noisy modality. The sense-making optimizer then selects an ML model from the model pool, which is suitable for the available input data modalities. The model pool contains different pretrained ML models suitable for different combinations of reliable input modalities.

The results for accuracy and performance (speedup) gain with sense–compute co-optimization in comparison with the baseline [29] are shown in Figs. 15 and 16. ML-driven eHealth applications of pain assessment and stress monitoring applications were used as input workloads. For a comprehensive evaluation, four different scenarios (S1–S4) with different noise components in input data of modalities are used. Scenario 1 (S1) is the baseline with no noise components. Scenario 2 (S2) has a wandering noise added to the original data. Scenario 3 has an additional motion artifact added to one input modality on top of the wandering noise, which makes that modality completely unreliable. In scenario 4 (S4), two modalities suffer from severe motion artifact noise. In scenario 2 (S2), the AMSER framework activates the selective feature selection to handle the noisy input modalities, resulting in higher prediction accuracy in comparison with the baseline. In scenario 3 with unreliable input (S3), the AMSER approach drops the entire unreliable modality, yielding a better prediction accuracy and performance gain. Similarly, the AMSER framework drops two noisy modalities in scenario 4 (S4), resulting in better prediction accuracy and significantly higher performance. The sensing awareness and synergistic compute knob actuation of the AMSER platform provides

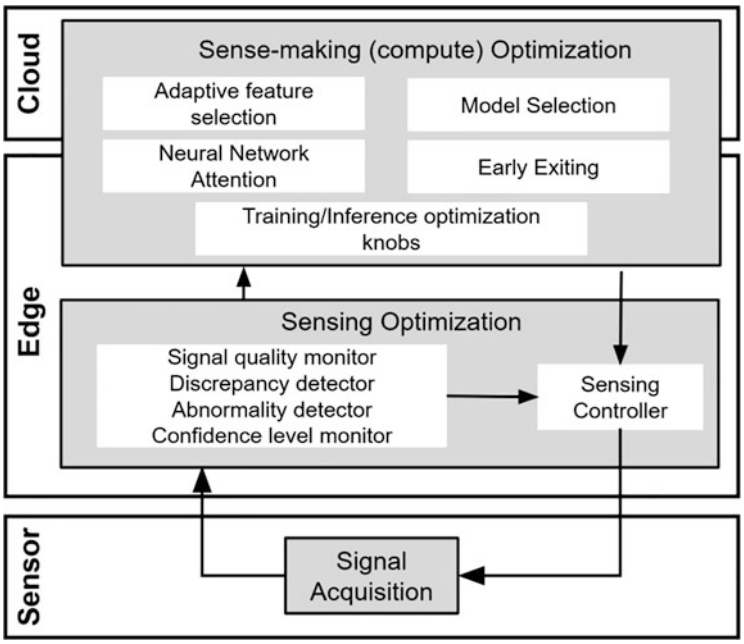


Fig. 14 An overview of sensing and sense-making (compute) optimization in a sensor–edge–cloud architecture

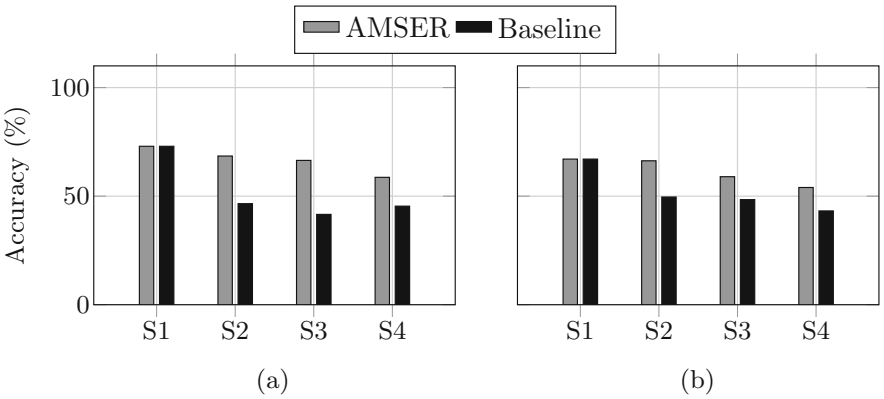


Fig. 15 Accuracy analysis in comparison with baseline [29, 51]. (**a**) Pain. (**b**) Stress

significant improvements in accuracy, efficiency, and performance, in comparison with existing disjoint sensing and compute optimization approaches.

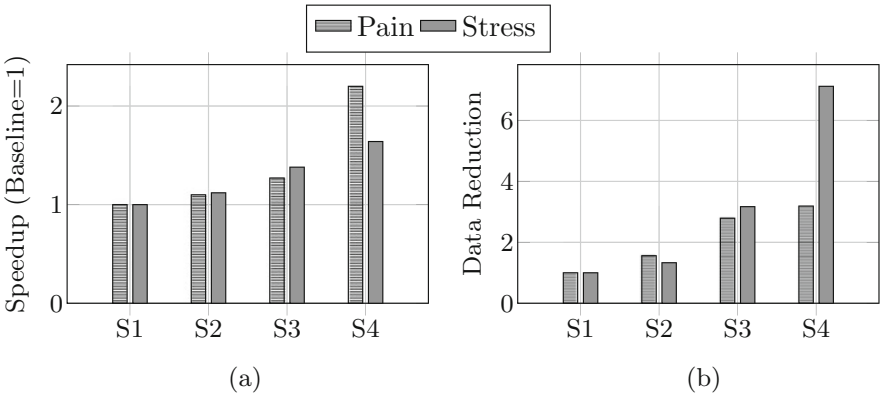


Fig. 16 (**a**) Performance analysis of the edge layer device for AMSER framework vs. baseline study [29]. (**b**) Data transfer volume between the sensors and edge layer device for AMSER framework vs. baseline study. (**a**) Performance gain. (**b**) Data volume reduction

5 Conclusions

In this chapter, we presented edge-centric optimizations for ML-driven eHealth applications through compute placement, improving the compute placement decisions through reinforcement learning agent, and cross-layered sense–compute co-optimizations. We also presented an exemplar case study of objective pain assessment to demonstrate the use cases of edge-ML-based smart eHealth applications, common data flows, computation challenges, and frameworks for deploying smart eHealth applications.

5.1 Key Insights

ML-driven smart healthcare applications have different input data characteristics, computational requirements, and quality metrics. Continuous stream of input data, varying network conditions, and computational requirements of different ML models create dynamic workload scenarios. At an application level, requirements include higher prediction accuracy of ML models, latency of inferencing results from ML models, resilience, and an overall higher quality of service. At a system-level, requirements include availability of compute nodes in edge and cloud layers, compute capabilities of edge nodes to meet performance requirements of ML models, network utilization, and overall energy efficiency. Considering both application and system-level parameters simultaneously is necessary for optimizing edge-centric ML-driven smart eHealth applications. Optimized compute placement has higher efficacy in meeting both application and system requirements simultaneously. Accuracy–performance trade-offs can also be explored within the compute

placement phase, by configuring the choice of ML models. Both model-based and model-free reinforcement learning agents can guide the compute placement decisions on choice of execution node and tuning accuracy–performance trade-offs with high degree of convergence. Further, multi-modal eHealth applications are prone to input data perturbations, which also presents an opportunity to exploit the inherent resilience to selectively process input data. This brings sensing awareness into computation, and compute-awareness to sensing through bi-directional feedback. Cross-layered sense–compute co-optimization improves sensing, computation, and communication aspects of edge-ML-based eHealth applications holistically.

5.2 Open Research Directions

5.2.1 Data Quality Management

Input data quality is an essential component for improving prediction accuracy of ML-driven smart eHealth applications. Processing exclusively quality input data also improves the bandwidth utilization and latency of tasks run on the edge nodes. Some of the techniques presented in this chapter address the sensing aspects through continuous monitoring and analysis of input data quality. Qualitative assessment of sensory data can be improved significantly beyond the rule-based monitors using cognitive learning models. Design of autonomous models for input data quality management remains an open challenge. Autonomous models enable reasoning for different input perturbations to assess true quality of sensory inputs. Consequently, the garbage data that is un-necessarily processed is minimized, supporting the scalability of edge-ML solutions. Quality assessment of input data is significant in other sensor-driven domains such as autonomous driving, robotics, and computer vision etc.

5.2.2 Contextual Edge Orchestration

Orchestration techniques for collaborative sensor-edge–cloud architectures improve a multitude of metrics in terms of performance, turnaround time, energy efficiency, accuracy trade-off exploration, and network utilization. Orchestration techniques presented in this chapter enable intelligent compute placement, offloading, and accuracy configuration decisions through rule-based heuristics. However, contextualization of system dynamics to reason for orchestration decisions, and exploration of accuracy–performance–energy trade-offs with context-awareness is an open research direction. Reinforcement learning has been widely used for optimal orchestration decisions at run-time, considering the varying system dynamics. The efforts in collecting training data, online updating, and convergence time influence the efficacy of such learning methods. In this perspective, design of model-free,

and few-shot learning models for run-time edge orchestration is a promising open research direction.

5.2.3 Sense–Compute Co-optimization

Cross-layered sense–compute co-optimization is the most effective strategy for improving sensor-dependent, edge-based ML applications. In this chapter, we presented adaptive sensing and sensing-aware computing techniques that uses system-wide monitoring and intelligence for sense–compute co-optimization. This approach focuses on selecting appropriate ML models based on quality of input modalities, exploiting the inherent resilience of multi-modal ML applications. Adaptive feature selection, and ML model selection enforce the idea of sense–compute co-optimization at a coarse-grained level, and require multiple pretrained models. Incorporating fine-grained edge layer ML model configurations such as early exit, greedy feature selection, neural network model attention, and saliency maps etc., can complement the model selection strategy. The feasibility of such edge layer-based ML model tuning in collaboration with cloud layer-based model selection is another open research direction.

References

1. Adibuzzaman, M., Ostberg, C., Ahamed, S., Povinelli, R., Sindhu, B., Love, R., Kawsar, F., Ahsan, G.M.T.: Assessment of pain using facial pictures taken with a smartphone. In: 2015 IEEE 39th Annual Computer Software and Applications Conference, vol. 2, pp. 726–731. IEEE, Piscataway (2015)
2. Aqajari, S.A.H., Cao, R., Kasaeyan Naeini, E., Calderon, M.D., Zheng, K., Dutt, N., Liljeberg, P., Salanterä, S., Nelson, A.M., Rahmani, A.M.: Pain assessment tool with electrodermal activity for postoperative patients: method validation study. JMIR Mhealth Uhealth **9**(5), e25258 (2021)
3. Aqajari, S.A.H., Naeini, E.K., Mehrabadi, M.A., Labbaf, S., Rahmani, A.M., Dutt, N.: GSR analysis for stress: Development and validation of an open source tool for noisy naturalistic GSR data (2020). arXiv preprint arXiv:2005.01834
4. Arif-Rahu, M., Grap, M.J.: Facial expression and pain in the critically ill non-communicative patient: state of science review. Intensive Crit. Care Nursing **26**(6), 343–352 (2010)
5. Azimi, I., et al.: HiCH: hierarchical fog-assisted computing architecture for healthcare IoT. ACM Trans. Embedded Comput. Syst. **16**(5), 1–20 (2017)
6. Bao, W., Li, W., Delicato, F.C., Pires, P.F., Yuan, D., Zhou, B.B., Zomaya, A.Y.: Cost-effective processing in fog-integrated internet of things ecosystems. In: Proceedings of the 20th ACM International Conference on Modelling, Analysis and Simulation of Wireless and Mobile Systems, pp. 99–108 (2017)
7. Barbera, M.V., Kosta, S., Mei, A., Stefa, J.: To offload or not to offload? The bandwidth and energy costs of mobile cloud computing. In: 2013 Proceedings IEEE Infocom, pp. 1285–1293. IEEE, Piscataway (2013)
8. Barr, J., Fraser, G.L., Puntillo, K., Ely, E.W., Gélinas, C., Dasta, J.F., Davidson, J.E., Devlin, J.W., Kress, J.P., Joffe, A.M., et al.: Clinical practice guidelines for the management of pain, agitation, and delirium in adult patients in the intensive care unit. Crit. Care Med. **41**(1), 263–306 (2013)

9. Barreto, A., Hou, S., Borsa, D., Silver, D., Precup, D.: Fast reinforcement learning with generalized policy updates. Proc. Natl. Acad. Sci. **117**(48), 30079–30087 (2020). https://www.pnas.org/doi/abs/10.1073/pnas.1907370117
10. Breivik, H., Borchgrevink, P.C., Allen, S.M., Rosseland, L.A., Romundstad, L., Breivik Hals, E., Kvarstein, G., Stubhaug, A.: Assessment of pain. Br. J. Anaesth. **101**(1), 17–24 (2008)
11. Cao, R., Aqajari, S., Kasaeyan Naeini, E., Rahmani, A.M.: Objective pain assessment using wrist-based ppg signals: A respiratory rate based method. In: 43rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC). IEEE, Piscataway (2021). Accepted for publication
12. Cao, X., Wang, F., Xu, J., Zhang, R., Cui, S.: Joint computation and communication cooperation for mobile edge computing. In: 2018 16th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt), pp. 1–6. IEEE, Piscataway (2018)
13. Chamola, V., Tham, C.K., Chalapathi, G.S.: Latency aware mobile task assignment and load balancing for edge cloudlets. In: 2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), pp. 587–592. IEEE, Piscataway (2017)
14. Chang, Z., Zhou, Z., Ristaniemi, T., Niu, Z.: Energy efficient optimization for computation offloading in fog computing system. In: GLOBECOM 2017-2017 IEEE Global Communications Conference, pp. 1–6. IEEE, Piscataway (2017)
15. Chatzaki, C., Pediaditis, M., Vavoulas, G., Tsiknakis, M.: Human daily activity and fall recognition using a smartphone's acceleration sensor. In: International Conference on Information and Communication Technologies for Ageing Well and e-Health, pp. 100–118. Springer, Berlin (2016)
16. Chetty, G., Yamin, M.: Intelligent human activity recognition scheme for eHealth applications. Malaysian J. Comput. Sci. **28**(1), 59–69 (2015)
17. Dogan, A.Y., Constantin, J., Ruggiero, M., Burg, A., Atienza, D.: Multi-core architecture design for ultra-low-power wearable health monitoring systems. In: 2012 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 988–993. IEEE, Piscataway (2012)
18. Duch, L., Basu, S., Braojos, R., Ansaloni, G., Pozzi, L., Atienza, D.: Heal-wear: an ultra-low power heterogeneous system for bio-signal analysis. IEEE Trans. Circuits Syst. I: Regul. Pap. **64**(9), 2448–2461 (2017)
19. Eshratifar, A.E., Abrishami, M.S., Pedram, M.: JointDNN: an efficient training and inference engine for intelligent mobile cloud computing services. IEEE Trans. Mobile Comput. **20**(2), 565–576 (2019)
20. Farahani, B., Barzegari, M., Aliee, F.S., Shaik, K.A.: Towards collaborative intelligent IoT eHealth: from device to fog, and cloud. Microprocess. Microsyst. **72**, 102938 (2020)
21. Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. In: International Conference on Machine Learning, pp. 1126–1135. PMLR (2017)
22. Gia, T.N., Jiang, M., Rahmani, A.M., Westerlund, T., Liljeberg, P., Tenhunen, H.: Fog computing in healthcare internet of things: a case study on ECG feature extraction. In: 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing, pp. 356–363. IEEE, Piscataway (2015)
23. Greene, S., Thapliyal, H., Caban-Holt, A.: A survey of affective computing for stress detection: Evaluating technologies in stress detection for better health. IEEE Consum. Electron. Mag. **5**(4), 44–56 (2016)
24. Gruss, S., Treister, R., Werner, P., Traue, H.C., Crawcour, S., Andrade, A., Walter, S.: Pain intensity recognition rates via biopotential feature patterns with support vector machines. PLoS One **10**(10), e0140330 (2015)
25. Gupta, D., Rodrigues, J.J., Peng, S.L., Nguyen, N.: Artificial intelligence for eHealth. Front. Public Health **10** (2022)
26. Han, H.J., et al.: Objective stress monitoring based on wearable sensors in everyday settings. J. Med. Eng. Technol. **44**(4), 177–189 (2020)

27. Jiang, M., Mieronkoski, R., Rahmani, A.M., Hagelberg, N., Salanterä, S., Liljeberg, P.: Ultra-short-term analysis of heart rate variability for real-time acute pain monitoring with wearable electronics. In: 2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), pp. 1025–1032. IEEE, Piscataway (2017)
28. Ju, W., Bao, W., Ge, L., Yuan, D.: Dynamic Early Exit Scheduling for Deep Neural Network Inference through Contextual Bandits, pp. 823–832. Association for Computing Machinery, New York (2021). https://doi.org/10.1145/3459637.3482335
29. Kächele, M., Thiam, P., Amirian, M., Werner, P., Walter, S., Schwenker, F., Palm, G.: Multimodal data fusion for person-independent, continuous estimation of pain intensity. In: Iliadis, L., Jayne, C. (eds.) Engineering Applications of Neural Networks, pp. 275–285. Springer, Cham (2015)
30. Kächele, M., Werner, P., Al-Hamadi, A., Palm, G., Walter, S., Schwenker, F.: Bio-visual fusion for person-independent recognition of pain intensity. In: International Workshop on Multiple Classifier Systems, pp. 220–230. Springer, Berlin (2015)
31. Kasaeyan Naeini, E., Jiang, M., Syrjälä, E., Mieronkoski, R., Calderon, M.D., Zheng, K., Dutt, N., Liljeberg, P., Salanterä, S., Nelson, A., Rahmani, A.M.: Research protocol for the smart pain assessment employing behavioral and physiologic indicators. In: JMIR Journal of Research Protocols (revision submitted) (2020)
32. Kasaeyan Naeini, E., Jiang, M., Syrjälä, E., Calderon, M.D., Mieronkoski, R., Zheng, K., Dutt, N., Liljeberg, P., Salanterä, S., Nelson, A.M., Rahmani, A.M.: Prospective study evaluating a pain assessment tool in a postoperative environment: Protocol for algorithm testing and enhancement. JMIR Res. Protoc. **9**(7), e17783 (2020)
33. Kasaeyan Naeini, E., Shahhosseini, S., Subramanian, A., Yin, T., Rahmani, A.M., Dutt, N.: An edge-assisted and smart system for real-time pain monitoring. In: 2019 IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE), pp. 47–52 (2019)
34. Kasaeyan Naeini, E., Subramanian, A., Calderon, M.D., Zheng, K., Dutt, N., Liljeberg, P., Salantera, S., Nelson, A.M., Rahmani, A.M.: Pain recognition with electrocardiographic features in postoperative patients: method validation study. J. Med. Int. Res. **23**(5), e25079 (2021)
35. Kattepur, A., Dohare, H., Mushunuri, V., Rath, H.K., Simha, A.: Resource constrained offloading in fog computing. In: Proceedings of the 1st Workshop on Middleware for Edge Clouds & Cloudlets, pp. 1–6 (2016)
36. Khan, M.A., Alkaabi, N.: Rebirth of distributed ai—a review of eHealth research. Sensors **21**(15), 4999 (2021)
37. Khelifi, H., Luo, S., Nour, B., Sellami, A., Moungla, H., Ahmed, S.H., Guizani, M.: Bringing deep learning at the edge of information-centric internet of things. IEEE Commun. Lett. **23**(1), 52–55 (2018)
38. Koelstra, S., Muhl, C., Soleymani, M., Lee, J.S., Yazdani, A., Ebrahimi, T., Pun, T., Nijholt, A., Patras, I.: DEAP: A database for emotion analysis; using physiological signals. IEEE Trans. Affective Comput. **3**(1), 18–31 (2011)
39. Kreps, G.L., Neuhauser, L.: New directions in eHealth communication: opportunities and challenges. Patient Educ. Couns. **78**(3), 329–336 (2010)
40. Kwak, N., Choi, C.H.: Input feature selection for classification problems. IEEE Trans. Neural Netw. **13**(1), 143–159 (2002)
41. Laitala, J., Jiang, M., Syrjälä, E., Naeini, E.K., Airola, A., Rahmani, A.M., Dutt, N.D., Liljeberg, P.: Robust ECG R-peak detection using LSTM. In: Proceedings of the 35th Annual ACM Symposium on Applied Computing, pp. 1104–1111 (2020)
42. Liu, J., Mao, Y., Zhang, J., Letaief, K.B.: Delay-optimal computation task scheduling for mobile-edge computing systems. In: 2016 IEEE International Symposium on Information Theory (ISIT), pp. 1451–1455. IEEE, Piscataway (2016)
43. Lou, P., Shi, L., Zhang, X., Xiao, Z., Yan, J.: A data-driven adaptive sampling method based on edge computing. Sensors **20**(8) (2020). https://www.mdpi.com/1424-8220/20/8/2174

44. Ma, M., Ren, J., Zhao, L., Tulyakov, S., Wu, C., Peng, X.: Smil: Multimodal learning with severely missing modality (2021). arXiv preprint arXiv:2103.05677
45. Mach, P., Becvar, Z.: Mobile edge computing: A survey on architecture and computation offloading. IEEE Commun. Surv. Tutorials **19**(3), 1628–1656 (2017)
46. Mao, Y., Zhang, J., Song, S., Letaief, K.B.: Power-delay tradeoff in multi-user mobile-edge computing systems. In: 2016 IEEE Global Communications Conference (GLOBECOM), pp. 1–6. IEEE, Piscataway (2016)
47. Merskey, H.: Pain terms: a list with definitions and notes on usage. Recommended by the IASP subcommittee on taxonomy. Pain **6**, 249–252 (1979)
48. Montesinos, V., Dell'Agnola, F., Arza, A., Aminifar, A., Atienza, D.: Multi-modal acute stress recognition using off-the-shelf wearable devices. In: 2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), pp. 2196–2201 (2019)
49. Mousavi, S.S., Schukat, M., Howley, E.: Deep reinforcement learning: an overview. In: Proceedings of SAI Intelligent Systems Conference, pp. 426–440. Springer, Berlin (2016)
50. Naeini, E.K., Azimi, I., Rahmani, A.M., Liljeberg, P., Dutt, N.: A real-time ppg quality assessment approach for healthcare Internet-of-Things. Proc. Comput. Sci. **151**, 551–558 (2019)
51. Naeini, E.K., Shahhosseini, S., Kanduri, A., Liljeberg, P., Rahmani, A.M., Dutt, N.: AMSER: Adaptive multi-modal sensing for energy efficient and resilient eHealth systems. IEEE/ACM Design, Automation and Test in Europe Conference (DATE'22) (2022)
52. Nan, Y., Li, W., Bao, W., Delicato, F.C., Pires, P.F., Zomaya, A.Y.: A dynamic tradeoff data processing framework for delay-sensitive applications in cloud of things systems. J. Parallel Distrib. Comput. **112**, 53–66 (2018)
53. Ning, H., Ye, X., Sada, A.B., Mao, L., Daneshmand, M.: An attention mechanism inspired selective sensing framework for physical-cyber mapping in internet of things. IEEE Internet Things J. **6**(6), 9531–9544 (2019)
54. Park, J., Samarakoon, S., Bennis, M., Debbah, M.: Wireless network intelligence at the edge. Proc. IEEE **107**(11), 2204–2239 (2019)
55. Rahmani, A.M., Gia, T.N., Negash, B., Anzanpour, A., Azimi, I., Jiang, M., Liljeberg, P.: Exploiting smart e-health gateways at the edge of healthcare Internet-of-Things: a fog computing approach. Fut. Gener. Comput. Syst. **78**, 641–658 (2018)
56. Schapire, R.E.: Explaining AdaBoost. In: Empirical Inference, pp. 37–52. Springer, Berlin (2013)
57. Schmidhuber, J.: Deep learning in neural networks: an overview. Neural Netw. **61**, 85–117 (2015)
58. Sen, T., Shen, H.: Machine learning based timeliness-guaranteed and energy-efficient task assignment in edge computing systems. In: 2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC), pp. 1–10. IEEE, Piscataway (2019)
59. Seo, D., Shahhosseini, S., Mehrabadi, M.A., Donyanavard, B., Lim, S.S., Rahmani, A.M., Dutt, N.: Dynamic iFogSim: A framework for full-stack simulation of dynamic resource management in IoT systems. In: 2020 International Conference on Omni-Layer Intelligent Systems (COINS), pp. 1–6. IEEE, Piscataway (2020)
60. Shahhosseini, S., Anzanpour, A., Azimi, I., Labbaf, S., Seo, D., Lim, S.S., Liljeberg, P., Dutt, N., Rahmani, A.M.: Exploring computation offloading in IoT systems. Inform. Syst. **107**, 101860 (2022)
61. Shahhosseini, S., Azimi, I., Anzanpour, A., Jantsch, A., Liljeberg, P., Dutt, N., Rahmani, A.M.: Dynamic computation migration at the edge: is there an optimal choice? In: Proceedings of the 2019 on Great Lakes Symposium on VLSI, pp. 519–524 (2019)
62. Shahhosseini, S., Hu, T., Seo, D., Kanduri, A., Donyanavard, B., Rahmani, A.M., Dutt, N.: Hybrid learning for orchestrating deep learning inference in multi-user edge-cloud networks (2022). arXiv preprint arXiv:2202.11098
63. Shahhosseini, S., Kanduri, A., Mehrabadi, M.A., Naeini, E.K., Seo, D., Lim, S.S., Rahmani, A.M., Dutt, N.: Towards smart and efficient health monitoring using edge-enabled situational-

awareness. In: 2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS), pp. 1–4. IEEE, Piscataway (2021)
64. Shahhosseini, S., Seo, D., Kanduri, A., Hu, T., Lim, S.s., Donyanavard, B., Rahmani, A.M., Dutt, N.: Online learning for orchestration of inference in multi-user end-edge-cloud networks. In: ACM Transactions on Embedded Computing Systems (TECS) (2022)
65. Sheng, Z., Mahapatra, C., Leung, V.C., Chen, M., Sahu, P.K.: Energy efficient cooperative computing in mobile wireless sensor networks. IEEE Trans. Cloud Comput. **6**(1), 114–126 (2015)
66. Stites, M.: Observational pain scales in critically ill adults. Crit. Care Nurse **33**(3), 68–78 (2013)
67. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT Press (2018)
68. Teerapittayanon, S., McDanel, B., Kung, H.T.: BranchyNet: fast inference via early exiting from deep neural networks. In: 2016 23rd International Conference on Pattern Recognition (ICPR), pp. 2464–2469 (2016)
69. Teerapittayanon, S., McDanel, B., Kung, H.T.: Distributed deep neural networks over the cloud, the edge and end devices. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), pp. 328–339. IEEE, Piscataway (2017)
70. Tompkins, D.A., Hobelmann, J.G., Compton, P.: Providing chronic pain management in the "fifth vital sign" era: historical and treatment perspectives on a modern-day medical dilemma. Drug Alcohol Depend. **173**, S11–S21 (2017). Prescription Opioids: new perspectives and research on their role in chronic pain management and addiction
71. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems, vol. 30 (2017)
72. Versluis, A., van Luenen, S., Meijer, E., Honkoop, P.J., Pinnock, H., Mohr, D.C., Neves, A.L., Chavannes, N.H., van der Kleij, R.M.: Series: eHealth in primary care. Part 4: addressing the challenges of implementation. Eur. J. Gen. Practice **26**(1), 140–145 (2020)
73. Wang, X., Han, Y., Leung, V.C.M., Niyato, D., Yan, X., Chen, X.: Convergence of edge computing and deep learning: a comprehensive survey. IEEE Commun. Surv. Tutorials **22**(2), 869–904 (2020)
74. Wang, X., Han, Y., Leung, V.C., Niyato, D., Yan, X., Chen, X.: Convergence of edge computing and deep learning: a comprehensive survey. IEEE Commun. Surv. Tutorials **22**(2), 869–904 (2020)
75. Wang, Y., Yao, Q., Kwok, J.T., Ni, L.M.: Generalizing from a few examples: a survey on few-shot learning. ACM Comput. Surv. 53(3), 1–34 (2020)
76. Werner, P., Al-Hamadi, A., Limbrecht-Ecklundt, K., Walter, S., Gruss, S., Traue, H.C.: Automatic pain assessment with facial activity descriptors. IEEE Trans. Affect. Comput. **8**(3), 286–299 (2016)
77. Werner, P., Al-Hamadi, A., Niese, R., Walter, S., Gruss, S., Traue, H.C.: Towards pain monitoring: Facial expression, head pose, a new database, an automatic system and remaining challenges. In: Proceedings of the British Machine Vision Conference, pp. 1–13 (2013)
78. Yick, J., Mukherjee, B., Ghosal, D.: Wireless sensor network survey. Comput. Netw. **52**(12), 2292–2330 (2008)
79. You, C., Huang, K.: Exploiting non-causal CPU-state information for energy-efficient mobile cooperative computing. IEEE Trans. Wirel. Commun. **17**(6), 4104–4117 (2018)
80. Zhang, K., Mao, Y., Leng, S., Zhao, Q., Li, L., Peng, X., Pan, L., Maharjan, S., Zhang, Y.: Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks. IEEE Access **4**, 5896–5907 (2016)

A Survey of Embedded Machine Learning for Smart and Sustainable Healthcare Applications

Sizhe An, Yigit Tuncel, Toygun Basaklar, and Umit Y. Ogras

1 Introduction

Embedded machine learning has recently drawn significant attention due to the fast development of machine learning (ML) and embedded devices. It is an application of artificial intelligence (AI) to make decisions or predictions from the existing data at the edge without explicit programming. The success of embedded ML heavily relies on the recent improvement of computation power since ML algorithms are highly data-intensive. Machines need to launch complex linear algebraic computations, such as matrix and vector operations, to learn the non-trivial relationship between the inputs and outputs. To date, computing clusters using multiple high-frequency central processing units (CPU), graphics processing unit (GPU), and tensor processing unit (TPU) [79] are the most widely used resources to perform such operations. However, when the users want to enjoy the convenience of ML without transmitting their local data, computing clusters are certainly not a good choice due to the prices and large form factor.

An embedded device refers to a small computer system—a combination of computer processors, memory, and input/output devices [91]. Nowadays, people have access to multiple personal devices, such as smartphones, smartwatches, and autonomous cars. Embedded devices such as Raspberry Pi [66], Nvidia Jetson [52], and Arduino [11] are powerful yet affordable due to the recent emergence of hardware. For example, an Nvidia Jetson Nano developer kit [52] with 128-core 4GB memory GPU that can run most ML algorithms only costs less than $100. Embedded machine learning enables the deployment of ML algorithms on edge devices rather than the powerful computational cluster. It allows the end users to

S. An · Y. Tuncel · T. Basaklar · U. Y. Ogras (✉)
University of Wisconsin-Madison, Madison, WI, USA
e-mail: sizhe.an@wisc.edu; tuncel@wisc.edu; basaklar@wisc.edu; uogras@wisc.edu

S. Pasricha, M. Shafique (eds.), *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing*, https://doi.org/10.1007/978-3-031-40677-5_6

perform machine learning directly on devices used in the field, thus leading to numerous novel applications.

Concurrent advances in machine learning and low-power computing areas pave the way for high-impact applications. These applications can attract social attention, promote industrial progress, and improve the quality of life. For example, computer vision (CV) and natural language processing (NLP) are major technical areas that widely apply machine learning since they are attached closely to the industry and commercial market. Specifically, computer vision helps machines understand images and videos, thus generating meaningful information and making decisions. Example applications include image classification, object detection, object tracking, and instance segmentation. The market for computer vision is expected to reach USD 48.6 billion by 2022 [38]. NLP refers to the technology that gives computers to understand text and language similar to human beings [37]. Application of NLP includes speech recognition, sentiment analysis, machine translation, and text summarization. Another popular machine learning application area is recommendation system (RS). Recommendation systems aim to recommend things to the users based on their previous interests and other factors. These systems predict the most likely content that will interest users. Many tech companies such as Google, Amazon, and Netflix rely on recommendation systems to enhance their customers' engagement.

As a particular subset of the previously mentioned ML applications trained solely with big data, embedded machine learning supports obtaining and processing new data locally. Embedded devices are usually equipped with different sensors to measure motion, biopotentials, and temperature. Embedded ML can directly use data from these sensors, thus enabling numerous novel applications. Target applications include smart healthcare, autonomous driving, professional sports, and power/energy management [23, 54, 87]. The rest of the chapter will first overview embedded machine learning frameworks and then offer examples of specific applications using embedded machine learning. This chapter focuses mainly on embedded machine learning applications with data obtained on-device. The concept of embedded machine learning and tinyML will be coherently interspersed and interact with the applications. Finally, we also introduce energy management as a service application since the deployment of ML applications on edge devices is limited by battery capacity.

2 Overview of Embedded Machine Learning Frameworks

Embedded machine learning frameworks typically consist of model training, model compression, and model inference, as illustrated in Fig. 1. Model training is the process of learning the non-trivial patterns or relationships between the inputs and outputs through an intensive search that includes trial and error. It usually needs a vast amount of data points to train to learn the hidden complex relationship between the inputs and outputs instead of memorizing from the existing data. Thus, model

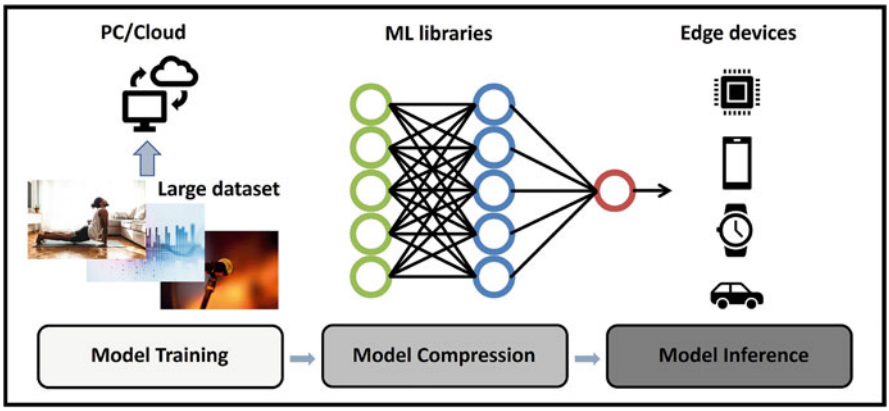


Fig. 1 Overview of embedding machine learning framework

training is usually performed in powerful computation processors, such as cloud servers, workstations, or personal computers (PC), as shown in Fig. 1.

The success of ML frameworks relies heavily on the improvement of computation power since ML algorithms need an intensive amount of data to train. However, users of ML applications do not participate in the model training part. Instead, they run the pre-trained ML models on their devices for inference in different applications. Therefore, it is crucial to ensure that the ML algorithms targeting embedded applications can run on edge devices with limited computational power and memory. To this end, popular ML libraries such as PyTorch [60] and TensorFlow [1] have been making significant efforts in compressing the size and accelerating the inference time of the model on embedded devices. For example, PyTorch Mobile [61] and TFLite [80] are the corresponding lightweight version of PyTorch and TensorFlow. Many recent studies take advantage of these lightweight ML frameworks that target embedded ML [26, 94, 95]. The lightweight version of TensorFlow, TFLite, can reduce the model size up to 75% with a minimal accuracy loss. For instance, a ResNet101 model, after optimized by TFLite, is only 44.9 MB, compared to 178.3 MB on TensorFlow with only 0.2% accuracy loss. Consequently, the tinyML concept is introduced with the fast development of lightweight ML libraries. TinyML refers to ML capable of performing on-device sensor data analytics at ultra-low power, thus enabling a variety of always-on applications and targeting battery-operated devices [82]. The middle part of Fig. 1 shows that model compression is bridging the model training and model inference using ML libraries.

Model inference refers to the process of inferring the most likely output of given inputs from the previously learned model. Hence, it does not require intensive computational power and can be easily deployed on edge devices. In the embedded machine learning flows, the model inference is the central part that runs on users' edge devices. For instance, users can access ML applications such as tracking

their vital signs using a smartphone or smartwatch [9] and setting up self-driving functions in their car. A recent study [94] designed and implemented convolution neural networks (CNNs) on smartphones using TFLite to estimate human activities in real time. Similarly, real-time human pose estimations for a single person and multi-person on mobile devices are proposed in [26] and [95], respectively. The right part of Fig. 1 shows that users can use the compressed ML algorithm for their customized applications using their edge devices.

3 Embedded Machine Learning Applications for Healthcare

The aging population has been becoming a serious concern all over the world. The consequent rise in health-related issues has drawn significant research attention from the industry and academic community. Technology companies have continuously increased their R&D expenditure for wearable devices that can be used for mobile activity and health monitoring. For instance, Apple utilizes its popular consumer-facing products, such as Apple Watch, to provide health-related features accessible on its watch to bridge wearables and clinical tools used in medical research [9]. In 2021, Google acquired the wearable giant Fitbit (smartwatch) to participate in the AI-enabled healthcare race among top-pitch technical companies [32]. Embedded machine learning enables various healthcare-related applications by feeding multi-modal sensing data obtained from humans into machine learning algorithms on devices.

Remote activity and healthy monitoring applications can provide valuable insights [12]. Hence, they can improve the quality of life in many healthcare applications, including but not limited to human activity recognition [4, 19], gait monitoring [5, 46], human pose estimation [2, 3, 92], and freezing of gait (FoG) [24, 55, 69]. For example, advanced ML algorithms can locally analyze the motion and physiological data from wearable sensors. This capability can enable many real-time applications such as irregular rhythm notification, early warning signs, and fall detection. These applications are also the first step toward diagnosis, prognosis, and rehabilitation of movement disorders similar to Parkinson's disease (PD) and stroke. The rest of this section discusses three illustrative examples as cases studies and summarizes the recent work in those areas.

3.1 Freezing-of-Gait Identification in PD Patients

Parkinson's disease (PD) is one of the most common age-related neurodegenerative diseases. It causes muscular rigidity, tremor, bradykinesia, slowness in movement, and postural instability [27, 45, 49, 83]. More than 50% of the PD patients develop freezing of gait (FoG) [44] in their advanced stages of the disease. Freezing of gait is a brief absence of the ability to walk despite the intention of moving the feet [59].

FoG episodes may include trembling of knees, short shuffling steps, or complete akinesia [71] and overall increase the risk of falling and deteriorate the patient's quality of life.

Clinical studies suggest that external stimuli, such as auditory, visual, or tactile cues, help patients to exit the FoG state and resume walking [59]. FoG identification is a challenging task since FoG episodes are rare events. Furthermore, clinical settings cannot provide the actual frequency of occurrence of FoG episodes. For example, FoG episodes mainly occur during turning, walking through doorways, or dual tasking, and 90% of them last less than 20 s [44]. Specific experimental sessions are designed to simulate the daily-life activities in clinical settings, such as walking and turning while dual tasking [44]. However, this practice is a challenging proposition due to fundamental limitations. On the one hand, the duration of these sessions in a clinical setting and the number of visits per patient are limited. On the other hand, the frequency of FoG occurrence and the duration of FoG episodes are short. Hence, the probability of observing FoG episodes during these simulations can be small.

FoG-related problems can be avoided by systems that can identify FoG episodes and provide an appropriate cueing mechanism such as audio. FoG identification can be divided into two subclasses: FoG detection and FoG prediction. FoG detection implies classifying FoG episodes *after patients start to experience them*. It has been heavily studied for over a decade. More than 50 studies related to FoG detection have been published by late 2019 [58]. In contrast, FoG prediction implies classifying FoG episodes before the occurrence of an FoG episode. The main goal of the FoG prediction studies is to *predict potential FoG episodes and prevent their onset* by providing preemptive cueing. Despite this promising potential, there are very few studies related to FoG prediction compared to FoG detection [58, 59, 73].

Both FoG detection and prediction studies use various wearable sensors placed on different parts of the patient's body. These sensors collect motion data using inertial measurement units, plantar pressure systems, and electromyography [44, 59]. Various machine learning approaches are utilized for FoG identification, including random forests, support vector machines (SVMs), nearest neighbor algorithms, and deep neural networks (DNNs) [57]. Classification algorithms, such as decision trees, SVMs, and *k*-nearest neighbor, require handcrafted spectral and statistical features extracted from the motion data of the PD patients, which needs substantial domain knowledge [57]. However, FoG identification approaches should require minimal preprocessing and manual effort to facilitate easy deployment on edge-AI devices. Approaches that employ DNNs do not require domain knowledge. They can directly utilize raw sensor data to identify FoG episodes. For example, convolutional neural networks (CNNs) are adopted widely among with long short-term memory (LSTM) networks to detect and predict FoG episodes [13, 53, 57, 73, 83]. Table 1 summarizes the recent FoG identification studies published since 2018. None of the above studies have embedded ML framework in an edge-AI device. These approaches require powerful computing resources that are hard to integrate on an edge device. Therefore, there is a critical need for lightweight FoG identification approaches

Table 1 A summary of recent FoG identification studies

	Sensor type and location	FoG identification approach
Sama et al. [69]	IMU placed on the waist	k-NN, random forest, logistic regression, Naïve Bayes, multilayer perceptron (MLP), SVM
Camps et al. [24]	IMU placed on the waist	CNN
Oung et al. [55]	3 accelerometers placed on the left shank, left thigh, and lower back	Probabilistic NN, SVM
Li et al. [41]	Accelerometer placed on the lower back	Mini-batch k-means clustering
Mikos et al. [47]	2 accelerometers placed on each ankle	MLP
Rad et al. [63]	3 accelerometers placed on the left shank, left thigh, and lower back	Denoising autoencoder
Handojoseno et al. [33]	EEG electrodes placed on the head	DNN
Torvi et al. [83]	3 accelerometers placed on the left shank, left thigh, and lower back	LSTM
El-Attar et al. [29]	Accelerometer placed on the left shank	DNN
Naghavi and Wade [49]	3 accelerometers placed on the left shank, left thigh, and lower back	Statistical analysis based on Kruskal–Wallis test
Naghavi et al. [50]	2 accelerometers placed on each ankle	k-NN, SVM, decision tree, MLP along with classifier bagging and synthetic minority over-sampling methods
Arami et al. [10]	3 accelerometers placed on the left shank, left thigh, and lower back	SVM
Demrozi et al. [28]	3 accelerometers placed on the left shank, left thigh, and lower back	k-NN
Reches et al. [67]	3 accelerometers placed on the left ankle, right ankle, and lower back	SVM
Shi et al. [75]	3 accelerometers placed on the left ankle, right ankle, and neck	CNN
Li et al. [42]	3 accelerometers placed on the left shank, left thigh, and lower back	CNN + LSTM
Sigcha et al. [77]	IMU placed on the waist	CNN + LSTM
Mancini et al. [44]	8 IMUs placed on the shins, feet, wrists, sternum, and lower back	Correlation and thresholding
Bikias et al. [20]	IMU placed on the wrist	CNN
Borzi et al. [21]	2 IMUs placed on the shins	k-NN, SVM, linear discriminant analysis, logistic regression

that leverage the wearable sensor data with minimal preprocessing of the data and activate an appropriate cueing mechanism locally on the edge device.

3.2 Human Activity Recognition

There has been growing interest in human activity recognition (HAR) due to its health monitoring and patient rehabilitation applications [4, 65, 76, 90]. Inertial measurement units (IMUs) are used to capture the body and joint movements to estimate or predict human activity. The recognized activities with time stamps are valuable insights for health monitoring and rehabilitation. The prevalence of low-cost motion sensors and embedded machine learning algorithms make it possible to perform human activity recognition on-device [18].

3.2.1 Processing Pipeline

The majority of HAR methods employ smartphones due to their popularity and easy access to integrated accelerometers and gyroscope sensors [6, 76, 90]. More recent work started using wearable devices for this purpose due to significant power consumption and form-factor benefits. A typical HAR framework consists of data preprocessing, feature extraction, and classifying algorithms, as shown in Fig. 2. Inertial measurement units, which typically include accelerometers and gyroscopes, are attached to different human body parts. They usually provide three-axis acceleration and angular velocity. If the activities are simple enough, a significant number of approaches use only the acceleration data since the gyroscope sensors have relatively larger power consumption [97]. The first step is to preprocess the raw sensor data to reduce measurement noise and construct activity windows. Then, the data in each activity window are used to produce features, such as the body acceleration and signal statistics (e.g., min, max, mean). Finally, these features are used by ML algorithms to recognize activities, such as standing, sitting, lying, walking, and jogging.

A range of methods is used during the preprocessing step. The most common preprocessing techniques include downsampling, low-pass filtering, and segmentation. The inertial sensors often sample at a high sampling rate (>50 Hz). However, most human activities are only at a few Hz range [18], making the sampled data redundant. Median and mean filters are two mainstream filters used for downsampling. Similarly, the sensor data, especially accelerometer data, are typically noisy. Therefore, most preprocessing techniques incorporate low-pass filtering and smoothing. Most ML algorithms require the inputs to have a fixed length. Therefore, preprocessing steps typically involve segmentation algorithms that divide the data into possibly overlapping windows. For example, segmentation algorithms can divide a long period (in the order of hours) of time series data into multiple fixed-length (e.g., one–ten seconds) windows. If the number of data samples in each window is uniform, the ML algorithms, such as DNNs, can conveniently process them.

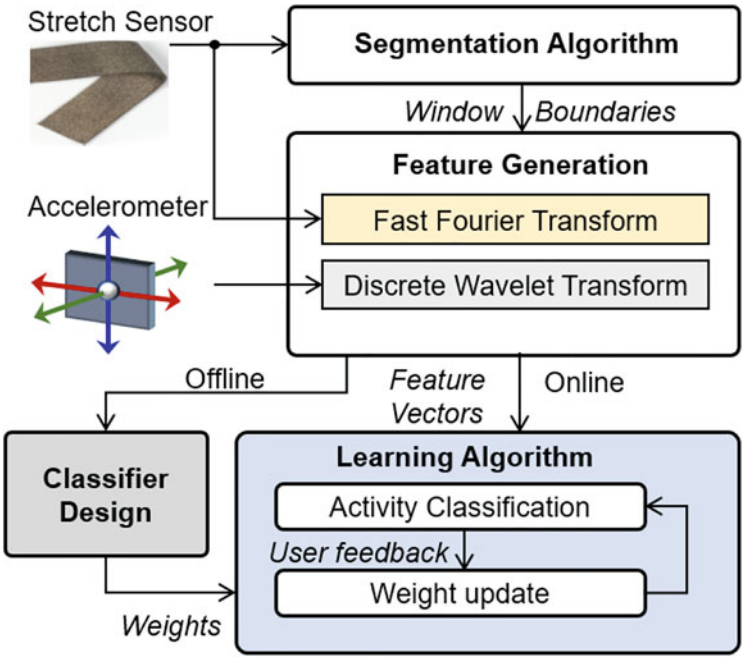


Fig. 2 Overview of human activity recognition. The figure is partially modified from [18]

3.2.2 Commonly Used ML Algorithms

Most of the HAR techniques employ supervised learning algorithms. Examples of supervised learning algorithms suitable for HAR include support vector machine (SVM), random forests, decision trees, k-nearest neighbors (k-NN), and neural network (NN). These algorithms take the labeled data to train the classifier, as outlined below:

- *Support vector machine*: SVM techniques try to find a hyperplane in high-dimensional space that separates two output classes [34]. If they cannot find the separating hyperplane in a lower dimension, it keeps mapping the data into higher dimensions until the separating hyperplane is found. HAR problem is essentially a multi-class classifying problem. Multiple classifiers need to be performed to apply SVM to HAR for differentiating more than two output classes since SVM is a binary classifier.
- *Random forests and decision trees*: Decision tree classifiers are commonly used for classification problems since it is intuitive and explainable. They use a series of rules to make decisions, just like how humans make decisions [34]. Decision trees take the dataset features to create binary questions and continually separate the dataset until all data samples are isolated to different classes. An ensemble

of tree-structured classifiers is employed for random forests. The most predicted classes among all trees are chosen as the final predicted class.

- *k-Nearest Neighbors*: This approach is one of the most traditional and popular techniques in classifying problems [34]. K-NN first computes k-nearest neighbors in the training set. Then, it chooses the most common class among k neighbors. That class is then the final estimated class. K-NN technique requires all training data to be stored locally. Many of the HAR techniques used k-NN for the offline training [70].
- *Neural networks*: Neural networks are widely used in classification and regression problems. Multilayer perceptron (MLP) classifier is one of the primary neural networks used for human activity recognition. It consists of three layers at least: one input layer, one hidden layer, and one output layer. Each layer of the MLP has multiple neurons/nodes with non-linear activation functions. In the HAR context, the number of hidden neurons in the hidden layers is crucial for obtaining good accuracy while retaining low computational complexity. The number of neurons in the output layer corresponds to the output classes. Convolution neural networks (CNNs) are popular for processing 2D images. Convolutional kernels can convolve with the input image layers by layers to extract the helpful feature maps instead of choosing the features manually in other techniques [93]. Researchers recently applied CNN in HAR by reshaping the 1D HAR features to a 2D feature map input [16].

3.2.3 Offline vs. Online Learning

Currently, there are two mainstream HAR training paradigms: offline training and online training. The training is performed on dedicated computing resources such as power CPU and GPU for offline training. Machine learning algorithms require a large amount of data to capture different patterns' behavior and learn how to classify them. Thus, dedicated computing resources are used since they can process a vast amount of data. The offline pre-trained models then are deployed to edge devices, for example, smartphones and smart wearables, to perform the inference for new users. Offline training is the fundamentals of all supervised training algorithms. The downside of this approach is that the performance of inferring on new users that have not appeared in the training data is inevitably worse than inferring on the trained users. Online training tackles this issue by continually training with new user data on the edge devices. The pre-trained models are also deployed to edge devices, but the new users' data obtained on the field are fed into the machine learning algorithm running on the edge device to train online. Since the amount of data is small compared to the offline training data, the edge devices can train the models in real time. For HAR on-device, the neural network is the most popular method since it supports online training [18].

3.3 Human Pose Estimation

Human pose estimation aims to detect and track key joints, such as wrists, elbows, and knees. It has rapidly growing applications areas, including rehabilitation, professional sports, and autonomous driving [23, 54, 87]. For instance, one of the leading causes of autonomous car accidents is "robotic" driving, where the self-driver makes a legal but unexpected stop and causes other drivers to crash into it [54]. Real-time human pose estimation can help computers understand and predict human states, thus leading to more natural driving. Likewise, remote rehabilitation applications, which are currently not feasible, can be enabled by human pose estimation.

Human pose estimation can be performed by processing image, video, LiDAR (light detection and ranging), inertial sensors (IMUs), or mmWave radar data. RGB image and video frames are the most common input types since they offer accurate real-world representations with true color. However, the RGB frame quality depends heavily on the environmental setting, such as light conditions and visibility. Alternatively, the LiDAR point cloud obtained by laser scanning overcomes these challenges. However, it has high-cost and significant processing requirements, making them unsuitable for indoor applications such as rehabilitation. mmWave radar can generate high-resolution 3D point clouds while maintaining low-cost, price, and power advantages. Inertial sensors (IMUs) can also reconstruct human pose using the sensing accelerations and gyroscopes [88, 89]. This section discusses these three broad approaches and recent literature.

3.3.1 Human Pose Estimation Using RGB Camera

In the computer vision field, human pose estimation has drawn attention after a seminal study in 2005 [64]. This study presents a framework to detect ten distinct body parts using rectangular templates from RGB images. He et al. [35] propose Mask R-CNN, which can reconstruct skeleton from RGB images using K masks by leveraging ResNet neural network architecture. It first detects K different key points and then connects them. Mask R-CNN has become popular due to its fast processing time and accurate estimation. Similarly, Cao et al. proposed OpenPose [25], a real-time human pose estimation technique that can detect human body, face, and foot key points together for the first time. OpenPose has become one of the popular benchmarks due to its decent performance and the easy-to-use open-source package. Besides the RGB video-based approach, Microsoft Kinect and Kinect V2 [74] provide depth cameras to extract the human joints representation. Both Kinect and Kinect V2 use an RGB camera and a depth sensor consisting of an infra-red camera and projector as sensing units to capture the information. The Kinect family has become one of the popular methods to obtain the ground truth label for training due to its convenience, low cost, and nice performance [7, 72, 96].

From an RGB image containing a person, the human pose estimation model typically consists of the cropping bounding box, extracting features, and predicting the joint coordinates. The first step is to crop the bounding box containing the human. The area of a person can be only 1/5 of the image or even smaller. For the human pose estimation task, the region of interest (ROI) is only the area related to humans. Hence, an efficient human bounding box detection algorithm is crucial. After obtaining the human bounding box, the next step is to extract useful features from the area. Most of the techniques use deep CNNs since many of them have been proved to be suitable feature extractors in the image processing field [35]. Finally, the model needs to output the human joints coordinates. In estimating the joint coordinates, heatmap-based and regression-based are two mainstream methods.

Heatmap-based models learn each joint point's position through the Gaussian distribution graphs. The method first renders Gaussian probability distribution heatmaps for every joint point and then applies argmax or soft-argmax operation to the heatmaps, thus obtaining the final estimation results. Since the maximum value of every heatmap corresponds to a joint's coordinates, the resolution of the output heatmap needs to be relatively high (64×64 usually). Thus, this method's computation and memory overhead are high since multiple high-resolution Gaussian heatmaps need to be rendered (one output heatmap for each joint).

Regression-based models represent another alternative that is simpler and more intuitive. It directly learns the joints' coordinates values using L1 or L2 loss. Since the regression-based method does not require rendering the heatmap and maintaining the high resolution, the output feature map can be small compared to the heatmap-based model. Thus, the computation and memory requirements of the regression-based model are significantly lower than the heatmap-based model. For instance, using Resnet-50, the floating-point operation per second (FLOPs) of the regression-based model is 1/20000 of the heatmap-based model [43]. This result shows that the regression model is friendly to the edge devices. Regression-based methods are widely applied in industry [43] since it is computationally efficient and straightforward. However, the heatmap-based method is generally more robust for occlusion and blur. In addition, the heatmap-based model has better explainability than the regression-based model. Recently, researchers have started to combine two methods to keep advantages of both of them [43].

3.3.2 Human Pose Estimation Using mmWave Radar

The human pose can also be reconstructed from mmWave signals. Compared to the RGB image source, mmWave signals preserve user privacy well since the mmWave signal does not reveal salient and rich information such as true-color images. At the same time, the sparse input source makes human pose estimation a more challenging task. Almost all mmWave human pose estimation methods use a regression-based model. In 2018, researchers proposed RF-Pose3D [96], a technique that reconstructs up to 14 body parts, including the head, neck, shoulders, elbows, wrists, hip, knees, and feet. This work first uses 12 camera nodes to record RGB-based video and

then obtain label key points from OpenPose. At the same time, radar signals at a few GHz are used to generate the RF heatmap. They then train a region proposal network (RPN) to zoom in on RF data and a CNN with ResNet architecture to extract the 3D skeleton from the region of interest. For keypoint localization, the average errors in the x, y, z axes are 4.2, 4.0, and 4.9 cm, respectively.

Besides being limited to 14 joints, this work does not leverage the mmWave radar's ability to obtain a high-quality point cloud. Thus, it requires a much more complex NN architecture with high computation cost. Moreover, multiple cameras and bulky radar signal generating systems hinder the practicality of the approach. The most recent mmWave radar-based pose estimation techniques use point cloud representation from the commercial radar device Texas Instrument (TI) xWR1x43 [81]. Sengupta et al. [72] propose mmPose, a human pose estimation technique that constructs the skeleton by using mmWave point cloud and a forked-CNN architecture. They use two radar devices and sum up the point values in the feature map level to overcome the sparse representation of the point cloud. An et al. [3] present a meta-learning and frame aggregation framework to help mmWave-based human pose estimation model converge faster for unseen scenarios. Xue et al. [92] propose the mmMesh technique to construct human mesh using mmWave point cloud.

Finally, another recent study proposes a mmWave-based assistive rehabilitation system (MARS) [2] using human pose estimation. It sorts the mmWave point cloud and performs matrix transformations before feeding them to a CNN model. MARS can reconstruct up to 19 human joints and human skeleton in 3D space using mmWave radar without raising privacy concerns and requiring strict lighting settings. Moreover, MARS provides the users with 19 joints velocity estimations, four critical angle estimations, and ten commonly used rehabilitation posture correction feedback. It incorporates point cloud preprocessing, a CNN that outputs joint positions, and rehabilitation movement feedback to the user. It first maps the 5D time series mmWave point cloud to a 5-channel feature map and then outputs 3D joint positions. It finally provides joint velocity, angle estimations, and posture correction feedback. The overview of MARS is shown in Fig. 3. An example of human pose estimation using mmWave radar point cloud is shown in Fig. 4.

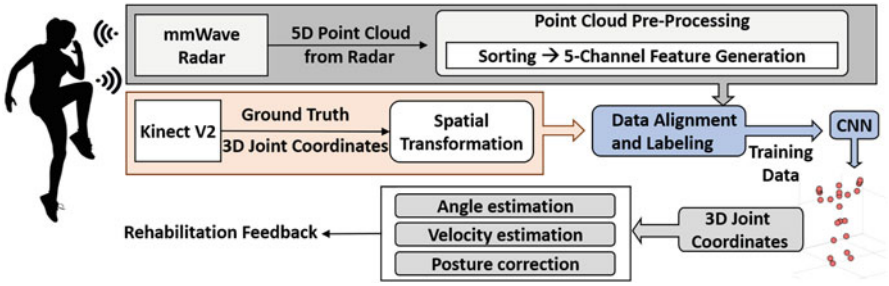


Fig. 3 Overview of human pose estimation using mmWave point cloud and its downstream healthcare-related tasks. The figure is partially modified from [2]

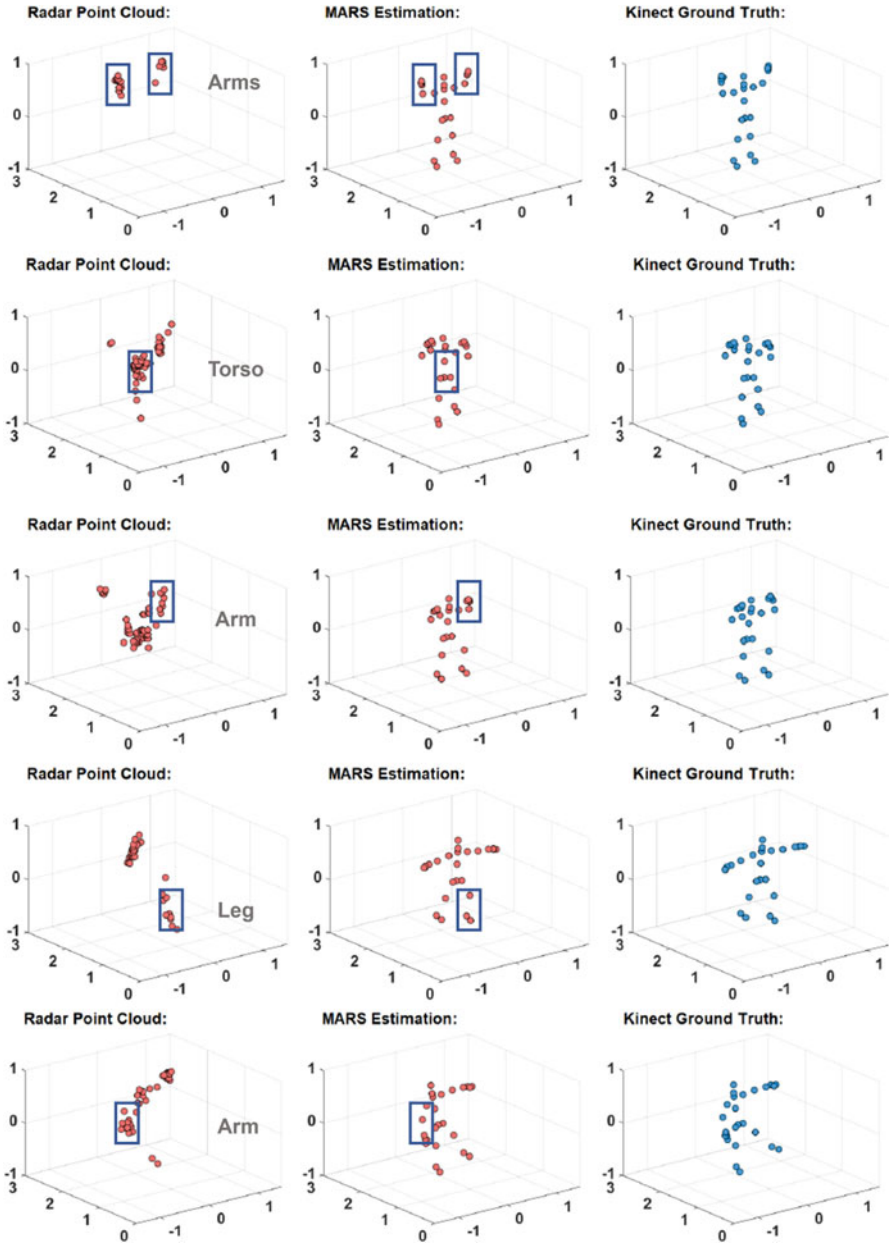


Fig. 4 Example of human pose estimation using mmWave radar point cloud. From left to right, it shows radar point cloud, MARS estimation, and the ground truth [2]. Some of the human parts are highlighted by the bounding boxes in the figure. The figure is partially modified from [2]

Model inference of MARS takes only 64 μs and consumes 442 μJ energy on the Nvidia Jetson Xavier-NX board. These results show the practicality of the proposed technique running real time on low-power edge devices. The accuracy of human pose estimation using mmWave is comparable to that of an RGB image. However, the explainability of the model and solution for free-form human pose estimation is still challenging. There are several future challenges for mmWave human pose estimation to be widely applied in real-life applications.

3.3.3 Human Pose Estimation Using Inertial Sensors

Besides cameras and radars, wearables such as inertial sensors (IMUs) also play an essential role in human pose estimation. IMU-based human pose estimation is relatively robust to different environmental settings since sensing is not interfered with by light conditions or visibility. Thus, it is more practical for occlusions or baggy clothing scenarios. In addition, the explainability of IMUs-based human pose estimation is pretty good since every IMU is placed in a specific position of a person. As one of the earliest studies in this field, [68] estimate human pose using 17 IMUs, and a Kalman filter is employed for all the measurements. It comprehensively defined 17 IMUs on a person, thus achieving accurate human pose estimation. However, the large number of IMUs requires long setup times and makes it uncomfortable for users. Marcard et al. proposed sparse inertial poser (SIP) [88]: automatic 3D human pose estimation from sparse IMUs. This work provides a new method to estimate the human pose using only six IMUs. By exploiting a statistical body model and jointly optimizing posture over continuous time frames to fit both orientation and acceleration data, SIP achieves positional errors of 3.9 cm. A follow-up work [89] combines IMUs and a moving camera to estimate multiple human poses in challenging outdoor scenes robustly.

In summary, human pose estimation can be performed using different input sources. Table 2 compares different input sources in terms of accuracy, privacy concern, price, and the anti-interference ability. Lightweight embedded machine learning algorithms enable running human pose estimation models on edge devices. In real-life applications, it is crucial to choose proper input sources of the human pose estimation model according to different requirements such as accuracy, privacy, and robustness.

Table 2 Comparison between different input sources for human pose estimation. Anti-interference here represents the robustness of the algorithm. Specifically, different input sources are affected by environmental conditions such as light and smoke to varying degrees

	Data form	Accuracy	Privacy	Price	Anti-interference
Camera	Image/video	⋆⋆⋆	⋆	⋆⋆	⋆
LiDAR	Point cloud	⋆⋆⋆	⋆⋆⋆	⋆	⋆⋆⋆
Radar	Point cloud/heatmap	⋆⋆	⋆⋆⋆	⋆⋆	⋆⋆⋆
IMUs	Accelerations	⋆	⋆⋆⋆	⋆⋆⋆	⋆⋆⋆

4 Energy Management

The commonly used edge devices include smartphones, smartwatches, and other wearable devices. Small form-factor devices, in particular, have severely limited battery capacity due to form-factor requirements, such as small size, lightweight, and flexibility. For example, the Oura Ring 3 incorporates a 22 mAh–3.7 V battery and advertises a battery life of 4 to 7 days [56]. Despite this limitation, these devices collect significant amounts of data to enable sophisticated health monitoring applications discussed in previous sections. The energy consumption soars if they transmit these data to a mobile device or to the cloud since wireless streaming of information is prohibitively power-consuming. Consequently, the recharge interval is short, which causes the users to stop using these devices. Therefore, significant research effort focuses on reducing the dependency of wearable devices on batteries.

Reducing the dependency on batteries critically depends on the developments in three main research areas: (1) energy harvesting (EH), (2) energy management, and (3) low-power design:

(1) **Energy harvesting** refers to techniques that generate power from ambient resources, such as light and motion. It provides a complementary energy source to batteries.
(2) **Energy management** encompasses the techniques that aim to optimally utilize the available energy from batteries and energy harvesting sources. It regulates the energy consumption to maximize the user experience within the constraints set by the energy source and the device.
(3) **Low-power design** refers to a broad class of design styles that aim to minimize the power consumption of the devices, while they meet the processing requirements. Popular techniques include clock and power gating, leakage power minimization, and heterogeneous computing, such as using domain-specific hardware accelerators to boost energy efficiency.

This chapter overviews energy management, a key enabler, and service application for edge devices running target ML applications. To this end, it first summarizes the wearable energy harvesting modalities and provides a general idea of the energy budget for wearable devices. Then, it presents an overview of optimal energy management techniques. We leave the low-power design practices and optimizations on energy consumption out of this chapter. We refer the interested readers to other surveys and books on low-power design techniques [15, 62].

4.1 Energy Sources and Budget

Wearable EH techniques generate usable electrical energy from various sources in a user's environment while conforming to the physical and comfort constraints associated with the wearable form factor [48]. The most common energy sources

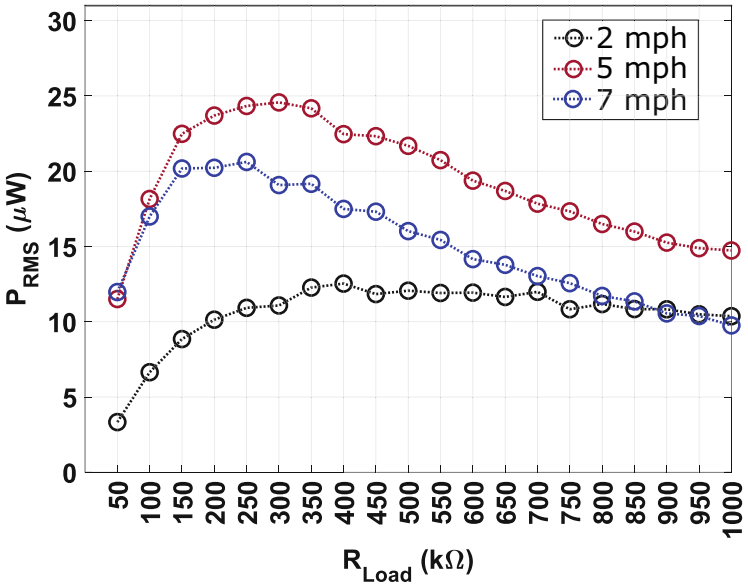


Fig. 5 Power vs. load resistance curve of a wearable piezoelectric energy harvester at different gait speeds

are light, motion, electromagnetic waves, and heat [78]. Ambient light has the highest potential for wearable EH devices. For instance, with an 8.1 cm^2 flexible PV cell, ambient light EH offers a capacity of over 1 mW outdoors (5000 lux) and close to 100 μW indoors (500 lux) [39]. Similarly, radio-frequency (RF) EH can harvest 10 μW with an 18.4 cm^2 flexible antenna with a signal strength of −10 dBm at 915 MHz [51]. Body-heat EH has power levels of about 3 μW with a 1 cm^2 flexible harvester at an ambient temperature of 15 °C (i.e., a temperature difference of 22 °C) [36]. Human-motion EH is particularly interesting for wearable applications because the energy is available on demand. For example, energy due to human activity is at hand when required by an activity-monitoring application. In addition, human-motion EH can harvest about 25 μW with a 23.8 cm^2 piezoelectric transducer, while the wearer is jogging (i.e., 5 mph gait speed) [84, 85], as illustrated in Fig. 5.

4.2 Optimal Energy Management

We can enable a long-term recharge-free operation if the edge devices have an energy-neutral operation. Energy neutrality means that the energy consumed over a time period (e.g., one day) is less than or equal to the energy produced during the same period. When the device has a battery, the energy stored in the battery can

temporarily power the device if the harvested energy is insufficient. However, the battery will be recharged back to its original level if the energy-neutral operation is guaranteed. Hence, the energy-neutral operation can automate battery charging through harvested energy such that the battery level is restored at the end of each day. In the absence of a battery, this system reduces to intermittent computing where the operating halts when no energy is available. In summary, optimal energy management techniques can maximize the device utility (e.g., the amount of time it remains active) under the available energy budget, whether from a battery or energy harvesting source.

Achieving energy-neutral operation is challenging due to the conflict between the uncertainty in harvested energy and the target application's quality-of-service (QoS) requirements. The application performance and utilization of the device can diminish when the harvested energy is limited [30]. For example, consider a wearable health application where the target device must collect the vital signals and process them locally to detect abnormalities. On the one hand, the device needs a steady and sufficient amount of energy to perform its intended operation, e.g., analyzing the collected signals within a deadline. On the other hand, the harvested energy may fluctuate widely and even vanish entirely during the same period. Therefore, the limited and highly varying nature of the harvested energy necessitates deliberate planning and management.

Energy management algorithms use the available energy judiciously to maximize the application performance while minimizing manual recharge interventions to achieve energy-neutral operation [86]. These algorithms should satisfy the following conditions to be deployed on a wearable resource-constrained device:

- Incurring low execution time and power consumption overhead
- Having a small memory footprint
- Being responsive to the changes in the environment
- Learning to adopt such changes

Kansal et al. [40] present the general framework of energy-neutral operation for energy harvesting devices. The authors propose a linear programming approach to maximize the duty cycle of a sensor node and a lightweight heuristic to help solve the linear programming with ease. Similarly, the work in [22] proposes a long-term energy management algorithm, referred to as long-term ENO, which aims to achieve energy neutrality for one year or more. As complementary to this work, going with a more fundamental control-theory approach, Geissdoerfer et al. [31] propose a feedback controller to achieve long-term ENO. To account for the application requirements when deciding the duty cycle of the nodes, Bhat et al. use a generalized utility function that defines the application characteristics [17]. They present a lightweight framework based on the closed-form solution of the optimization problem that maximizes the utility while maintaining energy-neutral operation. Although these are essential studies in this field, none of them consider user activity patterns, hence the stochastic nature of energy harvesting, which is at the core of wearable energy harvesting techniques.

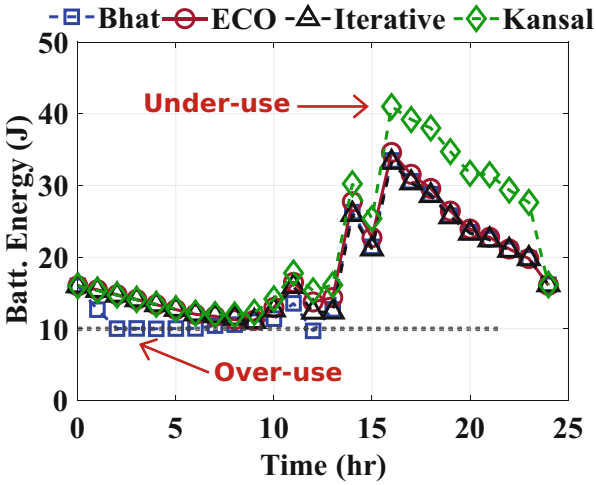


Fig. 6 Comparison of ECO to other approaches and an optimal iterative algorithm

A recently proposed energy management framework, ECO, is tailored for wearable use cases by incorporating user activity and energy harvesting uncertainty into energy management [86]. The ECO framework maximizes a utility function under specific battery energy constraints. The utility function can model any arbitrary metric, such as device throughput and classification accuracy. The framework takes the initial battery energy at the beginning of each day and the expected energy harvesting profile for a finite horizon (e.g., 24 h) as inputs. The expected energy harvesting profile is obtained from a novel EH forecasting model, which considers user patterns. At the beginning of the day, using the energy harvesting profile, ECO first finds the energy that the device can consume during each hour that maximizes the utility function under the battery energy constraints. As the day progresses, this solution is not optimum due to the variations in harvested energy. Using the actual harvested energy, ECO corrects the initial allocations using a lightweight runtime optimization algorithm after an hour. As a result, the ECO framework adapts to the deviations from the expected EH values with negligible runtime overhead. Figure 6 illustrates how ECO addresses the over-utilization and under-utilization of the energy seen in two prior approaches, which result in higher application utility. Moreover, measurements on a wearable device prototype show that ECO has 1000× smaller energy overhead than iterative optimal approaches with a negligible loss in utility.

ECO and other prior work are highly dependent upon the accuracy of the energy forecasts. They can compensate for deviations in user patterns *after the fact*, which causes a deviation from the optimal trajectory. As a remedy, reinforcement learning (RL)-based resource management algorithms are employed for energy management. RL-based approaches benefit from not relying on forecasts of the harvested energy, in contrast to the prediction-based techniques presented above.

These techniques implicitly learn user patterns, and they can proactively perturb the allocations before a significant deviation in usage pattern happens. RLMan is a recent prediction-free energy management approach based on RL [8]. It aims to maximize packet generation rate while avoiding power failures. tinyMAN is another RL-based approach that takes the battery level and the previous harvested energy values as inputs (states). It maximizes the utility of the device by judiciously allocating the harvested energy throughout the day (action). [14]. Over time, by interacting with the environment, the tinyMAN agent learns to manage the available energy on the device according to the harvested energy.

In conclusion, energy management remains a fundamental research field essential for the success of the embedded AI field. Energy management techniques must run as background service applications to ensure the successful operation of embedded AI applications under the available energy budget. Significant challenges include developing low-overhead techniques that maximize energy efficiency under uncertainty and scarce energy resources. Developing novel energy forecasting models is an important research direction for prediction-based approaches. Similarly, efficient runtime learning of user patterns is critical for prediction-free approaches. Finally, the developments in this field can transfer to other resource allocation problems in many different areas, including telecommunications to space-based systems.

5 Conclusions

Embedded machine learning enables numerous novel applications since low-power edge devices allow cutting-edge machine learning algorithms to obtain data from multiple sensors and run locally. This chapter overviewed the opportunities and challenges in the embedded machine learning applications context. It presented a survey of edge-AI application use cases for embedded machine learning. First, it overviewed embedded machine learning frameworks, consisting of model training, model compression, and model inference. Then, it presented several edge-AI applications for healthcare, such as freezing-of-gait identification for Parkinson's disease patients, human activity recognition, and human pose estimation. Finally, we discussed energy management as a fundamental enabler for wearable devices since battery shortage is one of the leading factors that limit embedded machine learning on wearable devices. Lightweight machine algorithms for these high-impact applications and other novel applications offer unique research opportunities.

References

1. Abadi, M., et al.: TensorFlow: large-scale machine learning on heterogeneous systems (2015). Software available from https://tensorflow.org

2. An, S., Ogras, U.Y.: MARS: mmWave-based assistive rehabilitation system for smart healthcare. ACM Trans. Embed. Comput. Syst. **20**(5s), 1–22 (2021)
3. An, S., Ogras, U.Y.: Fast and scalable human pose estimation using mmWave point cloud (2022). Preprint. arXiv:2205.00097
4. An, S., Bhat, G., Gumussoy, S., Ogras, U.: Transfer learning for human activity recognition using representational analysis of neural networks (2020). Preprint. arXiv:2012.04479
5. An, S., Tuncel, Y., Basaklar, T., Krishnakumar, G.K., Bhat, G., Ogras, U.Y.: Mgait: model-based gait analysis using wearable bend and inertial sensors. ACM Trans. Internet Things **3**(1), 1–24 (2021)
6. Anguita, D., Ghio, A., Oneto, L., Parra, F.X.L., Ortiz, J.L.R.: Energy efficient smartphone-based activity recognition using fixed-point arithmetic. J. Univ. Comput. Sci. **19**(9), 1295–1314 (2013)
7. Antunes, J., Bernardino, A., Smailagic, A., Siewiorek, D.P.: AHA-3D: a labelled dataset for senior fitness exercise recognition and segmentation from 3D skeletal data. In: Prof. of the British Machine Vision Conference (BMVC), p. 332 (2018)
8. Aoudia, F.A., Gautier, M., Berder, O.: RLMan: an energy manager based on reinforcement learning for energy harvesting wireless sensor networks. IEEE Trans. Green Commun. Netw. **2**(2), 408–417 (2018)
9. Apple: Apple Watch. Helping your patients identify early warning signs. https://www.apple.com/healthcare/apple-watch/ (2021). Accessed 8 Jul 2021
10. Arami, A., Poulakakis-Daktylidis, A., Tai, Y.F., Burdet, E.: Prediction of gait freezing in Parkinsonian patients: a binary classification augmented with time series prediction. IEEE Trans. Neural Syst. Rehabil. Eng. **27**(9), 1909–1919 (2019)
11. Arduino: Arduino. https://www.arduino.cc/ (2021). Accessed 8 Jul 2021
12. Basaklar, T., Tuncel, Y., An, S., Ogras, U.: Wearable devices and low-power design for smart health applications: challenges and opportunities. In: 2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), pp. 1–1. IEEE, Piscataway (2021)
13. Basaklar, T., Tuncel, Y., Ogras, U.Y.: Subject-independent freezing of gait (FoG) prediction in Parkinson's disease patients. In: 2021 IEEE Biomedical Circuits and Systems Conference (BioCAS), pp. 1–6. IEEE, Piscataway (2021)
14. Basaklar, T., Tuncel, Y., Ogras, U.Y.: tinyMAN: lightweight energy manager using reinforcement learning for energy harvesting wearable IoT devices (2022). Preprint. arXiv:2202.09297
15. Bellaouar, A., Elmasry, M.: Low-power digital VLSI design: circuits and systems. Springer Science & Business Media (2012)
16. Bevilacqua, A., MacDonald, K., Rangarej, A., Widjaya, V., Caulfield, B., Kechadi, T.: Human activity recognition with convolutional neural networks. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pp. 541–552. Springer, Berlin (2018)
17. Bhat, G., Park, J., Ogras, U.Y.: Near-optimal energy allocation for self-powered wearable systems. In: Proceedings of International Conference on Computer-Aided Design (ICCAD), pp. 368–375 (2017)
18. Bhat, G., Deb, R., Chaurasia, V.V., Shill, H., Ogras, U.Y.: Online human activity recognition using low-power wearable devices. In: 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 1–8. IEEE, Piscataway (2018)
19. Bhat, G., Tuncel, Y., An, S., Ogras, U.Y.: Wearable IoT devices for health monitoring. TechConnect Briefs **2019**, 357–360 (2019)
20. Bikias, T., Iakovakis, D., Hadjidimitriou, S., Charisis, V., Hadjileontiadis, L.J.: DeepFog: an IMU-based detection of freezing of gait episodes in Parkinson's disease patients via deep learning. Front. Robot. AI **8** (2021)
21. Borzì, L., Mazzetta, I., Zampogna, A., Suppa, A., Olmo, G., Irrera, F.: Prediction of freezing of gait in Parkinson's disease using wearables and machine learning. Sensors **21**(2), 614 (2021)
22. Buchli, B., Sutton, F., Beutel, J., Thiele, L.: Dynamic power management for long-term energy neutral operation of solar energy harvesting systems. In: Proceedings of the Conference on Embedded Network Sensor Systems, pp. 31–45 (2014)

23. Camille Simon-Al-Araji. Bringing AI to the NBA (2019)
24. Camps, J., Sama, A., Martin, M., Rodriguez-Martin, D., Perez-Lopez, C., Arostegui, J.M.M., Cabestany, J., Catala, A., Alcaine, S., Mestre, B., et al.: Deep learning for freezing of gait detection in Parkinson's disease patients in their homes using a waist-worn inertial measurement unit. Knowl. Based Syst. **139**, 119–131 (2018)
25. Cao, Z., Simon, T., Wei, S.-E., Sheikh, Y.: Realtime multi-person 2D pose estimation using part affinity fields. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 7291–7299 (2017)
26. Choi, S., Choi, S., Kim, C.: MobileHumanPose: toward real-time 3D human pose estimation in mobile devices. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2328–2338 (2021)
27. Deb, R., Bhat, G., An, S., Ogras, U., Shill, H.: Trends in technology usage for Parkinson's disease assessment: a systematic review. medRxiv (2021)
28. Demrozi, F., Bacchin, R., Tamburin, S., Cristani, M., Pravadelli, G.: Toward a wearable system for predicting freezing of gait in people affected by Parkinson's disease. IEEE J. Biomed. Health Inform. **24**(9), 2444–2451 (2019)
29. El-Attar, A., Ashour, A.S., Dey, N., El-Kader, H.A., El-Naby, M.M.A., Shi, F.: Hybrid DWT-FFT features for detecting freezing of gait in Parkinson's disease. In: Information Technology and Intelligent Transportation Systems, pp. 117–126. IOS Press, Amsterdam (2019)
30. Fraternali, F., Balaji, B., Sengupta, D., Hong, D., Gupta, R.K.: Ember: energy management of batteryless event detection sensors with deep reinforcement learning. In: Proceedings of the 18th Conference on Embedded Networked Sensor Systems, pp. 503–516 (2020)
31. Geissdoerfer, K., Jurdak, R., Kusy, B., Zimmerling, M.: Getting more out of energy-harvesting systems: energy management under time-varying utility with PREAcT. In: Proceedings of the 18th International Conference on Information Processing in Sensor Networks, pp. 109–120 (2019)
32. Google: Google completes Fitbit acquisition. https://blog.google/products/devicesservices/fitbit-acquisition/ (2021). Accessed 8 Jul 2021
33. Handojoseno, A.M.A., Shine, J.M., Nguyen, T.N., Tran, Y., Lewis, S.J.G., Nguyen, H.T.: Analysis and prediction of the freezing of gait using EEG brain dynamics. IEEE Trans. Neural Syst. Rehabil. Eng. **23**(5), 887–896 (2014)
34. Hastie, T., Tibshirani, R., Friedman, J.H., Friedman, J.H.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction, vol. 2. Springer, Berlin (2009)
35. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask R-CNN. In: Proceedings of IEEE International Conference on Computer Vision, pp. 2961–2969 (2017)
36. Huu, T.N., Van, T.N., Takahito, O.: Flexible thermoelectric power generator with Y-type structure using electrochemical deposition process. Appl. Energy **210**, 467–476 (2018)
37. IBM: Natural Language Processing (NLP). https://www.ibm.com/cloud/learn/naturallanguage-processing (2021). Accessed 8 Jul 2021
38. IBM: What is computer vision? https://www.ibm.com/topics/computer-vision (2021). Accessed 8 Jul 2021
39. Jokic, P., Magno, M.: Powering smart wearable systems with flexible solar energy harvesting. In: IEEE International Symposium on Circuits and Systems, pp. 1–4 (2017)
40. Kansal, A., Hsu, J., Zahedi, S., Srivastava, M.B.: Power management in energy harvesting sensor networks. ACM Trans. Embedd. Comput. Syst. **6**(4), 32 (2007)
41. Li, B., Zhang, Y., Tang, L., Gao, C., Gu, D.: Automatic detection system for freezing of gait in Parkinson's disease based on the clustering algorithm. In: 2018 2nd IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), pp. 1640–1649. IEEE, Piscataway (2018)
42. Li, B., Yao, Z., Wang, J., Wang, S., Yang, X., Sun, Y.: Improved deep learning technique to detect freezing of gait in Parkinson's disease based on wearable sensors. Electronics **9**(11), 1919 (2020)
43. Li, J., Bian, S., Zeng, A., Wang, C., Pang, B.,Liu, W., Lu, C.: Human pose regression with residual log-likelihood estimation. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 11025–11034 (2021)

44. Mancini, M., et al.: Measuring freezing of gait during daily-life: an open-source, wearable sensors approach. J. Neuroeng. Rehabil. **18**(1), 1–13 (2021)
45. Masiala, S., Huijbers, W., Atzmueller, M.: Feature-set-engineering for detecting freezing of gait in Parkinson's disease using deep recurrent neural networks (2019). Preprint. arXiv:1909.03428
46. Meng, Z., et al.: Gait recognition for co-existing multiple people using millimeter wave sensing. In: Proceedings of AAAI Conference on Artificial Intelligence, vol. 34, pp. 849–856 (2020)
47. Mikos, V., Heng, C.-H., Tay, A., Yen, S.-C., Chia, N.S.Y., Koh, K.M.L., Tan, D.M.L., Au, W.L.: A neural network accelerator with integrated feature extraction processor for a freezing of gait detection system. In: 2018 IEEE Asian Solid-State Circuits Conference (A-SSCC), pp. 59–62. IEEE, Piscataway (2018)
48. Mitcheson, P.D., Yeatman, E.M., Rao, G.K., Holmes, A.S., Green, T.C.: Energy harvesting from human and machine motion for wireless electronic devices. Proc. IEEE **96**(9), 1457–1486 (2008)
49. Naghavi, N., Wade, E.: Prediction of freezing of gait in Parkinson's disease using statistical inference and lower–limb acceleration data. IEEE Trans. Neural Syst. Rehabil. Eng. **27**(5), 947–955 (2019)
50. Naghavi, N., Miller, A., Wade, E.: Towards real-time prediction of freezing of gait in patients with Parkinson's disease: addressing the class imbalance problem. Sensors **19**(18), 3898 (2019)
51. Nguyen, S., Amirtharajah, R.: A hybrid RF and vibration energy harvester for wearable devices. In: IEEE Applied Power Electronics Conference, pp. 1060–1064 (2018)
52. Nvidia. Jetson Nano Developer Kit. https://developer.nvidia.com/embedded/jetson-nanodeveloper-kit (2021). Accessed 8 Jul 2021
53. O'Day, J., Lee, M., Seagers, K., Hoffman, S., Jih-Schiff, A., Kidziński, Ł., Delp, S., Bronte-Stewart, H.: Assessing inertial measurement unit locations for freezing of gait detection and patient preference. J. Neuroeng. Rehabil. **19**(1), 1–15 (2022)
54. Odemakinde, E.: Human pose estimation with deep learning – ultimate overview in 2021 (2021)
55. Oung, Q.W., Basah, S.N., Muthusamy, H., Vijean, V., Lee, H., Khairunizam, W., Bakar, S.A., Razlan, Z.M., Ibrahim, Z.: Objective evaluation of freezing of gait in patients with Parkinson's disease through machine learning approaches. In: 2018 International Conference on Computational Approach in Smart Systems Design and Applications (ICASSDA), pp. 1–7. IEEE, Piscataway (2018)
56. Oura. OURA – The most accurate guide on Sleep, Readiness, and Activity [Online] https://ouraring.com/. Accessed 1 Oct 2021
57. Pardoel, S.: Detection and prediction of freezing of gait in Parkinson's disease using wearable sensors and machine learning (2021)
58. Pardoel, S., Kofman, J., Nantel, J., Lemaire, E.D.: Wearable-sensor-based detection and prediction of freezing of gait in Parkinson's disease: a review. Sensors **19**(23), 5141 (2019)
59. Pardoel, S., Shalin, G., Nantel, J., Lemaire, E.D., Kofman, J.: Early detection of freezing of gait during walking using inertial measurement unit and plantar pressure distribution data. Sensors **21**(6), 2246 (2021)
60. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Steiner, A.B., Fang, L., Bai, J., Chintala, S.: PyTorch: an imperative style, high performance deep learning library. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates Inc., Red Hook (2019)
61. PyTorch: PyTorch Mobile. https://pytorch.org/mobile/home/ (2022). Accessed 8 Jul 2021
62. Rabaey, J.M., Pedram, M.: Low Power Design Methodologies, vol. 336. Springer Science & Business Media, Berlin (2012)
63. Rad, N.M., Laarhoven, T.V., Furlanello, C., Marchiori, E.: Novelty detection using deep normative modeling for IMU-based abnormal movement monitoring in Parkinson's disease and autism spectrum disorders. Sensors **18**(10), 3533 (2018)

64. Ramanan, D., Forsyth, D.A., Zisserman, A.: Strike a pose: tracking people by finding stylized poses. In: Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR), vol. 1, pp. 271–278. IEEE, Piscataway (2005)
65. Rashid, N., Demirel, B.U., Al Faruque, M.A.: AHAR: adaptive CNN for energy-efficient human activity recognition in low-power edge devices. IEEE Internet Things J. **9**(15), 13041–13051 (2022)
66. Raspberry Pi: Raspberry Pi. https://www.raspberrypi.com/documentation/ (2021). Accessed 8 Jul 2021
67. Reches, T., Dagan, M., Herman, T., Gazit, E., Gouskova, N.A., Giladi, N., Manor, B., Hausdorff, J.M.: Using wearable sensors and machine learning to automatically detect freezing of gait during a fog-provoking test. Sensors **20**(16), 4474 (2020)
68. Roetenberg, D., Luinge, H., Slycke, P.: Xsens MVN: full 6DOF human motion tracking using miniature inertial sensors. Xsens Motion Technol. BV. Tech. Rep. **1**, 1–7 (2009)
69. Samà, A., Rodríguez-Martín, D., Pérez-López, C., Català, A., Alcaine, S., Mestre, B., Prats, A., Crespo, M.C., Bayés, À.: Determining the optimal features in freezing of gait detection through a single waist accelerometer in home environments. Pattern Recogn. Lett. **105**, 135–143 (2018)
70. Sani, S., Wiratunga, N., Massie, S.: Learning deep features for KNN-based human activity recognition. In: CEUR Workshop Proceedings (2017)
71. Schaafsma, J.D., Balash, Y., Gurevich, T., Bartels, A.L., Hausdorff, J.M., Giladi, N.: Characterization of freezing of gait subtypes and the response of each to levodopa in Parkinson's disease. Eur. J. Neurol. **10**(4), 391–398 (2003)
72. Sengupta, A., Jin, F., Zhang, R., Cao, S.: Mm-pose: real-time human skeletal posture estimation using mmWave radars and CNNs. IEEE Sensors J. **20**(17), 10032–10044 (2020)
73. Shalin, G., Pardoel, S., Lemaire, E.D., Nantel, J., Kofman, J.: Prediction and detection of freezing of gait in Parkinson's disease from plantar pressure data using long short-term memory neural-networks. J. Neuroeng. Rehabil. **18**(1), 1–15 (2021)
74. Shao, L., Han, J., Xu, D., Shotton, J.: Computer vision for RGB-D sensors: kinect and its applications [special issue intro]. IEEE Trans. Cybern. **43**(5), 1314–1317 (2013)
75. Shi, B., Yen, S.C., Tay, A., Tan, D.M.L., Chia, N.S.Y., Au, W.L.: Convolutional neural network for freezing of gait detection leveraging the continuous wavelet transform on lower extremities wearable sensors data. In: 2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC), pp. 5410–5415. IEEE, Piscataway (2020)
76. Shoaib, M., Bosch, S., Incel, O.D., Scholten, H., Havinga, P.J.M.: A survey of online activity recognition using mobile phones. Sensors **15**(1), 2059–2085 (2015)
77. Sigcha, L., Costa, N., Pavón, I., Costa, S., Arezes, P., López, J.M., De Arcas, G.: Deep learning approaches for detecting freezing of gait in Parkinson's disease patients through on-body acceleration sensors. Sensors **20**(7), 1895 (2020)
78. Sudevalayam, S., Kulkarni, P.: Energy harvesting sensor nodes: survey and implications. IEEE Commun. Surv. Tutorials **13**(3), 443–461 (2010)
79. Svitla: CPU, GPU, and TPU for fast computing. https://svitla.com/blog/cpu-gpu-and-tpu-forfast-computing-in-machine-learning-and-neural-networks (2021). Accessed 8 Jul 2021
80. TensorFlow: TensorFlow Lite: ML for mobile and edge devices. https://www.tensorflow.org/lite (2022). Accessed 8 Jul 2021
81. Texas Instruments: IWR1443BOOST. https://www.ti.com/tool/IWR1443BOOST (2014). Accessed 29 Sep 2020
82. tinyML: tinyML Summit ahead! https://www.tinyml.org/ (2021). Accessed 8 Jul. 2021
83. Torvi, V.G., Bhattacharya, A., Chakraborty, S.G.: Deep domain adaptation to predict freezing of gait in patients with Parkinson's disease. In: 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 1001–1006. IEEE, Piscataway (2018)
84. Tuncel, Y., Bandyopadhyay, S., Kulshrestha, S.V., Mendez, A., Ogras, U.Y.: Towards wearable piezoelectric energy harvesting: modeling and experimental validation. In: Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design, pp. 55–60 (2020)

85. Tuncel, Y., Basaklar, T., Ogras, U.: How much energy can we harvest daily for wearable applications? In: 2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), pp. 1–6. IEEE, Piscataway (2021)
86. Tuncel, Y., Bhat, G., Park, J., Ogras, U.: ECO: enabling energy-neutral IoT devices through runtime allocation of harvested energy. IEEE Internet Things J. **9**(7), 4833–4848 (2022) https://doi.org/10.1109/JIOT.2021.3106283
87. Vakanski, A., Jun, H.-p., Paul, D., Baker, R.: A data set of human body movements for physical rehabilitation exercises. Data **3**(1), 2 (2018)
88. Von Marcard, T., Rosenhahn, B., Black, M.J., Pons-Moll, G.: Sparse inertial poser: automatic 3D human pose estimation from sparse IMUs. In: Computer Graphics Forum, vol. 36, pp. 349–360. Wiley Online Library (2017)
89. von Marcard, T., Henschel, R., Black, M.J., Rosenhahn, B., Pons-Moll, G.: Recovering accurate 3D human pose in the wild using IMUs and a moving camera. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 601–617 (2018)
90. Wang, A., Chen, G., Yang, J., Zhao, S., Chang, C.-Y.: A comparative study on human activity recognition using inertial sensors in a smartphone. IEEE Sensors J. **16**(11), 4566–4578 (2016)
91. Wikipedia: Embedded system. https://en.wikipedia.org/wiki/Embedded_system (2021). Accessed 8 Jul 2021
92. Xue, H., Ju, Y., Miao, C.,Wang, Y., Wang, S., Zhang, A., Su, L.: mmMesh: towards 3D real-time dynamic human mesh construction using millimeter-wave. In: Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services, pp. 269–282 (2021)
93. Yamashita, R., Nishio, M., Do, R.K.G., Togashi, K.: Convolutional neural networks: an overview and application in radiology. Insights Imaging **9**(4), 611–629 (2018)
94. Zebin, T., Scully, P.J., Peek, N., Casson, A.J., Ozanyan, K.B.: Design and implementation of a convolutional neural network on an edge computing smartphone for human activity recognition. IEEE Access **7**, 133509–133520 (2019)
95. Zhang, J., Zhang, D., Xu, X., Jia, F., Liu, Y., Liu, X., Ren, J., Zhang, Y.: MobiPose: real-time multi-person pose estimation on mobile devices. In: Proceedings of the 18th Conference on Embedded Networked Sensor Systems, pp. 136–149 (2020)
96. Zhao, M., et al.: RF-based 3D skeletons. In: Proceedings of Conference of the ACM Special Interest Group on Data Communication, pp. 267–281 (2018)
97. Zhu, S., Anderson, H., Wang, Y.: Reducing the power consumption of an IMU based gait measurement system. In: Pacific-Rim Conference on Multimedia, pp. 105–116. Springer, Berlin (2012)

Reinforcement Learning for Energy-Efficient Cloud Offloading of Mobile Embedded Applications

Aditya Khune and Sudeep Pasricha

1 Introduction

Faster wireless network speeds and rapid innovations in mobile technologies have changed the way we use our computers. It is estimated that 207.2 million people in the United States own a smartphone today while the number of smartphone users worldwide is estimated to be more than two billion [1]. These mobile devices are not only used for making voice calls but also efficiently able to run complex mobile applications that interact with the Internet. The volume of data being accessed and processed by smartphones and the sophistication of mobile applications are rapidly increasing over time. However, the rapid evolution in hardware and software capabilities of mobile devices has not been paralleled by a similar advance in battery technology [2]. As expected, high-end mobile applications increase the burden on the battery life of smartphones. For example, it has been shown that a GPS-based smartphone app can drain a mobile phone's battery completely within 7 hours [3].

A promising solution that is being considered to support high-end mobile applications is to offload mobile computations to the cloud [4–11]. Offloading is an opportunistic process that relies on cloud servers to execute the functionality of an application that typically runs on a mobile device. The terms "cyber foraging" and "surrogate computing" are also sometimes used to describe computation offloading. Such computation offloading is being considered today as a means to save energy consumption (thereby improving battery lifetime) and increase the responsiveness of mobile applications. The potential of computation offloading lies in the ability to

A. Khune
Qualcomm, Longmont, CO, USA

S. Pasricha (✉)
Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO, USA
e-mail: sudeep@colostate.edu

S. Pasricha, M. Shafique (eds.), *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing*, https://doi.org/10.1007/978-3-031-40677-5_7

sustain power hungry applications by releasing the energy consuming resources of the smartphone from intensive processing requirements.

In this chapter, we present the details of a framework for mobile-to-cloud offloading that was first presented in [12]:

- We study the behavior of a set of popular smartphone applications, in both local and offload processing modes. We identify possible bottlenecks during mobile-to-cloud offloading, as a function of the applications functional characteristics, such as data intensiveness and computation intensiveness. This study is crucial to establish the pros and cons of offloading when using various wireless networks.
- We quantify the influence of different wireless network technologies on mobile-to-cloud offloading. We perform several experiments to gain a clear understanding of the impact of selecting the appropriate network when offloading, while considering advances in current high-speed wireless data communication networks such as 3G, 4G, and Wi-Fi.
- We propose a novel middleware framework that uses a machine-learning technique called reinforcement learning (RL) to make offloading decisions effectively on a smartphone. The proposed framework considers various types of information on the mobile device, such as network type, network bandwidth, and user-context, to decide when to offload in order to minimize energy consumption. Our strategy utilizes unsupervised machine learning to select between available networks (3G, 4G, or Wi-Fi) when offloading mode is active. Our experiments with real applications on a smartphone highlight the potential of our framework to minimize energy consumed in mobile devices.

2 Prior Work

Many prior research efforts have proposed strategies to reduce energy consumption in mobile devices via machine learning-based device resource management methods [13–18] and offloading strategies [4, 8, 19, 20]. Kumar et al. [19] presented a mathematical analysis of offloading. Broadly, the energy saved by computation offloading depends on the amount of computation to be performed (C), the amount of data to be transmitted (D), and the wireless network bandwidth (B). If (D/C) is low, then it was claimed that offloading can save energy. Flores et al. [8] proposed a fuzzy decision engine for code offloading. The mobile device uses a decision engine based on fuzzy logic to combine various factors and decide when to offload. *Our framework discussed in this chapter considers many more factors than these works, such as network type, data size, and degree of computations when making decisions about offloading.* Cuervo et.al [4] proposed a system called MAUI, based on code annotations to specify which methods from a software class can be offloaded. Annotations are introduced in the source code by the developer during the development phase. At runtime, methods are identified by a MAUI profiler, which performs the offloading of the methods, if the bandwidth of the network and data

transfer conditions are ideal. MAUI aims to optimize both energy consumption and execution time, using an optimization solver. However, this annotation method puts an extra burden on the already complex mobile application development phase. Moreover, such annotations can cause unnecessary code offloading that drains energy [20]. *To reduce the complexity of the application development process, we recommend transferring the entire application processing to the cloud rather than utilizing a design-time code partitioning method. Further, we propose a novel adaptive reward-based machine learning approach to make smart offloading decisions that can achieve high energy efficiency with offloading and also improve application response time.*

3 Challenges with Offloading

In spite of existing research highlighting the potential of offloading in mobile devices, current offloading techniques are far from being widely adopted in mobile systems. The implementation of these computation offloading techniques for many real-world mobile applications in real-world scenarios has not shown promising results [6], with the mobile device consuming more energy in the offloading process than the energy savings achieved due to computing on servers in an offloaded manner.

Offloading decision engines must consider not only the potential energy savings from offloading but also how the response time of the application is impacted by offloading. An effective decision to offload processing to the cloud must reduce energy without significantly increasing response time. Such decisions are heavily impacted by wireless network inconsistency. The power consumed by the network radio interface is known to contribute a considerable fraction of the total device power, and it varies depending on wireless signal strength [21]. With the recent advent of high bandwidth 4G networks, there has been increased interest in the offloading domain, but from our experiments and results presented in later sections of this chapter, we found that 4G consumes more energy than Wi-Fi and 3G. Some of the prior works [22] in this area also confirm this observation.

The network quality of a 4G connection at a mobile device's location greatly affects the battery life. If the device is in the area that does not have 4G coverage, there is no advantage to a 4G interface, and if 4G network search is not disabled, then the radio's search for a nonexistent signal will drain the battery quickly. In case of a weak signal, the device uses more power to send and receive data to and from the network. A strong 4G signal uses less battery, but the biggest problem is the constant switching from 4G to 3G and back again. Also, throughout a typical day, at different times, the performance of a wireless network varies because of changing traffic load on the network. We refer to all such problems due to the mobile network as "network inconsistency" problems.

To counter the impact of network inconsistency on a mobile device and to optimize the offloading experience, we propose a novel offloading framework based

on reinforcement learning. This framework not only decides when to offload but also helps a mobile device select between the different available wireless networks, to achieve consistent improvements by using offloading even in the presence of varying network conditions. In the following sections, we describe our framework in detail.

4 Offloading Performance of Mobile Applications

We analyzed the performance implications of offloading by comparing two scenarios – one where all computations are performed only on the mobile device without using the cloud at all (local mode) and the other where there was a complete reliance on the cloud computation (offload mode), with minimal computations on the mobile device. We selected five diverse and popular commercially available smartphone applications for our experiment. Our evaluation focuses on two metrics: (i) battery consumption and (ii) response time. We compared the results obtained with these applications for 3G, 4G (HSPA+), and Wi-Fi networks. This comparative study was meant to help us identify various factors that need to be considered for the design of cloud offloading strategies for mobile applications, for example, identifying the best possible network for offloading to the cloud, for a given mobile application, at a specific location.

4.1 Experimental Setup

The power estimation models required to estimate battery consumption were built using power measurements on the LG G3 device running the Android OS version 5.0.1. The contact between the smartphone and the battery was instrumented, and current was measured using the Monsoon Solutions power monitor [23] (Fig. 1). The monitor connects to a computer running the Monsoon Solutions power tool software, which allows real-time current and power measurements. We also used the Android Device Bridge, a software tool to perform battery drain measurements on the android device. The experiments were performed using AT&T's 3G and 4G (HSPA+) networks and Comcast's 100 Mbps (2.4 GHz Band) Wi-Fi network. We performed these experiments around the Colorado State University campus in Fort Collins, Colorado, the United States.

Before conducting our experiments, we followed a few rules to ensure meaningful and accurate results while avoiding human error. These rules are as follows: (1) set the device's screen to a consistent and fixed brightness level, to minimize interference from varying screen power consumption (e.g., for different ambient light scenarios); in our measurements, we used the lowest screen brightness

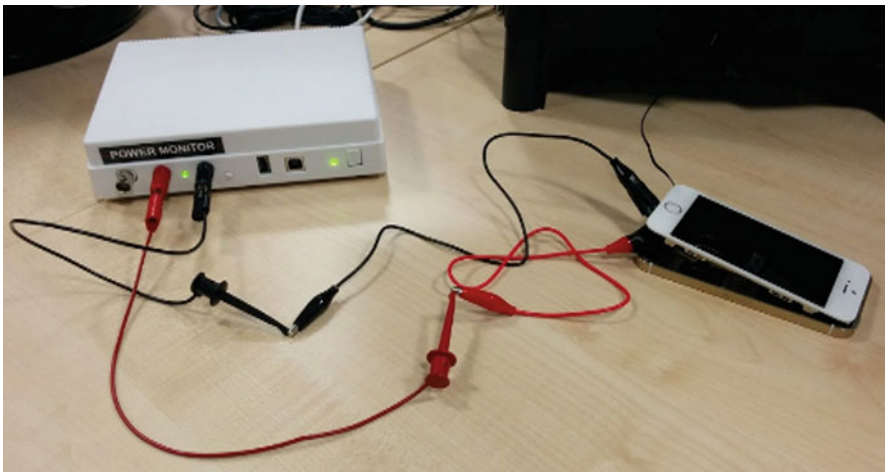


Fig. 1 Monsoon power monitor setup

level; (2) kill all background processes before measurements; and (3) repeat each experiment over 15 iterations to improve result confidence and minimize human error. We selected five diverse commercially available smartphone applications for our experiments: (i) matrix operations, (ii) Internet browser; (iii) zipper (file compression); (iv) voice recognition and translation; and (v) torrent (file download).

The next subsection gives the details of all the applications considered and the results of their execution for the two scenarios (local and offloading modes) outlined earlier.

4.2 Experimental Results

4.2.1 Matrix Operation App

The matrix calculator app [24] runs on android-based devices. The user is first asked to enter the size of the matrix and all the digits of the matrix manually, and then the user can direct the application to calculate the inverse of that matrix (using the adjoint method). For our experiments, we used a set of matrix sizes from 3×3 to 9×9. For the cloud part, we implemented the functionality of calculating the matrix inverse using the Amazon Web Services (AWS) EC2 cloud instance [25]. Figure 2 shows the results from our experiment. The energy consumption in local processing mode is equal to the battery drain in the device while performing the matrix operation, whereas in the cloud mode, energy consumption is the total of battery drain during the idle time of the mobile device while the operation is being performed remotely on the cloud and the time for data transfer between the mobile device and the cloud.

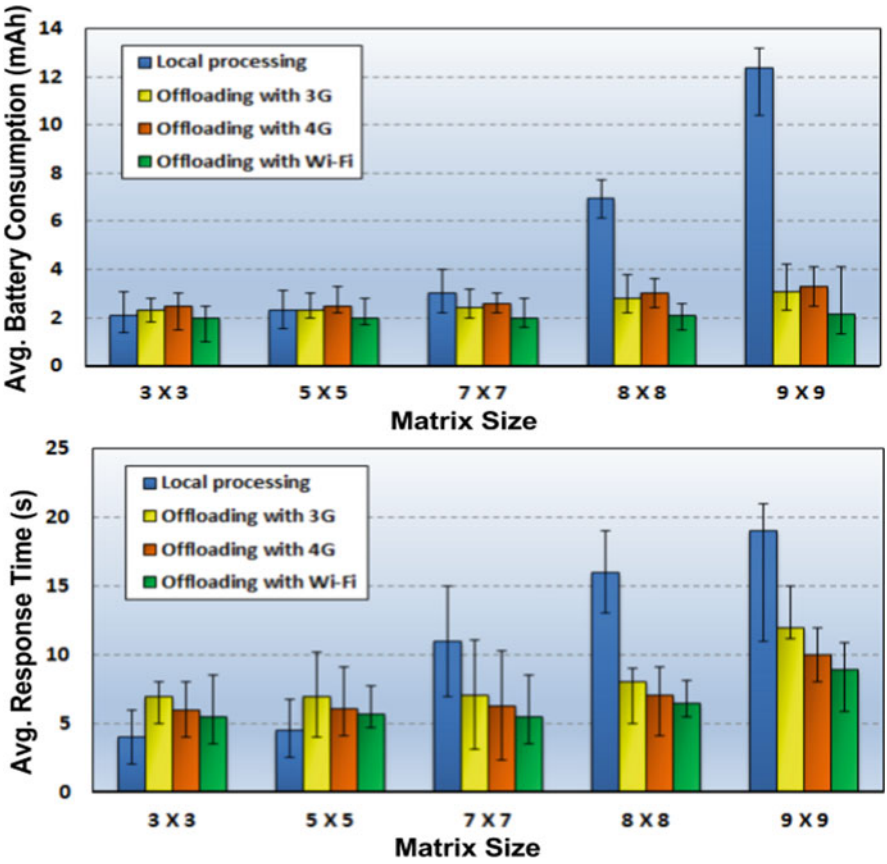


Fig. 2 Average battery consumption and average response time on a mobile device for a matrix operation with varying matrix sizes

It can be observed that in the local processing mode, the battery consumption of the device increases manifolds with the increasing matrix size, largely, because there is an increase in the CPU's energy consumption as the number of floating point operations increase. Local processing is found to be suitable for operations on small matrices (i.e., 3×3 and 5×5) allowing for low energy consumption on the device and low response time. On the other hand, offloading to the cloud saves energy and reduces response time when the matrix size increases. The device in offloading mode saves maximum energy (and also has minimum response time) when used with Wi-Fi. The results show that 3G performs slightly better than 4G as far as energy is concerned, whereas 4G gives better response time than 3G for the same operations.

4.2.2 Internet Browser App

Cloud-based web browsers use a split architecture where processing of a mobile web browser is offloaded to the cloud partially. This involves cloud support for most browsing functionalities such as execution of JavaScript, image transcoding and compression, and parsing and rendering of web pages. For our experiments, we used the Mozilla Firefox [26] and Puffin [27] browsers. Puffin is a commercially available cloud-based mobile browser, and Mozilla Firefox is a local browser available from the Google Play store. Our experiments are performed for a data range starting as low as 150 Kib to a session involving 5 MB of data transfer to load the web pages. Figure 3 shows the results obtained by measuring data transfer (response) time and energy consumed by these browsers for loading two different websites: (i) www.yahoo.com and (ii) www.wikipedia.org.

We observed that the results obtained fluctuated significantly due to network inconsistency. For example, the plots in Fig. 3 show that the response time/battery consumption of a browser session with around 3 MB data usage is sometimes more than that of a session which uses 5 MB of data. To counter such network inconsistency problems, we conducted 15 iterations of each experiment across different locations and at different times of the day. In general, our results show that cloud-based web browsers are faster but more expensive in terms of energy consumption. For small data transfers, it is suitable to use web browsers with local processing to save energy. For a typical user, the data transfer amount during a browsing session does not go beyond 5–6 MBs for a single session. Thus, for most websites in typical usage scenarios, a local browser will provide greater energy savings than when using offloading. The response time results indicate that for larger data usage scenarios, offloading can be beneficial. Fourth generation not only provides lower response time but also consumes more energy than 3G for the offloading scenarios. Wi-Fi outperforms both 3G and 4G in offloading mode, for response time and energy consumption.

4.2.3 Zipper App

Zipping large files in order to compress them is a widely used functionality on most computers. Zipper [28] is an android app that compresses files locally on a mobile device. For the cloud-based file compression, we used an AWS cloud instance and zipping tool available on the web [29]. Figure 4 shows the results for energy consumption and the response time when zipping various PDF and Word document files ranging in size from 15–255 MB. It can be observed that for the zipping operation, local computation is most efficient in terms of energy consumption. Offloading provides benefits only in response time and that too only for large file sizes. When offloading, 4G consumes more energy than 3G for smaller file sizes (15–105 MBs) whereas 3G consumes more energy than 4G for larger file sizes (175–255 MBs). Fourth generation is faster than 3G but slower than Wi-Fi. Wi-Fi gives the best results in terms of energy and response time when offloading.

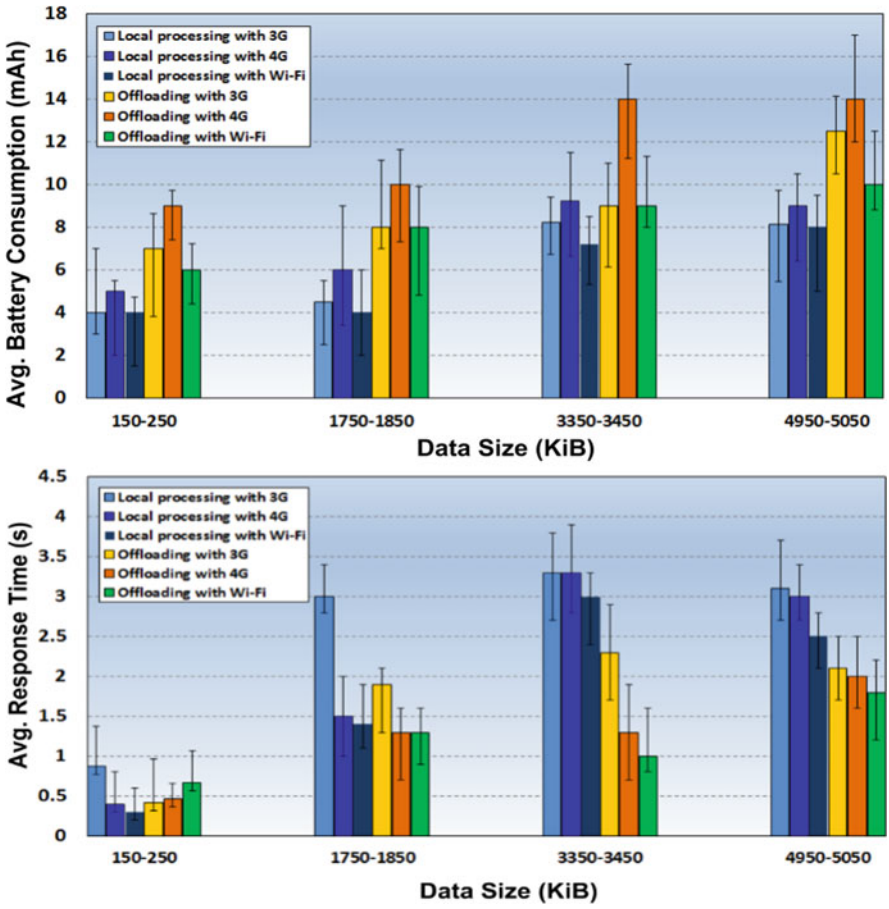


Fig. 3 Average battery consumption and response time on a mobile device for an Internet browsing session with varying data sizes

4.2.4 Voice Recognition and Translation App

There are several popular apps for voice recognition and translation available from app stores, for example, Google Translate [30] for android and Speak and Translate [31] for iOS. Google Translate is a cloud-based app, which also has an offline translation mode that performs local processing on the device with a small neural network. The application allows for downloading an installation package to support the local processing mode. It makes use of the statistical machine translation method, which relies on large amounts of data to train a machine translation engine.

Figure 5 shows the energy consumption of the Google Translate app for a range of words. These measurements were recorded while translating 20–140 words from the English (the United States) to Marathi language. From the results in

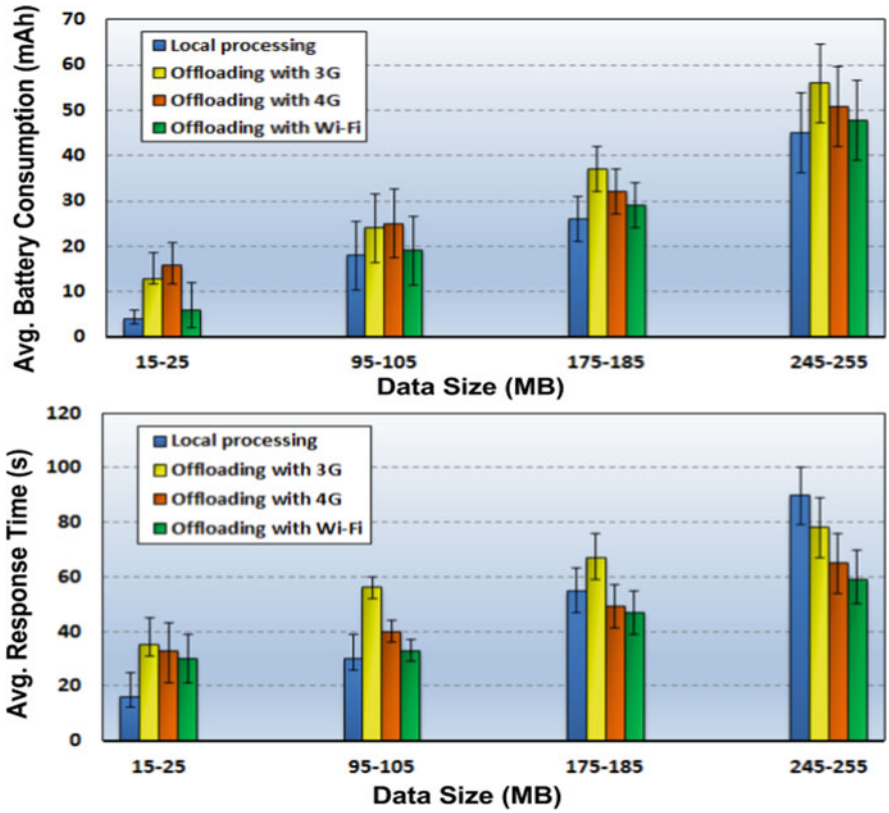


Fig. 4 Average battery consumption and response time on a mobile device for zipping/compressing files of varying sizes

Fig. 5, we can clearly observe that the local processing mode is more efficient in terms of energy consumption as compared with the cloud processing mode. The voice recognition and translation accuracy for local processing was 79.26% and for offloaded processing was 88.51%. This is because the offloaded voice data is processed by more powerful cloud servers, which are capable of running the complex computations of a larger neural network and other machine learning algorithms for more efficient translation.

4.2.5 Torrents App

We used the android-based torrent app Flud [32] to perform torrent downloads in local mode. In the cloud mode, a cloud server is used as a BitTorrent client to download torrent pieces on behalf of a mobile device. While the cloud server downloads the torrent, the mobile device switches to the sleep mode until the

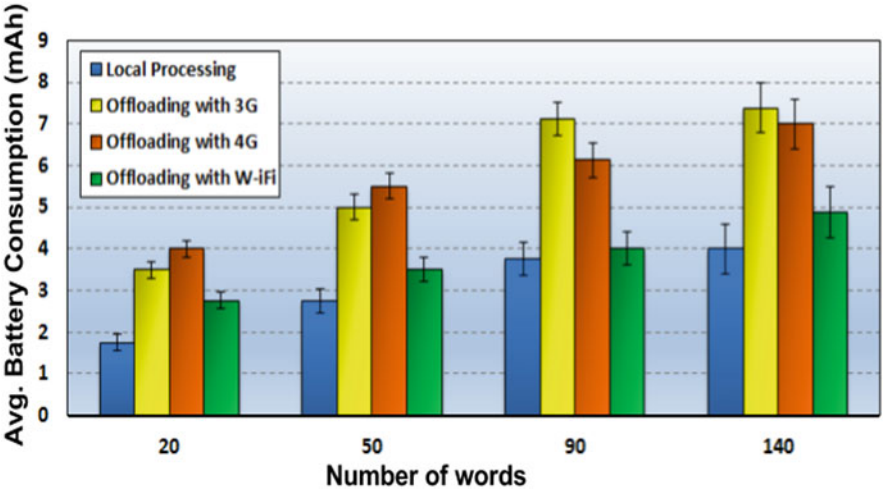


Fig. 5 Average battery consumption on a mobile device for voice recognition and translation operations

cloud finishes the torrent processes, and then the cloud uploads the downloaded torrent file in a single process to the mobile device. Kelenyi et al. [33] presented a similar strategy for torrent file download. This strategy saves energy consumption in smartphones as downloading torrent pieces from multiple peers consumes more energy than downloading one burst of pieces from the cloud.

For our experiments, we used torrent file sizes ranging from 25–85 MB, with an AWS cloud instance being used for the cloud mode. Figure 6 shows the results of our experiments for this application. It is interesting to note that out of all the applications that we consider, offloaded processing proves to be most beneficial in terms of both energy savings and response time for the torrent download application, which is data intensive but not compute intensive. Fourth generation is faster than 3G but slower than Wi-Fi, which is consistent with earlier observations. Fourth generation performs slightly better than 3G in terms of energy consumption for higher data sizes (45–85 MBs), but for smaller data sizes, 3G is more energy efficient.

4.3 Summary of Findings

The overall performance when offloading depends on various factors such as the amount of data required by the application, wireless network signal type and strength and the functionality of the application under consideration. In some prior work [19, 34], it was concluded that offloading is beneficial when an application is compute intensive and at the same time less data intensive. However, we found

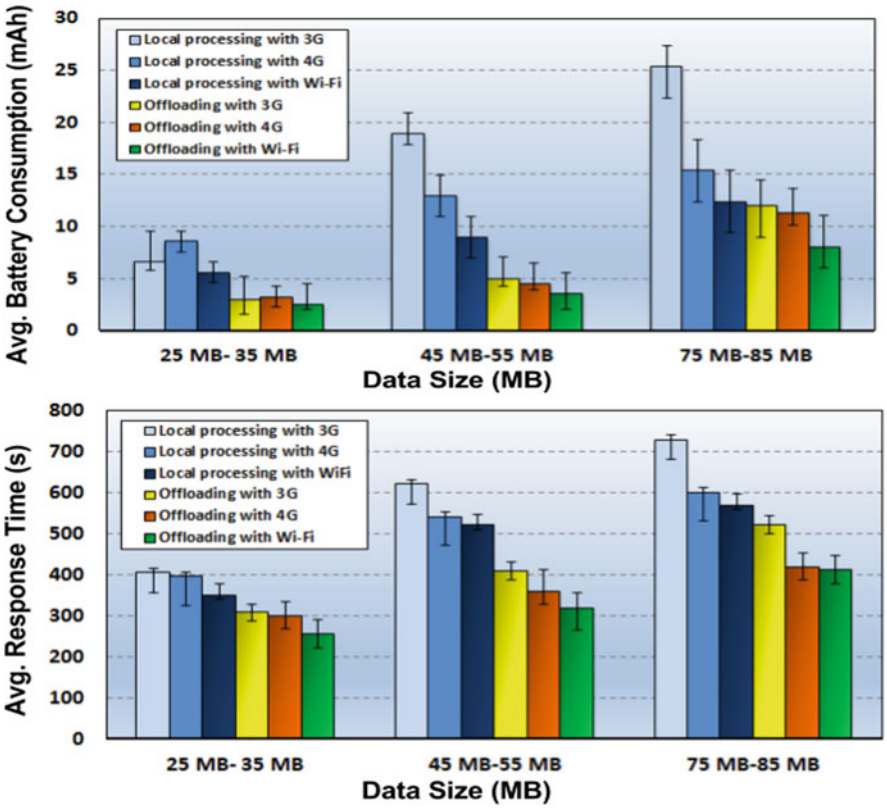


Fig. 6 Average battery consumption and response time on a mobile device for torrent file download operations

that this is not always the case. For instance, offloading is beneficial for applications that may not be compute intensive, but are data intensive, for example, the torrent application.

To make offloading more practical, it is important to reduce the energy spent in the communication between the mobile device and the cloud. Our experiments indicate that choosing the best possible network for offloading is a critical decision. One may assume that because 4G is faster than 3G, we should always rely on it for offloading when Wi-Fi is not available. However, our results indicate that 4G is more power hungry than 3G most of the time.

Network quality is also a factor that cannot be ignored. We found that a good 3G-coverage performs far better as opposed to poor 4G-coverage and vice versa. In the region of cell tower edges or where the coverage of 3G/4G ends, we found that the handover process results in high battery drain. This is because the device in such scenarios is constantly searching for the network, frequently scanning the wireless spectrum around it to determine which tower it should tether itself to. The more

networks there are to choose from, the longer the scans take. Some apps require a channel to be established between the base station and the mobile device at regular intervals, which can significantly drain the device battery.

Another observation is that as 4G generally provides faster data rates than 3G, users tend to consume more data when connected on 4G than 3G. The radio-network interface in the 4G (or LTE) device is functionally a lot more sophisticated and does a lot more than a 3G interface. This interface is the single biggest source of battery drain in a mobile device, apart from its display. Unlike the display, however, the network interface radio is always on.

In conclusion, we observed from our experiments on real applications running on a real mobile device that the overall performance of offloading depends on various factors, such as the amount of data used by the application, network signal type (3G, 4G, and Wi-Fi), network signal strength, and the complexity of the functionality of the application under consideration.

5 Adaptive Offloading

The decision to offload a mobile application to the cloud is a complex one due to the distributed nature and many real-time constraints of the overall system. To make an effective offloading decision, it is vital to consider various factors as we discovered after our experimental analysis presented in the previous section. As these factors vary at runtime, there is a need for an adaptive offloading approach that takes the variations of these factors at runtime into consideration when making decisions.

A few prior works [8, 9] propose an offloading decision engine that considers the contextual parameters on a device and on the cloud to make an offloading decision adaptively. Flores et al. [8] proposed a fuzzy decision engine for code offloading. The mobile device runs the fuzzy logic decision engine, which is utilized to combine n number of variables (e.g., application data size and network bandwidth) that are obtained from the overall mobile cloud architecture. The fuzzy logic decision engine works in three steps, namely: fuzzification, inference, and defuzzification. In fuzzification, input data is converted into linguistic variables, which are assigned to a specific membership function. A reasoning engine is applied to the variables, which makes an inference based on a set of rules. Lastly, the outputs from the reasoning engine are mapped to linguistic variable sets again in the defuzzification step. This offloading decision engine in [8] assumes a consistent network performance during offloading. However, as observed in our experiments, such consistency is difficult to achieve because of frequent mobile user movements and variable network quality (due to factors such as location of the device and load on the network [21]). Moreover, the offloading decision engine in [8] mainly emphasizes energy savings; however, response time is also a crucial metric for various applications that should not be ignored, otherwise user quality of service degradation can become so severe that any effort to save energy becomes irrelevant.

In the next section, we describe our reward-based middleware framework for adaptive offloading that overcomes the challenges mentioned above, to make more efficient decisions related to when and how to offload applications from a mobile device to the cloud.

6 Middleware Framework for Efficient Offloading of Mobile Applications

To simplify the mobile application development process and at the same time avoid problems caused by hard coded annotations, our framework proposes to transfer all the computation for an application to the cloud instead of partial (selective) offloading of the application. Our framework involves a novel decision engine on the mobile device that works together with a clone virtual machine (VM) of the mobile software environment to execute applications on cloud servers. Figure 7 shows a high-level overview of the proposed framework. The framework is implemented at the middleware level in the software stack of the Android OS and runs in the background as an android service. As a result, our framework requires no changes to any of the applications or the Android OS. The runtime monitor component periodically triggers the reinforcement learning (RL) module to generate/update a Q-learning table. At any time, this Q-table contains information to guide the decision for when and how to offload an application to the cloud, depending on multiple factors. The remainder of this section provides a detailed overview of the RL mechanism and our algorithm to generate and use the Q-table.

6.1 Reinforcement Learning (RL)

RL is an unsupervised learning approach, which focuses on learning by having software agents interact with an environment and then taking actions to maximize some notion of a reward. In supervised learning (e.g., using neural networks), a training set of correctly identified observations is required to train a prediction model. RL differs from supervised learning in that correct input/output pairs of identified observations do not need to be presented, so there is no need for a pretrained model. Moreover, an RL algorithm performs well as it has better exploration capabilities than unsupervised learning methods. For this reason, RL is being widely used in gaming and control problems, for example, to determine the next best move in games [35, 36]. RL cuts down the need to manually specify rules, and agents learn simply by playing the game or exploring different moves in an automated manner.

In RL, the state-action value function is a function of both state and action, and its value is a prediction of the expected sum of future reinforcements. The state-

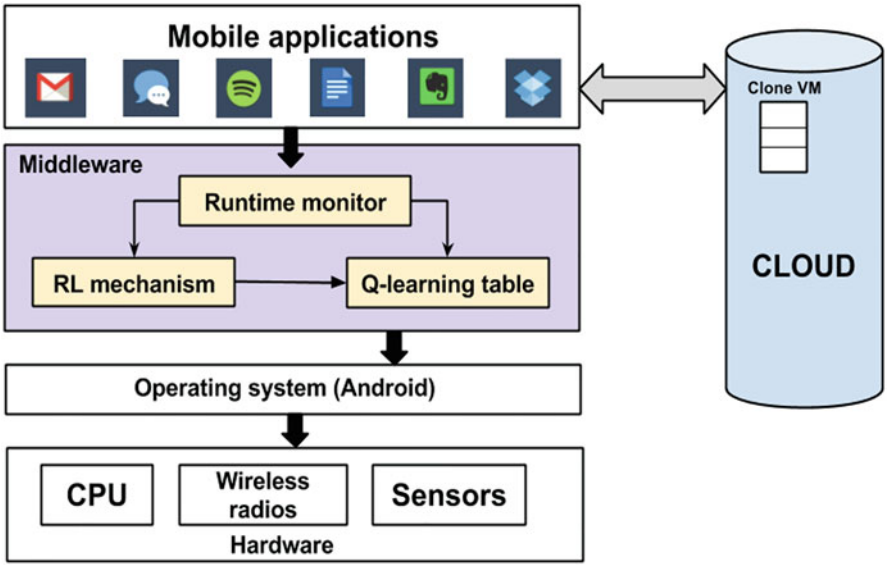


Fig. 7 Reinforcement learning (RL)-based middleware framework for efficient application offloading to cloud

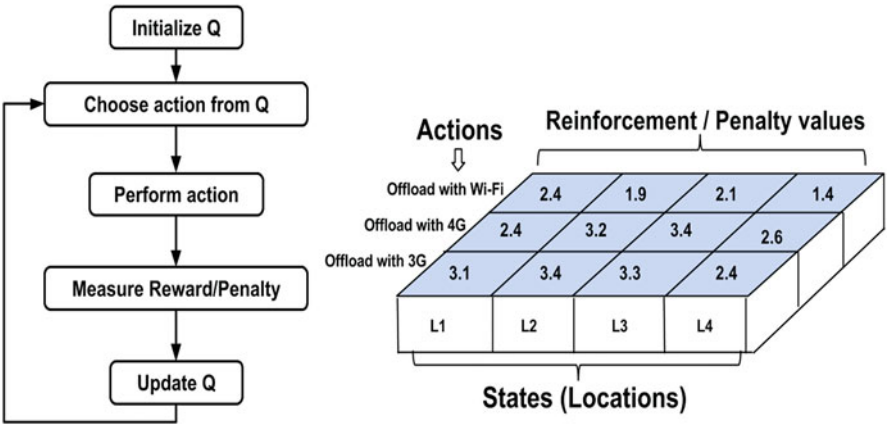


Fig. 8 Q-learning flow with example of Q-table

action value function is referred to as the Q-function [37]. Figure 8 summarizes how a typical Q-learning reinforcement algorithm works. Q-learning is a reward-based mechanism that generates a Q-table with reinforcement or penalty values. The figure illustrates a section of a Q-table where the possible actions are offloading with 3G, 4G, or Wi-Fi network, when the user is at different locations L1–L4. Actions are chosen, and the penalty values are calculated for respective actions to update the Q-table.

Suppose the system is at a defined state s_t at time t. Upon taking action a_t from that state, we observe the one step reinforcement r_{t+1}, and the next state becomes s_{t+1}. This continues until we reach a goal state, K steps later. The reward R_t in this goal state is shown below:

$$R_t = \sum_{k=0}^{K} r_{t+k+1} \tag{1}$$

The objective with RL is to find actions a_t that maximize (or minimize) the sum of reinforcements or rewards r_t in Eq. (1). This can be reduced to the objective of acquiring the Q-function $Q(s_t, a_t)$ that predicts the expected sum of future reinforcements, where the correct Q-function determines the optimal next action. So, the RL objective is to make the following approximation as accurate as possible:

$$Q\left(s_t, a_t\right) \approx \sum_{k=0}^{\infty} r_{t+k+1} \tag{2}$$

The Q-function stores reinforcement values for each state and action pair of the system. Eq. (2) formulates the RL for a multistep decision problem (e.g., predicting sequential actions in a Tic-Tac-Toe game [37]). In our middleware framework, we use RL for a single-step decision problem as there are no sequential states that are dependent on the previous state of the system. This version of the problem is formulated as:

$$Q\left(s_t, a_t\right) \approx \sum_{t=1}^{n} r_t \tag{3}$$

The Q-function is ultimately queried by the system to select the optimal action a_t, in state s_t:

$$a_t = \arg\ \min\ Q\left(s_t, a\right) \tag{4}$$

6.2 RL Algorithm to Generate Q-Function

The state of a mobile device is defined using the contextual information of the device such as its location, available network type, and network strength. These contextual factors are chosen as we consider them to be crucial for efficient offloading. The runtime monitor extracts the contextual information of the device to form state values of the system. For example, consider a mobile device that is at location L1, where it has access to a 3G network type with "strong" network strength. From this state, if an application processing needs to be offloaded, then the Q-function

is called to select the appropriate network that would result in the least penalty in terms of energy or response time (or both). In our framework, the following state and action values are used to generate the Q-function:

Set of state values (discrete values):

- Location = L1, L2, L3, . . . , L*n*
- Network carrier = 3G, 4G, Wi-Fi
- Network strength = Strong, Medium, Weak
- Data Size = data_small, data_medium, data_large

Set of action values

- Offload using 3G network
- Offload using 4G network
- Offload using Wi-Fi network

The location L1-Ln can be any geographic area where the user uses the offloading application, for example, office and home. More state-action pairs can be added to the above list to account for factors that might affect offloading, for example, we can add "Time of Day" as another state value, as it is observed that network performance is slow at certain times of day when the network load is high. However, a larger set of state-value pairs will result in a larger Q-function requiring greater overhead to manage it.

The Q-function is generated as follows: Initially, when the mobile device is at a location L1, the runtime monitor accesses contextual information from the device such as location, networks available, and network strength. A small data file is then uploaded from the mobile device to the cloud, using the primary network carrier. The battery consumed and total response time taken for this operation are measured. The uploading operation is repeated with varying (small, medium, and large) data sizes with all available networks at the location (3G, 4G, and Wi-Fi) activated one by one. For each of these uploading operations, the runtime monitor measures the battery amount consumed and response time to complete the operation. The Q-table is then populated with the penalty values calculated using Eqs. (5), (6), and (7):

$$P_{3G} = P_{b3G} * x + P_{t3G} * y \tag{5}$$

$$P_{4G} = P_{b4G} * x + P_{t4G} * y \tag{6}$$

$$P_{WiFi} = P_{WiFi} * x + P_{WiFi} * y \tag{7}$$

Thus, in our RL framework, the reinforcement values are essentially the penalty values P_{3G}, P_{4G}, and P_{WiFi}. The set of possible individual penalty values are shown in Table 1. Once populated, the Q-table can be updated periodically in the background when the user is not actively using the device. In Eqs. (5), (6), and (7), to optimize battery consumption and response time, we used weights x and y, respectively, with penalty values. Both x and y parameters take values between 0

Table 1 Penalty values in RL algorithm

Penalty values	Offload w/ 3G	Offload w/ 4G	Offload using Wi-Fi
Battery (P_b) processing	P_{b3G}	P_{b4G}	P_{bWiFi}
Response time (P_t)	P_{t3G}	P_{t4G}	P_{tWiFi}
Total penalty	P_{3G}	P_{4G}	P_{WiFi}

Fig. 9 Decision-making using Q-table (vector of key value pairs)

and 1. For our experiments in Sect. 7, we used $x = 0.5$ and $y = 0.5$ to balance minimizing battery consumption and response time.

Figure 9 shows an example of the decision-making process with the help of two simple scenarios. For a data intensive application at location L1, we have 3G and 4G networks available as shown in first two lines of the Q-table in the figure. The penalty value for 4G at location L1 is lesser; therefore, the 4G network is selected for offloading the application to the cloud. For a less data intensive application at location L2, out of all the networks available, 3G is selected because Wi-Fi has weak signal strength with higher penalty and 4G also has a higher penalty.

7 Experimental Results

To evaluate the efficacy of our proposed framework, we conducted a set of experiments. We implemented our middleware framework and its decision engine on an android-based mobile device. To form the Q-function of our RL algorithm, real user data was collected at different geographical locations around the Colorado

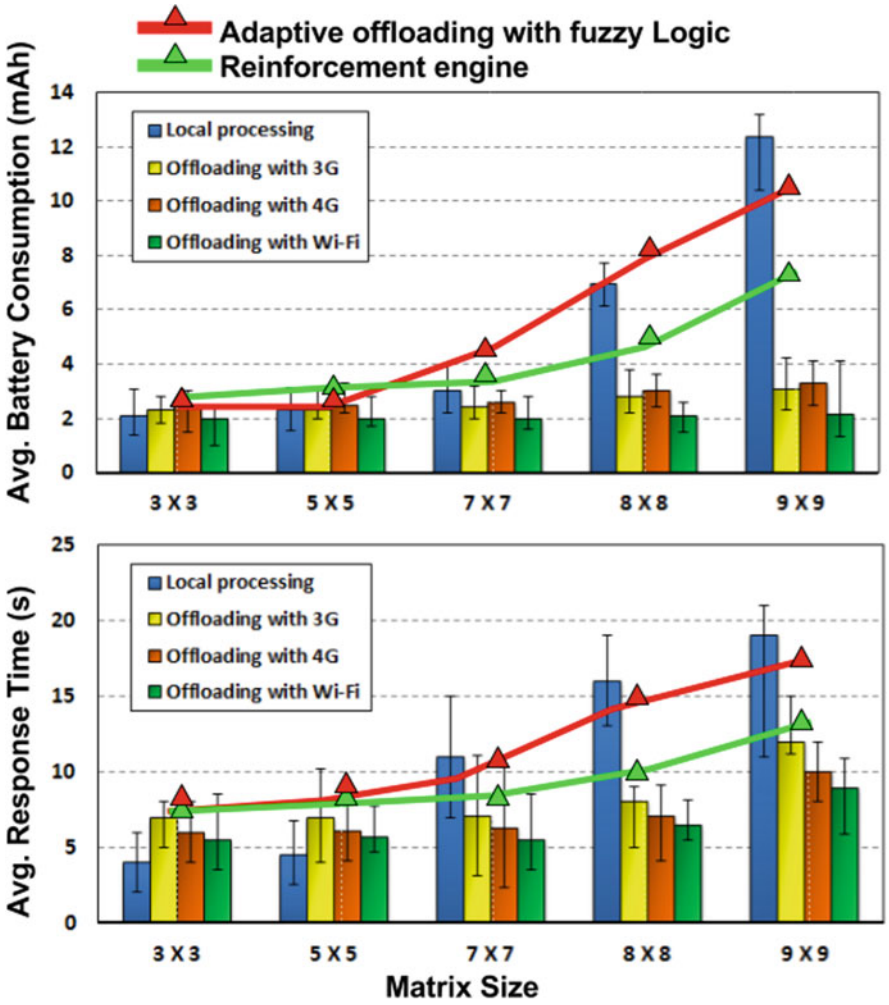


Fig. 10 Average battery consumption and response time of Matrix operations app with learning methods

State University campus area, in Fort Collins, Colorado. We compared our work with the fuzzy logic decision engine proposed by Flores et al. [8], which we discussed in Sect. 5 and which we also implemented on the android-based mobile device.

Figure 10 shows the results for the matrix operation app with our proposed RL-based decision engine and the fuzzy logic-based decision engine from [8]. Similarly, Fig. 11 shows the results for the zipper app, and Fig. 12 shows results for the torrent app. In all the scenarios, the task of a decision engine is to decide whether to offload and select the network to offload with. In these figures, the red trend line shows

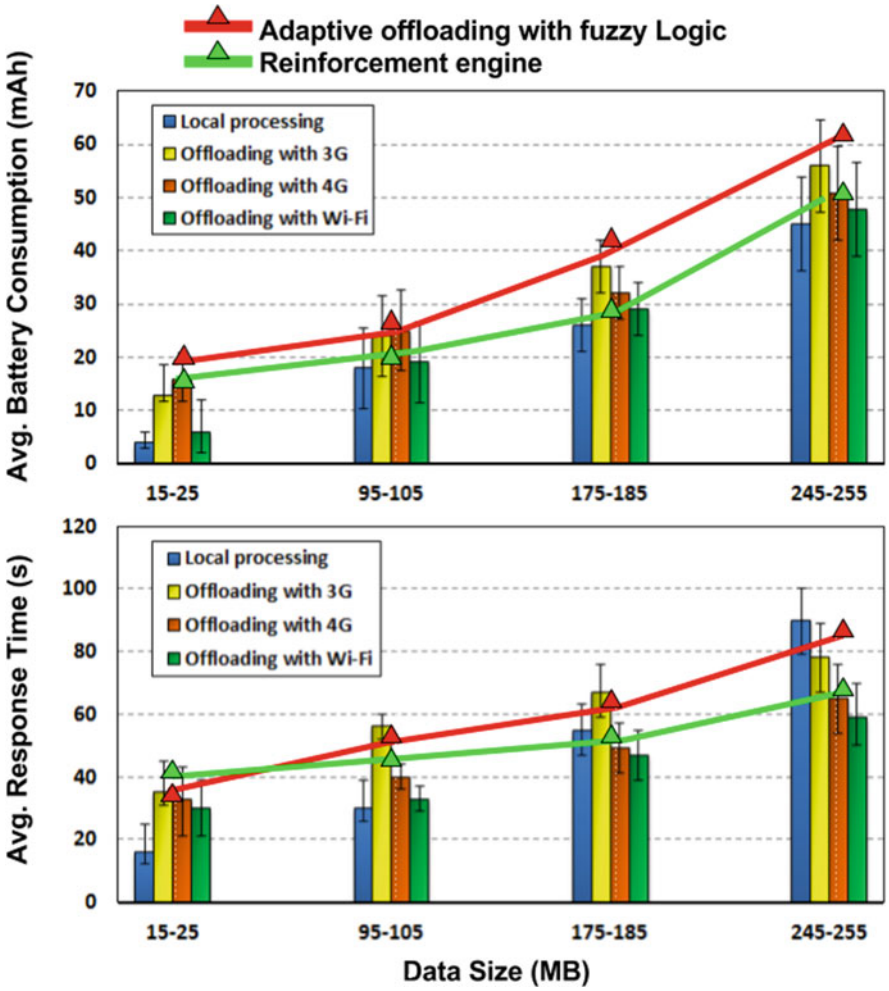


Fig. 11 Average battery consumption and response time of zipper app with learning methods

results with the fuzzy decision engine [8] whereas the green trend line shows the results with our RL-based middleware framework. We have also shown bars with the results for offloading with each available network and local processing (from Sect. 4) as a reference.

In general, our results show that our proposed RL-based decision engine outperforms the fuzzy logic approach from [8]. For less data intensive operations, the results of RL and fuzzy logic overlap. For instance, in the case of the zipper application (Fig. 11), for lower data sizes fuzzy logic shows better results, possibly because the Q-table generated using our RL algorithm uses 25 MB as the minimum data size. For any data size lower than this minimum value, the RL-based framework

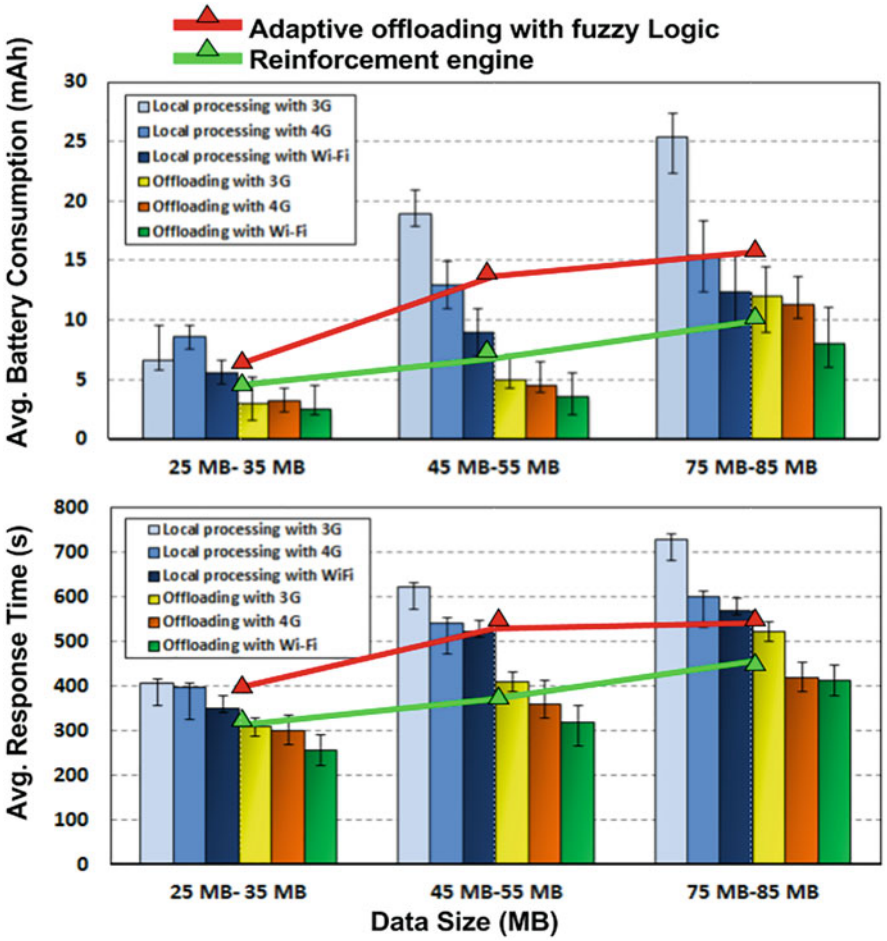


Fig. 12 Average battery consumption and response time of torrent app with learning methods

is thus less effective at making predictions. This can be improved using a wider range of data files/sizes when populating the Q-table. For higher data sizes and more complex computations, our RL approach gives improved battery consumption and response time than [8].

Figure 13 summarizes the prediction accuracy of both the learning methods being compared. It can be observed that our RL-based engine has better prediction accuracy, which is crucial for making effective offloading decisions. The overall performance of offloading depends on various factors, such as the amount of data required by the application, network signal type (3G, 4G, and Wi-Fi) and network signal strength, and the complexity of the functionality of the application under observation. By considering all of these individual factors in the decision process, unlike the fuzzy logic approach from [8], and by utilizing a more sophisticated and

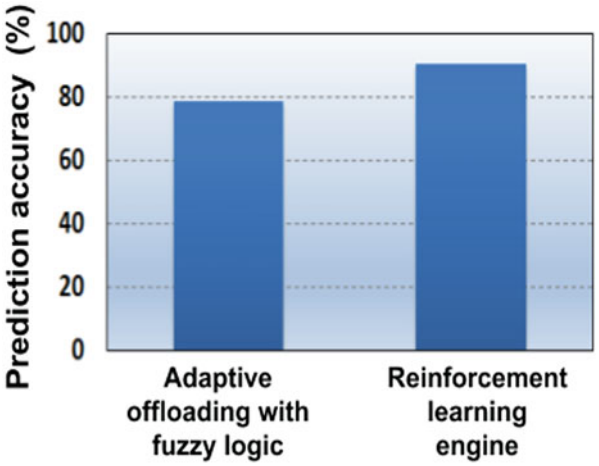


Fig. 13 Prediction accuracy of learning methods

powerful learning algorithm, our framework is able to achieve notably better results compared with [8]. Our results show that proposed RL-based offloading system can save up to 30% battery power with up to 25% better response time as compared with the fuzzy logic-based approach.

8 Conclusions and Future Work

In this chapter, we analyzed real mobile applications to determine the benefits of application offloading. We found that overall performance with offloading depends on various factors such as the amount of data and type of usage, available network carrier, and signal strength. These factors should be considered while making a decision to offload a mobile application. To make offloading more practical, it is important to reduce the energy spent in the communication between the mobile device and the cloud. In our experiments, we compared energy consumption in mobile devices for varying network types (3G, 4G, and Wi-Fi). This comparison shows that selecting an appropriate wireless network for offloading is crucial. We subsequently presented a novel network-aware mobile middleware framework based on reinforcement learning to accomplish energy-efficient offloading in smartphones. Our results show that we can save up to 30% battery power with up to 25% better response time when using our proposed framework compared with a state-of-the-art fuzzy logic-based offloading approach from prior work. As part of future work, researchers can consider the evaluation of a more diverse set of mobile applications and characterizing their bottlenecks, explore new algorithms for low-overhead offloading decision-making on smartphones and other mobile devices

(e.g., wearables), and consider the characterization and use of additional wireless networks for offloading, such as emerging 5G networks.

Acknowledgments This work was supported by the National Science Foundation (NSF), through grant CNS-2132385.

References

1. The Statistical Portal. [Online]. Available: www.statista.com/statistics (2016). Accessed 7 Mar 2022
2. Ali, F.A., Simoens, P., Verbelen, T., Demeester, P., Dhoedt, B.: Mobile device power models for energy efficient dynamic offloading at runtime. J. Syst. Softw. **113**, 173–187 (2016)
3. Gaonkar, S., Li, J., Choudhury, R.R., Cox, L., Schmidt, A.: Micro-blog: sharing and querying content through mobile phones and social participation. In: Proceedings ACM Mobisys, pp. 174–186. ACM (2008)
4. Cuervo, E., Balasubramanian, A., Cho, D.K., Wolman, A., Saroiu, S., Chandra, R., Bahl, P.: Maui: making smartphones last longer with code offload. In: Proceedings ACM Mobisys, pp. 49–62. ACM (2010)
5. Chun, B.G., et al.: Clonecloud: elastic execution between mobile device and cloud. In: ACM EuroSys, pp. 301–314. ACM (2011)
6. Flores, H., Srirama, S.: Mobile code offloading: should it be a local decision or global inference? In: Proceedings ACM Mobisys, pp. 539–540. ACM (2013)
7. Kosta, S., et al.: Thinkair: dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In: Proceedings IEEE INFOCOM, pp. 945–953. IEEE (2012)
8. Flores, H.R., Srirama, S.: Adaptive code offloading for mobile cloud applications: exploiting fuzzy sets and evidence-based learning. In: Proceedings ACM Mobisys, pp. 9–16. ACM (2013)
9. Khairy, A., et al.: Smartphone energizer: extending smartphone's battery life with smart offloading. In: IEEE IWCMC, pp. 329–336. IEEE (2013)
10. Banga, G., Crosby, S., Pratt, I.: Trustworthy computing for the cloud-mobile era: a leap forward in systems architecture. IEEE Consum. Electron. Mag. **3**(4), 31–39 (2014)
11. Corcoran, P., Datta, S.K.: Mobile-edge computing and the internet of things for consumers: extending cloud computing and services to the edge of the network. IEEE Consum. Electron. Mag. **5**(4), 73–74 (2016)
12. Khune, A., Pasricha, S.: "Mobile network-aware middleware framework for energy efficient cloud offloading of smartphone applications", IEEE. Consum. Electron. **8**(1), 42 (2019)
13. Donohoo, B., Ohlsen, C., Pasricha, S., Anderson, C., Xiang, Y.: Context-aware energy enhancements for smart mobile devices. IEEE Trans. Mob. Comput. **13**(8), 1720–1732 (2014)
14. Donohoo, B., Ohlsen, C., Pasricha, S.: A middleware framework for application-aware and user-specific energy optimization in smart mobile devices. J Pervasive Mob. Comput. **20**, 47–63 (2015)
15. Donohoo, B., Ohlsen, C., Pasricha, S., Anderson, C.: Exploiting spatiotemporal and device contexts for energy-efficient mobile embedded systems. In: IEEE/ACM Design Automation Conference (DAC). IEEE (2012)
16. Tiku, S., Pasricha, S.: Energy-efficient and robust middleware prototyping for smart mobile computing. In: IEEE International Symposium on Rapid System Prototyping (RSP). IEEE (2017)
17. Pasricha, S., Doppa, J., Chakrabarty, K., Tiku, S., Dauwe, D., Jin, S., Pande, P.: Data analytics enables energy-efficiency and robustness: from mobile to manycores, datacenters, and networks. In: ACM/IEEE International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS). IEEE (2017)

18. Pasricha, S., Raid Ayoub, M., Kishinevsky, S.K., Mandal, U.Y.O.: A survey on energy management for mobile and IoT devices. IEEE Des. Test. **37**(5), 7–24 (2020)
19. Kumar, K., Lu, Y.-H.: Cloud computing for mobile users: can offloading computation save energy? Computer. **43**(4), 51–56 (2010)
20. Flores, H., Hui, P., Tarkoma, S., Li, Y., Srirama, S., Buyya, R.: Mobile code offloading: from concept to practice and beyond. IEEE Commun. Mag. **53**(3), 80–88 (2015)
21. Li, J., Bu, K., Liu, X., Xiao, B.: Enda: embracing network inconsistency for dynamic application offloading in mobile cloud computing. In: Proceedings ACM SIGCOMM, pp. 39–44. ACM (2013)
22. Sivakumar, A., et al.: Cloud is not a silver bullet: a case study of cloud-based mobile browsing. In: Proceedings ACM Mobisys, p. 21. ACM (2014)
23. Monsoon Solutions Inc. official website. [Online]. Available: http://www.msoon.com/LabEquipment/Power-Monitor (2016). Accessed 9 Nov 2016
24. Matrix Calculator app at Google play store. [Online]. Available: https://play.google.com/store/apps/details?id=ru.alex-anderskokov.matrix&hl=en (2016). Accessed 7 Mar 2022
25. Amazon web services (AWS). [Online]. Available: https://aws.amazon.com/ (2016). Accessed 7 Mar 2022
26. Mozilla Firefox. [Online]. Available: https://www.moz-illa.org/en-US/firefox/android (2016). Accessed 7 Mar 2022
27. Puffin Web Browser. [Online]. Available: http://www.puffinbrowser.com/ (2016). Accessed 7 Mar 2022
28. Zipper Android App. [Online]. Available: https://play.google.com/store/apps/details?id=org.joa.zipperplus&hl=en (2016). Accessed 7 Mar 2022
29. Ezyzip: The simple online zip tool. [Online]. Available: http://www.ezyzip.com/ (2016). Accessed 7 Mar 2022
30. Google Translate Android App. [Online]. Available: https://play.google.com/store/apps/details?id=com.google.android.apps.translate&hl=en (2016). Accessed 7 Mar 2022
31. Speak & Translate iOS app. [Online]. Available: https://itunes.apple.com/us/app/speak-translate-free-live/id804641004?mt=8 (2016). Accessed 7 Mar 2022
32. Fuld – Torrent Downloader app. [Online]. Available: https://play.google.com/store/apps/details?id=com.delphicoder.flud&hl=en (2016). Accessed 7 Mar 2022
33. Kelenyi, I., Nurminen, J.K.: Cloudtorrent-energy-efficient bittorrent content sharing for mobile devices via cloud services. In: Proceedings IEEE CCNC, pp. 1–2. IEEE (2010)
34. Altamimi, M., et al.: Energy cost models of smartphones for task offloading to the cloud. IEEE Trans. Emerg. Topics Comput. **3**(3), 384–398 (2015)
35. Li, H., Liu, D., Wang, D.: Integral reinforcement learning for linear continuous-time zero-sum games with completely unknown dynamics. IEEE Trans. Autom. Sci. Eng. **11**(3), 706–714 (2014)
36. Abouheaf, M., et al.: Multi-agent discrete-time graphical games and reinforcement learning solutions. Automatica. **50**(12), 3038–3053 (2014)
37. Anderson, C.: Introduction to machine learning. Website [Online]. Available: https://www.cs.colostate.edu/~anderson/cs545/index.html/doku.php (2016). Accessed 7 Mar 2022

Part II
Cyber-Physical Application Use-Cases for Embedded Machine Learning

Context-Aware Adaptive Anomaly Detection in IoT Systems

Rozhin Yasaei and Mohammad Abdullah Al Faruque

1 Introduction

Over the last decade, IoT has grabbed substantial attention due to advancements in computation and communication, and it is utilized in many applications such as smart home, automotive, and medical aid. The rapid growth of IoT has raised concerns about the security and reliability of these systems. There are a tremendous amount of work in the literature that focuses on various aspects of IoT systems such as communication network [24, 38], hardware security [5, 15, 22, 23], or software security [3, 34, 40, 41]. However, the physical layer of IoT as a cyber-physical system (CPS) is overlooked. To ensure the security of CPS systems, in addition to a bottom-up security attitude, a holistic approach is required [8–10, 12].

The ultimate goal of an IoT system is to control the environment and maintain it in the desired state. In order to explain the important role of sensors in fulfilling this goal, we categorize IoT systems under two categories, as depicted in Fig. 1: (i) a *closed-loop control* system and (ii) a *monitoring system*. On the one hand, a *closed-loop control system* consists of three major components: (i) sensors; (ii) controller; and (iii) actuators (see Fig. 1a). The sensors monitor the system and send the status to the controller, which processes the sensor readings, decides how to react, and sends the control signals to the actuators to maintain the state of system and environment.

On the other hand, *monitoring systems* mainly contain sensors that measure numerous parameters in the system and provide the user with information to take proper action (see Fig. 1b). Although a *monitoring system* cannot directly manipulate the environment, it informs a supervising user of events that happen

R. Yasaei (✉) · M. A. Al Faruque
University of California Irvine, Irvine, CA, USA
e-mail: ryasaei@uci.edu; alfaruqu@uci.edu

S. Pasricha, M. Shafique (eds.), *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing*, https://doi.org/10.1007/978-3-031-40677-5_8

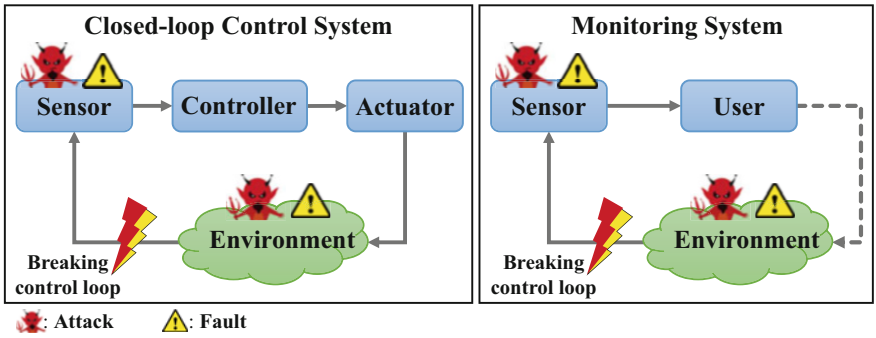


Fig. 1 Two categories of IoT systems; (**a**) closed-loop control system, and (**b**) monitoring system

in the system, and the user controls the system manually. Thus, a monitoring system is eventually a part of a control loop.

In both categories, sensors are an essential component of the control loop since sensor measurements determine the action that is needed to maintain the system in the desired state. Malfunction or manipulation of a sensor can break the control loop [4] and, consequently, disrupt the services offered by the IoT system. Fault in a sensor device leads to the appearance of anomalous values in its readings, whereas not all anomalies in sensor measurements indicate sensor breakage because an unexpected event in the environment may cause an anomaly as well. Observing the possible anomalies in an IoT system, we present a classification of anomalies that facilitates identification of the anomaly's source:

- **Environmental Anomaly (EA):** The environment is the area that surrounds the sensor, and the sensor measures its physical properties. Any anomaly in the environment affects the measurements of the sensor and disrupts it. An EA may occur as a result of malicious activities or unexpected incidents in the environment.
- **Sensing Device Anomaly (SDA):** When the operation of a sensor is corrupted, its measurements do not follow the same pattern, and an SDA is observed. This corruption occurs because of either security or reliability issues. For instance, [1, 46] discuss some attacks on the physical layer.

Current anomaly detection methods model the normal behavior of a device [7, 13, 31, 32] and label any deviation from expected behavior as an anomaly. Most of the works concentrate on anomaly detection in the network layer of IoT systems [20]. In spite of reasonable performance in network intrusion detection, these methods have a high rate of false alarms when used with sensor signals. They misinterpret the environmental variation in the sensors measurements as an SDA and disregard the potential information encoded in the relation between the system and the physical world, known as the **context** of the system (refer to Sec. 3.1 for the definition of context). Conventionally, context-aware methods are applied to a variety of applications [2], and recently, these methods are used to secure the authentication of co-located devices [17, 35, 36, 42].

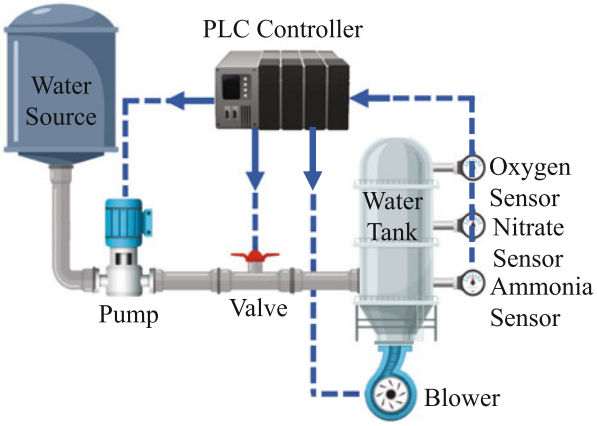


Fig. 2 Schema of the wastewater plant

In this chapter, we propose an adaptive data-driven model for unsupervised anomaly detection in IoT systems based on the sensor measurements. The model monitors the system to detect anomalies, identifies the type of anomaly (SDA or EA), and locates them [44]. To this end, we develop an algorithm to extract the patterns in sensor signals and generate the context of system. Then, we associate the sensors from different modalities based on the context and cluster the sensors with similar behavior. We develop our customized recurrent neural network (RNN), followed by a consensus algorithm to detect and localize anomalies. The consensus algorithm checks the consistency between sensors in each cluster and determines the type of anomaly. An IoT system has a dynamic structure that is open to changes, such as adding new nodes, removing the existing ones, or updating the framework and protocols. In order to address the variation in IoT systems over time, our model is designed to be adaptive and update itself.

1.1 Motivational Example

As a real-world IoT system, we study the environmental training center wastewater plant in Riccione [14]. The primary purpose of wastewater treatment is the elimination of nitrate. Nitrate contamination is a severe environmental problem because it can exhibit toxicity toward aquatic life, present a public health hazard, and affect the suitability of wastewater. In the treatment process, the wastewater is pumped to the tanks, which are equipped with sensors to monitor the concentration of oxygen, ammonia, and nitrate in the water. The actuators, such as blowers and valves, are controlled by a Programmable Logic Controller to adjust the level of chemicals (Fig. 2). Given the importance of the nitrate level, anomaly detection is applied to detect abnormal changes. Consider two scenarios with anomalous rise in

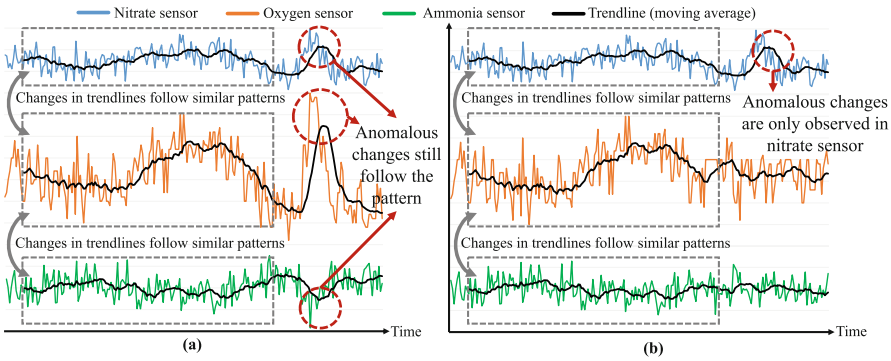


Fig. 3 Synthetic sensor signals in the (**a**) first scenario (EA), (**b**) second scenario (SDA)

nitrate level; in the first scenario, environmental changes alter the water temperature, which affects the chemical reactions in the water tank (Fig. 3a, an example of EA). In the second scenario, the nitrate sensor is broken or manipulated by an attacker (Fig. 3b, an example of SDA). The current anomaly detection methods rely solely on nitrate sensor data, whereas the validity of its data is questionable. Thus, they cannot find the source of anomaly and discriminate EA and SDA.

A recent study [14] analyzes the sensors of this wastewater plant and reveals the correlation between ammonia, oxygen, and nitrate sensor data. More specifically, when the rise in oxygen density reaches a certain threshold, the ammonia concentration decreases, and the nitrate concentration increases. Further investigation reveals the scientific rationale for this correlation; oxygen triggers the chemical reaction, which affects the ammonia and nitrate concentration. By considering this relationship, it is possible to validate sensor signals. In the first scenario, the incident affects all sensors. Despite irregularities in the sensor signals, they are consistent with each other. Thus, we can conclude that the integrity of the sensors' data is not compromised. In the second scenario, the anomaly in the nitrate sensor data is inconsistent with the patterns of other sensor signals. It indicates that the cause of the abnormality is fault or attack. This type of relationship between sensors is not limited to this wastewater plant, and it is observed in many IoT systems due to the availability of many heterogeneous sensors.

1.2 Threat Model

The proposed methodology aims to detect SDA and EA, which occur due to an unexpected incident in the environment, reliability issue, or security breakage. Accidental damage, degradation, and defects are examples of plausible reliability problems that cause unintended device malfunctions. In contrast, the security breakage scenario involves an attacker who intentionally exploits the vulnerabilities

in the system. In this threat model, the adversary has access to the sensor device and fiddles with it to inject fault, alter functionality, or deny its service. As another possible scenario, the attacker can control the communication channel and send faulty signal to the controller as sensor measurements. The model can detect anomalies in a standalone sensor, but to distinguish between SDA and EA in a sensor, it should be associated with at least two other sensors. To deceive this method, the attacker should be able to discover how sensors are clustered, learn the correlations and patterns in the sensors' signals, and manipulate them in a way that imitates the same correlation as before. It means that in addition to sensors, the attacker should have full access to the clustering layout of sensors and the trained anomaly detection model. It is assumed that the attacker does not have these privileges.

1.3 Research Challenges

Anomaly detection in the IoT sensors is challenging due to the following reasons [11]:

- The IoT data are multi-variant time-series data that are collected from a heterogeneous network of sensors with different modalities, data dimensions, sampling rates, specifications, and locations.
- Low-cost and resource-constrained sensors are usually sensitive to noise, and deployment of them in IoT systems affects the quality of data.
- Due to a lack of prior knowledge about possible anomalies and scarcity of anomalous observations, there is not enough labeled anomalous data available, and conventional supervised machine learning techniques are not applicable.
- IoT systems have dynamic characteristics that may be altered over time because of environmental changes, human interaction, mobility of devices, and updating firmware or software. Consequently, a static model fails to imitate the system in the long term.

1.4 Contributions

To the best of our knowledge, this is the first context-aware anomaly detection method for IoT systems. Our novel contributions to address the aforementioned challenges are summarized below:

- **Context-aware sensor association algorithm**: We develop a multi-modality clustering method to associate sensors that experience similar contextual variation.

- **Consensus-based strategy for unsupervised anomaly detection**: We design a methodology to pinpoint the anomalies without reliance on prior knowledge about possible anomalies.
- **Adaptive data-driven model**: Our proposed anomaly detection model is periodically updated at runtime to adapt itself to new states caused by variations in the system.

2 Related Works

General anomaly detection algorithms can be classified into the following main categories [14]:

Statistical or Probabilistic Methods: These methods create a statistical or probabilistic model based on history data, which represents normal behavior [16, 39]. Upcoming observation is then compared with this model, and it is marked as an anomaly if it is statistically unlikely, or the probability of such observation is low.

Proximity Methods: These methods compute distances between data points to differentiate between anomalous and normal data. Two well-known techniques that fall in this category are the *local outlier factor* [6] and *clustering* [18] methods.

Predictive Methods: In these methods, the anomaly detection problem is converted to obtain an accurate sequence prediction algorithm that captures the recent and long-term trends in data sequences and reproduces them to predict future measurements. Afterward, the predictions are compared with the new observations to spot deviations from expected normal behavior. Recurrent neural networks (RNN) are capable of capturing the relationship between measurements over time because the feedback loops in the hidden layer of RNN can imitate memory.

Long short-term memory (LSTM) layer was introduced in 1997 by Hochreiter and Schmidhuber [19] to overcome the shortcomings of RNN. It has gained a lot of attention lately because of its high accuracy in sequence prediction [7, 31, 32]. Conv-LSTM encoder–decoder is one of the neural network architectures that is used in the literature to enhance sequence prediction performance [21, 25, 26, 43, 45]. It contains convolutional layers to extract the essential features of input sequences and LSTM layers to perform the sequence prediction based on the features. Then, the anomaly is identified based on the reconstruction error of the model. LSTM–LSTM encoder–decoder [33, 37, 47] is another popular architecture that follows a similar strategy, but it utilizes LSTM layers instead of convolutional layers for feature extraction.

Our methodology inherits the advantages of both probabilistic and predictive methods. We implement and compare the Conv-LSTM and LSTM–LSTM encoder–decoder as our predictive models. Then, the reconstruction error, derived from the difference between real and predicted values, is modeled by a multivariate Gaussian estimators to detect the anomaly.

3 Anomaly Detection Methodology

Our proposed methodology (see Fig. 4) detects SDA and EA in an IoT system to ensure sensing devices operate as they are expected.

3.1 Context Generation

The context of a system is defined as an abstraction formed by extracting features from system circumstances and individual element constructs [27]. It describes the condition in which the system is operating and affects the outcome of the system. The first step for obtaining our context-aware data-driven model is to generate the context of the system by encoding its physical properties. Understanding and transforming this information such that it can be mathematically described is called **context generation**. Following the strategy presented by Sadeghi et al. in [35], we convert all sensor signals into binary fingerprints regardless of their modality. The procedure of fingerprint generation has the following steps:

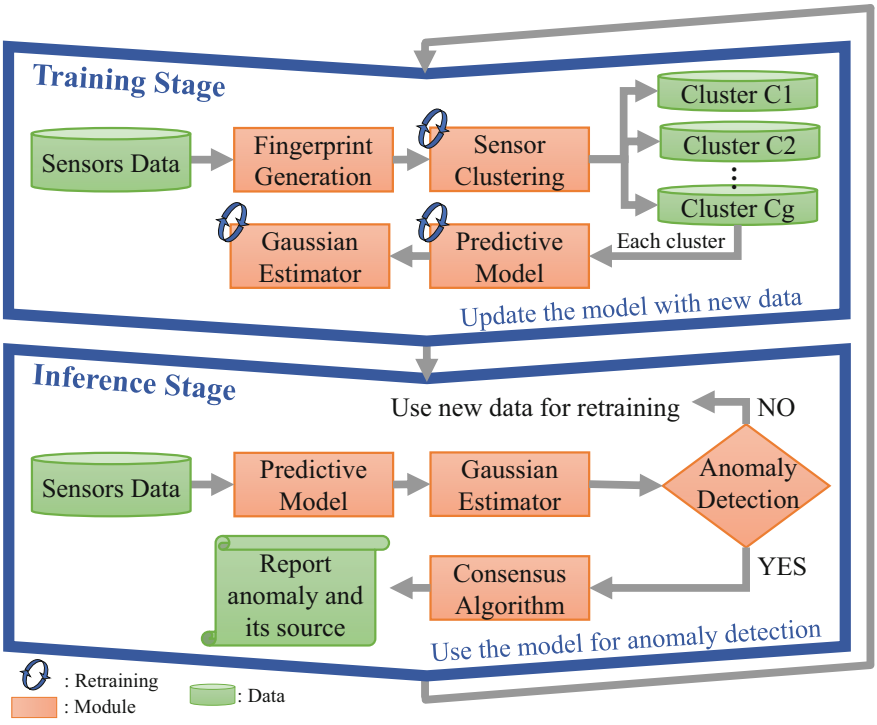


Fig. 4 The architecture of our methodology in the training and inference stage

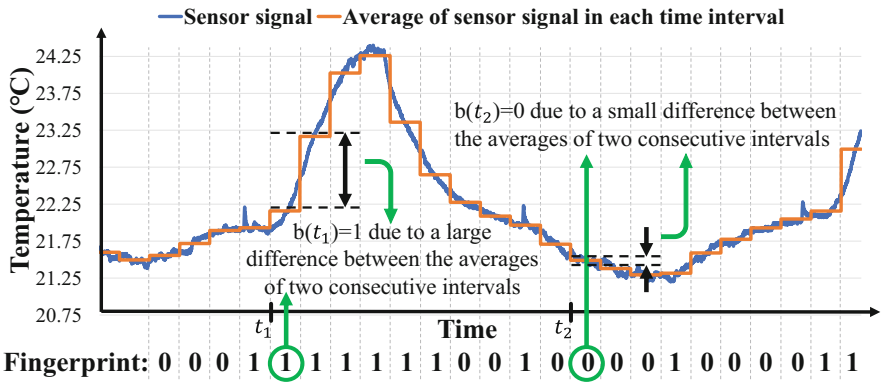


Fig. 5 Extracting the fingerprint of a temperature sensor

Step1: Each sensor continuously monitors the environment by taking a measurement each z seconds. The value z depends on the sampling rate of sensor and may vary for different sensors. In a time window of q seconds from timestamp t, the sensor records $v = [q/z]$ measurements and forms a snapshot vector $S_t = (s^t, s^{t+z}, \ldots, s^{t+z(v-1)})$.

Step2: The ϵ_{rel} is a pre-defined threshold that controls the amount of variation that is said to conform a change. The values obtained in a snapshot are averaged, and the variation bit $b(t)$ is calculated as follows:

$$\overline{S}_t = \frac{1}{v}\sum_{s \in S_t} s, \; b(t) = \begin{cases} 1, & \text{if } \left|\frac{\overline{S}_{t+z} - \overline{S}_t}{\overline{S}_t}\right| > \epsilon_{rel} \\ 0, & \text{o.w.} \end{cases}$$

Step3: Finally, a sequence of $k+1$ consecutive snapshots $seq(t, t+kz) = (\overline{S}_t, \overline{S}_{t+z}, \ldots, \overline{S}_{t+kz})$ has an associated fingerprint $F(seq(t, t+kz)) = (b(t), b(t+z), \ldots, b(t+(k-1)z))$. The fingerprints of all the sensors with different sampling rates have the same length because each snapshot is the average of sensor measurements in a particular time interval. Figure 5 illustrates the process of generating the fingerprint of a temperature sensor.

3.2 Sensor Association

Although each sensor's measurements differ based on its modality and physical location, the sensors that are affected by the same event follow similar patterns in their fingerprints. Based on this observation, we develop a sensor association algorithm that comprises two primary steps: I) pattern extraction and II) sensor clustering.

In the first step, we split each fingerprint into smaller sub-sequences and cluster the sub-sequences of different sensors that have a similar binary pattern. For simplicity, assume that $F_i = F(seq(t, t + zk)_i)$ represents the fingerprint of the sensor i, which is split into d smaller sub-sequences ${f_i}^j$ as follows:

$$F_i \longrightarrow \left({f_i}^1, {f_i}^2, \ldots, {f_i}^d\right),\ d = \frac{k-o}{l-o},$$

where l and o are the hyperparameters that determine the sub-sequences length and their overlap accordingly. Afterward, our clustering algorithm is performed on the sub-sequences of index j ($j \in [1, d]$) of all sensors $({F_1}^j, {F_2}^j, \ldots, {F_n}^j)$ to group the ones with similar binary patterns. Hence, it assign a pattern number $p_i^j \in \{0, 1, \ldots, p_{max}^j\}$ to ${f_i}^j$. Next, the clustering is repeated for index $j + 1$, and after d iterations, all sub-sequences are clustered. Notice that p_{max}^j, which represents the number of clusters for index j, may vary for different index values. Eventually, for each sensor, the pattern numbers form a *pattern history vector* $P_i = ({p_i}^1, {p_i}^2, \ldots, {p_i}^d)$.

In the second step, one final clustering is performed on the given set of sensors pattern history P_i to determine the sensors cluster layout $C = c_1, c_2, \ldots c_g$, where c_i represents clusters and g is the number of sensor clusters. The sensors with similar contextual variations exhibit the same patterns in many sub-sequences, and we cluster them together. An example of the sensor association procedure is demonstrated in Fig. 6. In this example, the final clustering groups the first and third sensors together.

All the mentioned clustering processes are done using our customized clustering algorithm that minimizes the distance between data points in the same cluster, the *intra-cluster distance (IC)*, and maximizes the distance among data point of one cluster from other cluster data points, the *inter-cluster distance (OC)*. The distance matrices are defined as follows:

$$IC = \overline{IC_i},\ IC_i = \frac{1}{|c_i|} \sum_{m,k \in c_i} Hamming(P_m, P_k)$$

$$OC = \overline{OC_{i,j}},\ OC_{i,j} = \min_{m \in c_i, k \in c_j} \{Hamming(P_m, P_k)\},$$

where c_i and P_m represent a cluster and the pattern history of sensor m, respectively. Algorithm 1 is a pseudocode that elaborates on our clustering algorithm. Our clustering has the following properties:

- It can be applied to data with string type because the distance matrices are based on the Hamming distance function, which calculates the number of non-matching bits.
- The number of clusters is automatically tuned. Initially, clustering is performed with an upper bound of the number of clusters. Afterward, the algorithm

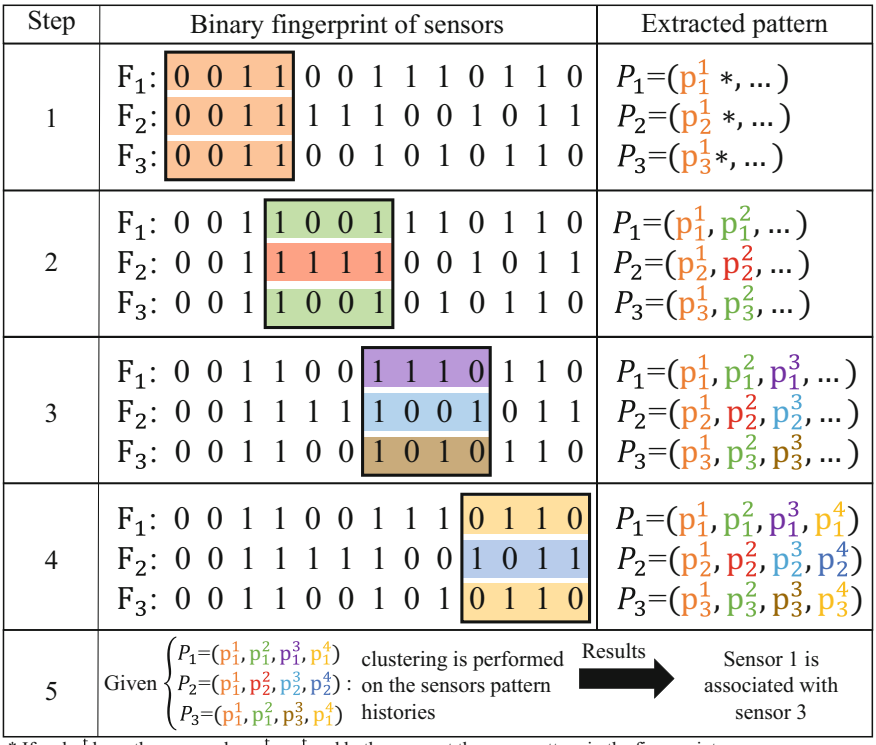


Fig. 6 The procedure of extracting the patterns in sensor signals and clustering them

automatically removes the nodes that are not close to any cluster and eliminates clusters with two nodes to reach optimum value for the number of clusters.

After sensor association, we evaluate the system to ensure that there is no standalone sensor that is not clustered. A standalone sensor is vulnerable because it is not related to any group of sensors that can verify its proper operation. In this case, the anomaly detection still can be applied to the independent sensor individually, but the SDA and EA are indistinguishable. The user is warned about this vulnerability in sensors and can resolve the issue by adding more sensors to the system.

3.3 Predictive Model

The next module of our methodology is the predictive model that predicts the future measurements of sensors according to the clustering layout and history of

Algorithm 1: Customized clustering algorithm for extracting patterns in sensor fingerprints and sensor association

Input: Fingerprints: $F \in IR^{n \times k}$, number of sensors: n, sub-sequence length: l, overlap: o
Output: Cluster layout $C = \{c_1, c_2, \ldots, c_g\}$
Initialize $d = \frac{k-o}{l-o}$
Initialize $p_{max} =$ max # of patterns in sub-sequence
Initialize $iter_{max} =$ max # of iterations
Initialize the center of clusters randomly
foreach $j \in \{1, 2, \ldots, d\}$ **do**
  foreach $i \in \{1, 2, \ldots, n\}$ **do**
    Split fingerprint F_i to obtain sub-sequences ${f_i}^j$;
  Clustering:
  foreach $iter \in \{1, 2, \ldots, iter_{max}\}$ **do**
    foreach $i \in \{1, 2, \ldots, n\}$ **do**
      $p_i^j = Argmin_{x \in C} Hamming(f_i^j, center(x))$;
    foreach $x \in \{1, 2, \ldots, p_{max}\}$ **do**
      $center(x) = mean(\{f_i^j | p_i^j = x\})$;
    if *no changes in* $center(x)$ **then**
      break;
  foreach $x \in \{1, 2, \ldots, p_{max}\}$ **do**
    Calculate inter-cluster (OC) metrics;
    if $Hamming(F_i^j, center(p_i^j) > OC$ **then**
      Remove F_i^j from cluster p_i^j;
      Add F_i^j in *unclustered* nodes;
      Update p_{max}^j;
    if $|c_i| < 3$ **then**
      Remove cluster x;
      Update p_{max}^j;
    Add clusters to pattern histories P_i;
Perform the clustering again on pattern histories $P_i, i \in [1, n]$ to associate sensors;
return Sensors Cluster layout $C = \{c_1, \ldots, c_g\}$

measurements. We construct a recurrent neural network (RNN) for each cluster of sensors as the predictive model. As it is depicted in Fig. 7, our RNN comprises LSTM encoder–decoder and dense layers, which encode the features of input sequences of length l_i and predict the future sequences of length l_o based on the encoded features. Sequences of data are derived from the input time-series signals using the sliding window technique. Afterward, the sequences are scaled through a **Min–Max Scaler** before being treated by the encoder because input signals come from multi-modality sensors with different signal ranges. Eventually, we have a set of predictive models $DT = \{M_1, M_2, \ldots, M_g\}$, where g is the number of clusters in the system and M_i represents the model for cluster c_i. Given cluster c_i that contains n_i nodes, the model M_i takes as input a matrix $X_i \in \mathbb{R}^{l_i \times n_i}$ to predict another matrix $Y_i \in \mathbb{R}^{n_i \times l_o}$.

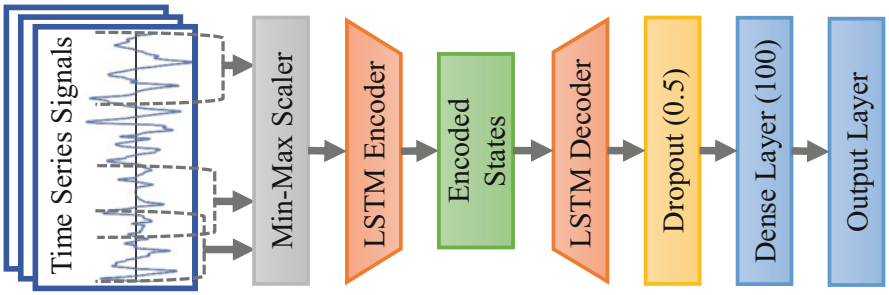


Fig. 7 The architecture of RNN used as predictive model

On top of predictive models, **multivariate Gaussian estimators** are trained to learn the probability of finding a particular error vector. This probability is used to ascertain whether the errors between predictions and real measurements correspond to the system's normal behavior, or an anomaly has occurred. A multivariate Gaussian distributor $G_i = \mathcal{N}(\mu_i, \sigma_i)$ is fitted on the reconstruction error matrix E_i, which is the difference between the real values and the predicted values. The parameters μ_i and σ_i are computed using **maximum likelihood estimation**.

$$\mu_i = \frac{1}{m}\sum_{k=1}^{m} e_{i_j}^k = \bar{e}_{i_j},\ \sigma_i = \frac{1}{m}\sum_{k=1}^{m}(e_j^k - \mu_j)(e_j^k - \mu_j)^T.$$

3.4 Anomaly Detection

In the training stage, the predictive models and estimator modules are periodically used at runtime to infer anomalies. The frequency in which anomaly detection is performed can vary depending on the system specifications. At the runtime, an input measurement x^t is compared with model prediction y^t, and the reconstruction error e^t is calculated. Then, x^t is classified as anomalous if $p^t < \alpha$, where p^t is the probability of obtaining the error vector given by the Gaussian estimator G. α is a pre-defined threshold value, and it is tuned to maximize the F-score of the model.

When anomalous data is discovered, we utilize our consensus algorithm to differentiate between EA and SDA. EA occurs as a result of an incident in the environment. If the EA causes an anomaly in a sensor signal, the correlated sensors are affected by the event and show abnormal changes in their signals. In contrast, SDA influences the sensors individually and results in an anomaly in one or some of the sensors in a cluster. For each cluster, the consensus algorithm inspects the consistency of the sensor behaviors. It uses a voting mechanism to check if all sensors in a cluster agree on the occurrence of an environmental incident. To account

for inertia in the physics of the system, we check the consensus in the time intervals instead of data points.

3.5 Model Adaptation

Due to the high variation in the IoT system and environment, we add the property of **aliveness** to our method, which means the model automatically gets updated to adapt to the system alteration and make more accurate predictions. As Fig. 4 demonstrates, the sensor association, predictive model, and estimator modules are trainable. There are two levels of updating the model: (i) *complete update*, which retrains all trainable modules in order, and (ii) *partial update*, which only retrains the predictor model. These update processes are triggered under three circumstances:

- Change in the number of sensors in the system (either added or removed) triggers *complete update*.
- Each time the sensors send data, the anomaly detection model first validates the new data. Afterward, *partial update* is triggered using the new anomaly-free data.
- If *complete update* is not provoked during a fixed interval of time $t_{retrain}$, it is triggered automatically. This way, the model accounts for changes in the environment, location, and placement. This parameter $t_{retrain}$ can be tuned by the user, depending on how frequently the system layout is changed.

4 Results and Evaluation

4.1 Fog Computing Architecture

Cloud servers are the common and potent available computation resource in IoT systems. However, the bandwidth of network and data transmission become a bottleneck due to rapid expansion of IoT nodes and the quantity of data. As a result, fog computing has emerged, which provides storage, computation, and application services closer to end user with dense geographical distribution [29]. In the fog architecture (Fig. 8), the bottom layer comprises a heterogeneous network of edge nodes with limited resources. The fog nodes in the middle layer collect and process the data from edge devices and communicate to the cloud via the Internet.

Our methodology is fog-empowered, and the developed model for our target IoT system is implemented on a fog node. For the IoT systems with a high density of devices and a massive volume of data, our method is scalable, and it still supports fog computing. Basically, the LSTM encoder–decoder networks are responsible for most of the computation in our method. Thus, instead of training an extensive network for the whole system, we construct a small network

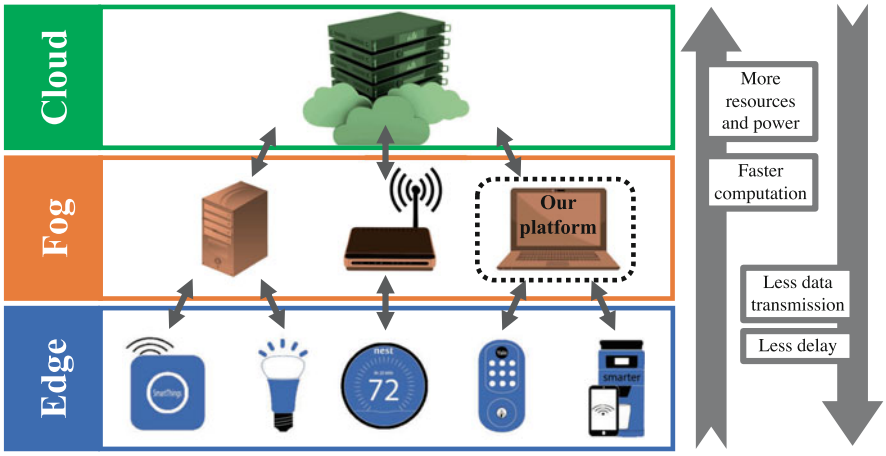


Fig. 8 Fog computing hierarchy in IoT systems

for each cluster of associated sensors that can be distributed between fog nodes. Furthermore, we perform several optimizations to meet resource constraints. In the sensor association, we use the binary fingerprint instead of time-series signals, which lowers storage usage and complicity. The sliding window technique in LSTM network contributes to reducing storage usage as well.

4.2 Experimental Setup

To build and evaluate our methodology, we implement an IoT testbed in our laboratory. Our experimental setup consists of an ad hoc network of multi-modality IoT sensors, a software-defined radio (SDR) connected to an edge computing device, a gateway, and a laptop as fog node. For this particular research, we have used 62 sensors that measure 13 different physical parameters (see Table 1). The acoustic sensor is a wide range microphone with two right and left channels that captures the sound of the space, and its output is amplified and recorded by the handy recorder ZOOM-H6. The Raspberry Pi board, which is directly connected to ZOOM-H6, collects its data and transmits it over the Internet, and this part of the system simulates devices such as Google Home or Alexa. The other sensors are on the low-power embedded boards operated by TinyOS that are equipped with a wireless communication module based on the IEEE 802.15.4 standard. We have implemented the IEEE 802.15.4 standard in the SDR device (USRP-B210) and created a wireless network of sensors in which SDR collects the sensor's data and sends commands to them. SDR is connected to an edge computing device, a Raspberry Pi board, which works as a base station and gathers all data. The base station contains a Wi-Fi module and links the local network of IoT devices to the

Table 1 List of sensors in our experimental setup

Sensor	Sensor board	# of sensors
Temperature	MTS-CM5000	12
Humidity	MTS-CM5000	12
Visible light	MTS-CM5000	12
Infrared light	MTS-CM5000	12
Force and load	MTS-CO1000	2
Tilt	MTS-CO1000	2
Accelerometer	MTS-CO1000	2
Presence detector	MTS-SE1000	2
Magnetic	MTS-SE1000	1
CO_2	MTS-AR1000	1
CO	MTS-AR1000	1
Dust	MTS-SH3000	1
Acoustic	ZOOM-H6	2

Internet through a router. It provides the system with the capability to be monitored in any device that is connected to the Internet by looking up the base station and logging using the password. The algorithms and anomaly detection model are implemented on a Laptop with 8Gb DDR4 RAM and the Intel(R)Core(TM) i5-6300HQ 2.3GHz processor that receives the data from base station and does the computations as a fog node in the IoT system. A powerful router such as Qotom Mini PC Q500G6 has similar capabilities and is capable of running the model at the gateway level. Figure 9 demonstrates the components of our experimental setup and their connections.

4.3 Evaluation

We evaluate our methodology using the data collected by the sensor layout of Sect. 4.2.

4.3.1 Sensor Association Evaluation

One of the contributions of our clustering algorithm is the capability to automatically tune the number of clusters and remove the ones that lack a sufficient number of sensors or have sensors that are far apart regarding the Hamming distance between their fingerprints. Initially, we set the number of clusters to 20 in our system under test, and the algorithm reduces the number to 6. In order to assess the performance of sensor association method, *inter-cluster* and *intra-cluster* distances are calculated for all clusters and plotted in Fig. 10. The notable difference between inter-cluster distance and the intra-cluster distance indicates that related sensors are clustered together, and the clusters are well separated from each other.

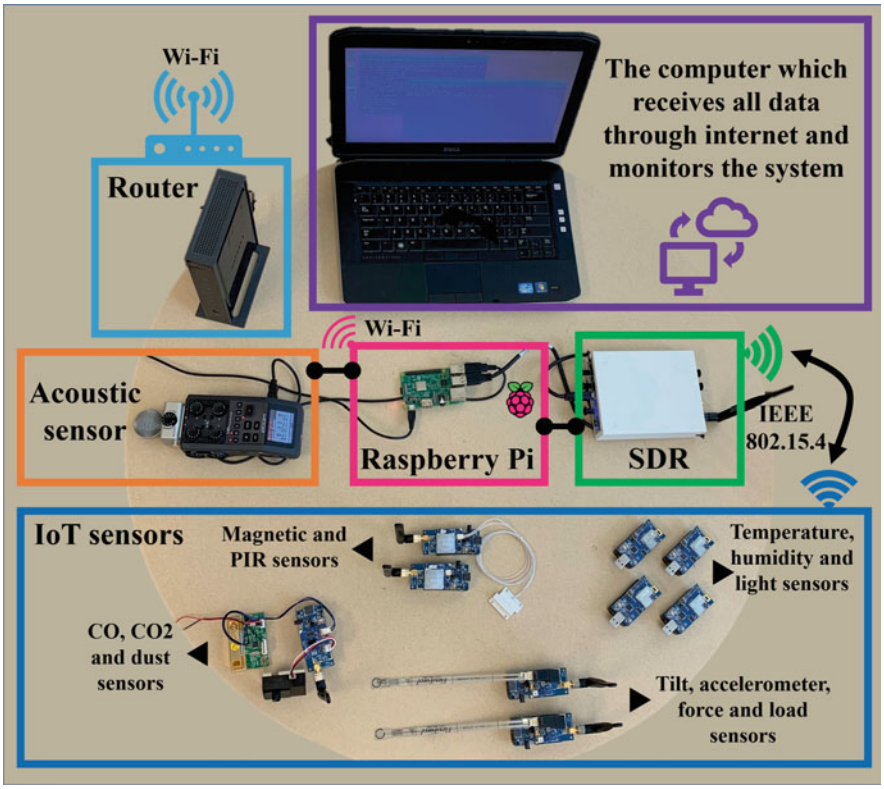


Fig. 9 The scaled-down version of experimental setup

Another validation method used is physical intuition, which explains the relationships among the associated sensors. For example, co-located sensors experience a similar context. Therefore, they are expected to be associated with each other. This intuition supports the result of our algorithm in which co-located humidity, temperature, and light sensors are clustered together, as it is shown in Fig. 12. Another intuition behind the fact is that any physical process may have multi-modality emissions, and the sensors that capture the emission of one incident should be clustered together. It explains the clustering of PIR, vibration sensor (accelerometer and force), magnetic door switch, and acoustic sensor since they all capture the event of entrance through the door. These observations indicate that this strategy is capable of finding relations between sensors with similar contextual variations, further confirmed by the anomaly detection results in the next section.

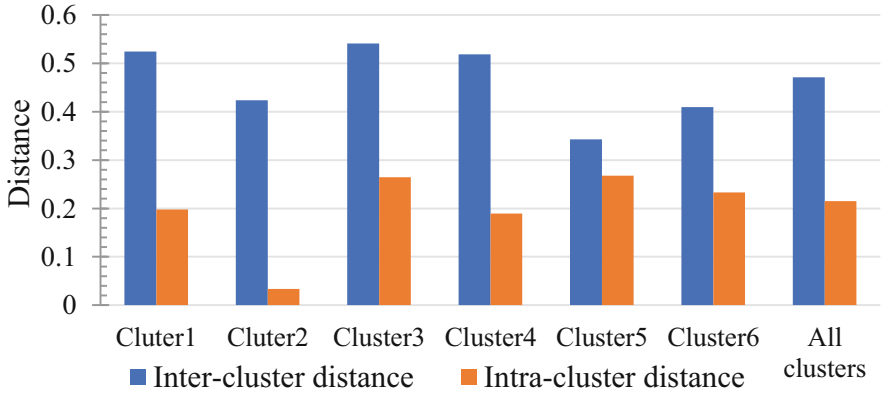


Fig. 10 The inter-cluster and intra-cluster distances of sensor clusters

4.3.2 Anomaly Detection Evaluation

The anomaly detection model is unsupervised, and it is trained only on the normal data and evaluated using a validation dataset with synthetic anomalies. To analyze the results, true positives (*TP*), false positives (*FP*), and false negatives (*FN*) are counted in the results to compute the validation scores. Although the most intuitive performance measure is accuracy, which is the ratio of correctly predicted observation to the observations, it is not appropriate for unbalanced datasets such as anomaly detection where one category representing the overwhelming majority of the data points. Therefore, we use the *Precision(P), Recall(R)*, and $F_\beta\ score$ as performance metrics.

$$P = \frac{TP}{TP + FP}, R = \frac{TP}{TP + FN}, F_\beta score = \frac{P \times R \times (1 + \beta^2)}{\beta^2 \times P + R}.$$

Recall expresses the ability to find all anomalous observations in a dataset, while precision expresses the proportion of the observations our model labels as anomaly, actually is anomalous. $F_\beta\ score$ is the weighted average of precision and recall that provides a better intuition toward both key important capability of model. We implement the current state-of-the-art methods for anomaly detection in time-series data. Due to importance of precision, $F_{0.5}\ score$, which favors precision over recall, is calculated for evaluation in addition to $F_1\ score$. According to the results in Table 2, our methodology has the best performance with highest $F\ scores$ and precision.

Table 2 Comparison with the state-of-the-art methods

Method	Base model	Context-aware	Precision	Recall	$F_{0.5}$ score	F_1 score
IoT-CAD	LSTM	Yes	92%	56%	81%	70%
[32]	LSTM	No	64%	44%	58%	52%
[28]	Conv-LSTM	No	51%	95%	56%	66%
[30]	One-Class SVM	No	89%	25%	60%	39%

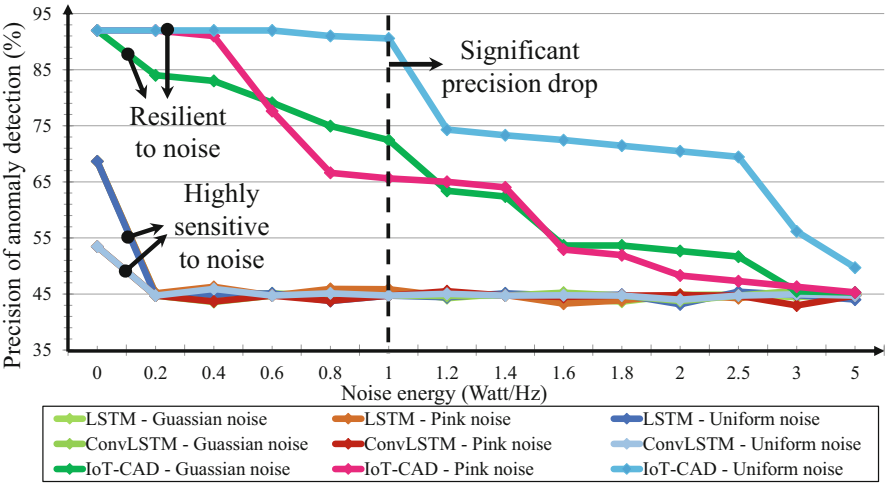


Fig. 11 Evaluating the resilience of different models to pink, Gaussian, and uniform noise signals

4.4 Robustness

We evaluate the robustness of our methodology by adding three different types—pink, Gaussian, and uniform—of noise signals to the sensor measurements and observing the performance of the model. As Fig. 11 indicates, although the precision of anomaly detection is decreased as the noise power increases in all models, our model is more resilient to noise and maintains the high precision.

4.5 Case Study

As a case study, we analyze a cluster of associated sensors, which includes three humidity, temperature, and light sensors located in close proximity. As shown in Fig. 12, the predicted values are very close to the real measurements, which indicates the competency of our method to learn the normal behavior of sensors and predict the future measurements precisely. Furthermore, we observe that the pattern of changes in the sensor signals is similar. As the marked areas of Fig. 12 highlight,

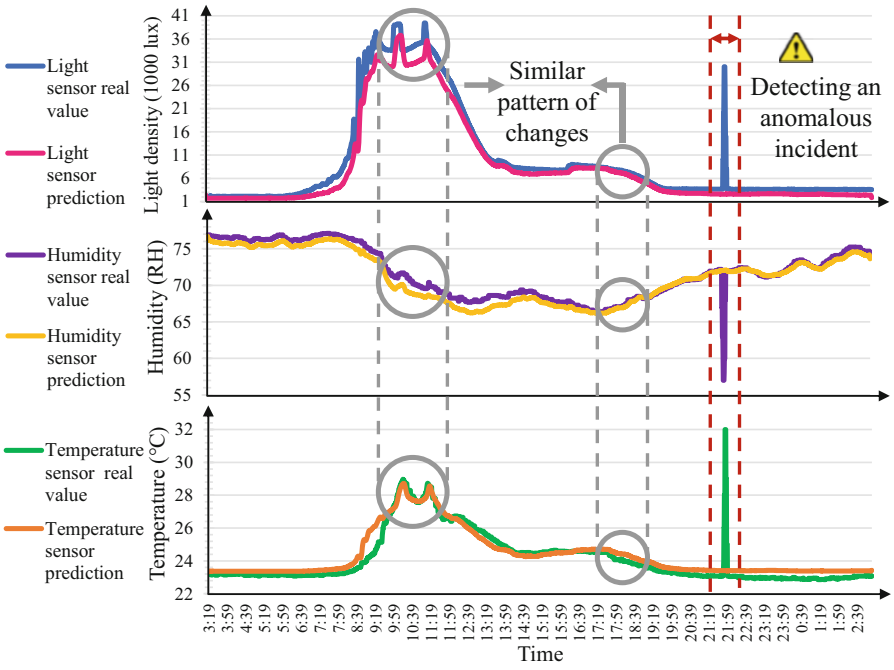


Fig. 12 The real and predicted values of three correlated sensors: light, humidity, and temperature sensors

any drop in the trend of humidity sensor comes with an increase in the trend of other sensors. It confirms the correlation among the sensors as the sensor association algorithm suggests. We simulate a fire incident in the environment as an EA, and the measurements from all sensors show an anomaly.

4.6 Timing Analysis

The timing of method depends on the number of sensors, length of time-series signals, and computing platform that is used to implement the model. We implement our methodology on a fog computing platform and train it on data collected from 62 heterogeneous sensors for 8 days (roughly, 2.3 million data measurements). The training stage starts with the fingerprint generation process, which is repeated for all sensors (62 times). The sensor association process involves 604 times performing clustering to cluster the patterns and then sensors. Eventually, the clustering layout and sensor measurements are used for training the predictive model in an iterative process until the convergence of the model. Although the initial training is time-consuming, it occurs once, and the process of anomaly detection on the new measurements using the trained model only takes 0.532 s, which means it is real

Table 3 Timing results

Process	Recurrence	Time (seconds)
Fingerprint generation	62	2.98
Training sensor association	604	10.54
Training predictive model	1	2472.20
Anomaly detection	Periodic	0.532

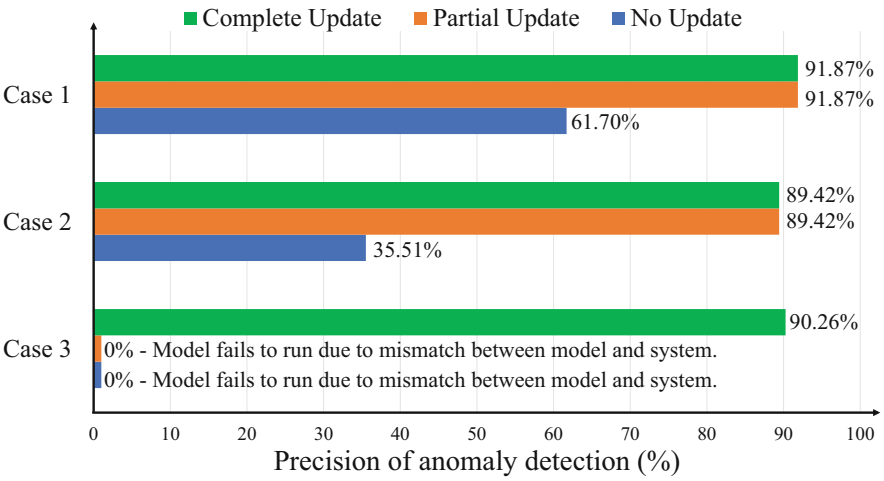


Fig. 13 Analysis of effect of partial and complete update on preserving the performance of model

time in our system under test. As mentioned in Sect. 3.5, the retraining process is triggered under some conditions, but it is faster than initial training since it is limited to new data and does not interrupt the anomaly detection (refer to Table 3).

4.7 Aliveness Assessment

To assess whether updating the model is beneficial for maintaining the model's performance over time, we test it in three scenarios. The first and second scenarios simulate the effect of system degradation or environmental variation over time. In this regard, the measurements of temperature sensors are increased 5 °C in case 1, and 15 °C in case 2 for a day. The third scenario simulates changes in the layout of the IoT system by eliminating a sensor.

The model is initially trained on the original data before the occurrence of scenarios and tested with synthesized data from the cases. In the tests, we examine the effect of the partial update, complete update, and no update on the precision of the model, refer to Fig. 13. According to results, the variation in cases 1 and 2 leads to a significant drop in the precision of the model without updating while

updating the model effectively preserves the high performance because of retraining the predictive model. The third case highlights the advantage of the complete update. Any alteration (add or remove) in the number of sensors in the IoT system changes the input layer dimension of the neural network. Thus, the model cannot perform anomaly detection in case 3 unless the sensor association is retrained to update the layout of sensors, and the model is reconstructed on the new layout. Results confirm that the complete update is successful in maintaining the performance despite removing a sensor. Based on this experiment, it can be concluded that being adaptive is crucial for the models used for IoT.

5 Conclusion

In this chapter, we present a novel context-aware adaptive data-driven model for anomaly detection in IoT systems. It generates context information by encoding the relations among the IoT sensors and clusters the correlated sensors based upon similar patterns and contextual variation. According to the extracted context, a predictive model detects the anomalies, and a consensus-based algorithm determines the type of detected anomalies and pinpoints their source. Our proposed methodology can identify the anomalies with a 92% precision in real time on a fog computing platform. Compared with other methods, it has higher performance and the capability to update itself to account for variations in the system and environment.

References

1. Ahemd, M.M., Shah, M.A., Wahid, A.: IoT security: a layered approach for attacks and defenses. In: 2017 International Conference on Communication Technologies (ComTech), pp. 104–110 (2017). https://doi.org/10.1109/COMTECH.2017.8065757
2. Alegre, U., Augusto, J.C., Clark, T.: Engineering context-aware systems and applications: A survey. J. Syst. Softw. **117**, 55–83 (2016)
3. Altolini, D., Lakkundi, V., Bui, N., Tapparello, C., Rossi, M.: Low power link layer security for IoT: implementation and performance analysis. In: 2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC), pp. 919–925 (2013). https://doi.org/10.1109/IWCMC.2013.6583680
4. Barua, A., Al Faruque, M.: Hall spoofing: a non-invasive DoS attack on grid-tied solar inverter. 29th USENIX Security (2020)
5. Brasser, F., El Mahjoub, B., Sadeghi, A.R., Wachsmann, C., Koeberl, P.: TyTAN: tiny trust anchor for tiny devices. In: 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1–6. IEEE, Piscataway (2015)
6. Breunig, M.M., Kriegel, H.P., Ng, R.T., Sander, J.: LOF: identifying density-based local outliers. In: ACM Sigmod Record, vol. 29, pp. 93–104. ACM, New York (2000)
7. Chauhan, S., Vig, L.: Anomaly detection in ECG time signals via deep long short-term memory networks. In: 2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA), pp. 1–7. IEEE, Piscataway (2015)

8. Chen, H.C., Faruque, M.A.A., Chou, P.H.: Security and privacy challenges in IoT-based machine-to-machine collaborative scenarios. In: Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (2016)
9. Chhetri, S.R., Rashid, N., Faezi, S., Al Faruque, M.A.: Security trends and advances in manufacturing systems in the era of Industry 4.0. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD) (2017)
10. Chhetri, S.R., Faezi, S., Canedo, A., Faruque, M.A.A.: QUILT: quality inference from living digital twins in IoT-enabled manufacturing systems. In: Proceedings of the International Conference on Internet of Things Design and Implementation, pp. 237–248. ACM, New York (2019)
11. Cook, A., Mısırlı, G., Fan, Z.: Anomaly detection for IoT time-series data: a survey. Internet Things J. (2019)
12. Faezi, S., Chhetri, S.R., Malawade, A.V., Chaput, J.C., Grover, W.H., Brisk, P., Al Faruque, M.A.: Oligo-Snoop: a non-invasive side channel attack against DNA synthesis machines. In: NDSS (2019)
13. Filonov, P., Lavrentyev, A., Vorontsov, A.: Multivariate industrial time series with cyber-attack simulation: fault detection using an LSTM-based predictive data model (2016)
14. Giannoni, F., Mancini, M., Marinelli, F.: Anomaly detection models for IoT time series data. arXiv (2018)
15. Halak, B., Zwolinski, M., Mispan, M.S.: Overview of PUF-based hardware security solutions for the Internet of Things. In: 2016 IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS), pp. 1–4 (2016). https://doi.org/10.1109/MWSCAS.2016.7870046
16. Han, M.L., Lee, J., Kang, A.R., Kang, S., Park, J.K., Kim, H.K.: A statistical-based anomaly detection method for connected cars in Internet of Things environment. In: Hsu, C.H., Xia, F., Liu, X., Wang, S. (eds.) Internet of Vehicles – Safe and Intelligent Mobility, pp. 89–97. Springer International Publishing, Cham (2015)
17. Han, J., Chung, A.J., Sinha, M.K., Harishankar, M., Pan, S., Noh, H.Y., Zhang, P., Tague, P.: Do you feel what I hear? enabling autonomous IoT device pairing using different sensor types. In: 2018 IEEE Symposium on Security and Privacy (SP), pp. 836–852. IEEE, Piscataway (2018)
18. He, Z., Xu, X., Deng, S.: Discovering cluster-based local outliers. Pattern Recogn. Lett. **24**(9–10), 1641–1650 (2003)
19. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
20. Hodo, E., Bellekens, X., Hamilton, A., Dubouilh, P., Iorkyase, E., Tachtatzis, C., Atkinson, R.: Threat analysis of IoT networks using artificial neural network intrusion detection system. In: 2016 International Symposium on Networks, Computers and Communications (ISNCC), pp. 1–6 (2016). https://doi.org/10.1109/ISNCC.2016.7746067
21. Jixiang, L.U., Qipei, Z., Zhihong, Y., Mengfu, T.U., Jinjun, L.U., Hui, P.: Short-term load forecasting method based on CNN-LSTM hybrid neural network model. AEPS **43**(8), 131 (2019). https://doi.org/10.7500/AEPS20181012004
22. Lesjak, C., Bock, H., Hein, D., Maritsch, M.: Hardware-secured and transparent multi-stakeholder data exchange for industrial IoT. In: 2016 IEEE 14th International Conference on Industrial Informatics (INDIN), pp. 706–713 (2016). https://doi.org/10.1109/INDIN.2016.7819251
23. Lesjak, C., Druml, N., Matischek, R., Ruprechter, T., Holweg, G.: Security in industrial IoT – quo vadis? e & i Elektrotechnik und Informationstechnik **133**(7), 324–329 (2016). https://doi.org/10.1007/s00502-016-0428-4
24. Lin, W.C., Ke, S.W., Tsai, C.F.: CANN: an intrusion detection system based on combining cluster centers and nearest neighbors. Knowl.-Based Syst. **78**, 13–21 (2015)
25. Lin, T., Guo, T., Aberer, K.: Hybrid neural networks for learning the trend in time series. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, pp. 2273–2279 (2017). https://doi.org/10.24963/ijcai.2017/316, http://infoscience.epfl.ch/record/262447

26. Liu, Y., Zheng, H., Feng, X., Chen, Z.: Short-term traffic flow prediction with Conv-LSTM. In: 2017 9th International Conference on Wireless Communications and Signal Processing (WCSP), pp. 1–6 (2017). https://doi.org/10.1109/WCSP.2017.8171119
27. Longueville, B., Gardoni, M., et al.: A survey of context modeling: approaches, theories and use for engineering design researches. In: DS 31: Proceedings of ICED 03, the 14th International Conference on Engineering Design, Stockholm, pp. 437–438 (2003)
28. Luo, W., Liu, W., Gao, S.: Remembering history with convolutional LSTM for anomaly detection. In: 2017 IEEE International Conference on Multimedia and Expo (ICME), pp. 439–444 (2017). https://doi.org/10.1109/ICME.2017.8019325
29. Lyu, L., Jin, J., Rajasegarar, S., He, X., Palaniswami, M.: Fog-empowered anomaly detection in IoT using hyperellipsoidal clustering. Internet Things J. (2017)
30. Ma, J., Perkins, S.: Time-series novelty detection using one-class support vector machines. In: International Joint Conference on Neural Networks (2003)
31. Malhotra, P., Vig, L., Shroff, G., Agarwal, P.: Long short term memory networks for anomaly detection in time series. In: Proceedings, p. 89. Presses universitaires de Louvain, Louvain-la-Neuve (2015)
32. Malhotra, P., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P., Shroff, G.: LSTM-based encoder-decoder for multi-sensor anomaly detection (2016). Preprint. arXiv:1607.00148
33. Malhotra, P., TV, V., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P., Shroff, G.: Multi-sensor prognostics using an unsupervised health index based on LSTM encoder-decoder (2016)
34. McCune, J.M., Li, Y., Qu, N., Zhou, Z., Datta, A., Gligor, V., Perrig, A.: TrustVisor: efficient TCB reduction and attestation. In: 2010 IEEE Symposium on Security and Privacy, pp. 143–158. IEEE, Piscataway (2010)
35. Miettinen, M., Asokan, N., Nguyen, T.D., Sadeghi, A.R., Sobhani, M.: Context-based zero-interaction pairing and key evolution for advanced personal devices. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 880–891. ACM, New York (2014)
36. Miettinen, M., Nguyen, T.D., Sadeghi, A.R., Asokan, N.: Revisiting context-based authentication in IoT. In: 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), pp. 1–6. IEEE, Piscataway (2018)
37. Qin, Y., Song, D., Chen, H., Cheng, W., Jiang, G., Cottrell, G.: A dual-stage attention-based recurrent neural network for time series prediction (2017)
38. Raza, S., Wallgren, L., Voigt, T.: SVELTE: real-time intrusion detection in the Internet of Things. Ad Hoc Netw. **11**(8), 2661–2674 (2013)
39. Sedjelmaci, H., Senouci, S.M., Al-Bahri, M.: A lightweight anomaly detection technique for low-resource IoT devices: A game-theoretic methodology. In: 2016 IEEE International Conference on Communications (ICC), pp. 1–6 (2016). https://doi.org/10.1109/ICC.2016.7510811
40. Seshadri, A., Luk, M., Perrig, A., van Doorn, L., Khosla, P.K.: SCUBA: secure code update by attestation in sensor networks. In: Workshop on Wireless Security (2006)
41. Seshadri, A., Luk, M., Perrig, A.: SAKE: Software attestation for key establishment in sensor networks. In: Nikoletseas, S.E., Chlebus, B.S., Johnson, D.B., Krishnamachari, B. (eds.) Distributed Computing in Sensor Systems, pp. 372–385. Springer, Berlin (2008)
42. Wan, J., Lopez, A., Faruque, M.A.A.: Physical layer key generation: securing wireless communication in automotive cyber-physical systems. ACM Trans. Cyber Phys. Syst. **3**(2), 13 (2018)
43. Wu, Y., and Tan, H.: Short-term traffic flow forecasting with spatial-temporal correlation in a hybrid deep learning framework. arXiv preprint arXiv:1612.01022 (2016)
44. Yasaei, R., Hernandez, F., Al Faruque, M.A.: IoT-CAD: context-aware adaptive anomaly detection in IoT systems through sensor association. In: 2020 IEEE/ACM International Conference on Computer Aided Design (ICCAD), pp. 1–9. IEEE, Piscataway (2020)
45. Yu, H., Wu, Z., Wang, S., Wang, Y., Ma, X.: Spatiotemporal recurrent convolutional networks for traffic prediction in transportation networks. Sensors **17**(7) (2017). https://doi.org/10.3390/s17071501, https://www.mdpi.com/1424-8220/17/7/1501

46. Zhao, K., Ge, L.: A survey on the Internet of Things security. In: 2013 Ninth International Conference on Computational Intelligence and Security, pp. 663–667 (2013). https://doi.org/10.1109/CIS.2013.145
47. Zhao, Z., Chen, W., Wu, X., Chen, P.C.Y., Liu, J.: LSTM network: a deep learning approach for short-term traffic forecast. IET Intell. Transp. Syst. **11**(2), 68–75 (2017). https://doi.org/10.1049/iet-its.2016.0208

Machine Learning Components for Autonomous Navigation Systems

Kruttidipta Samal and Marilyn Wolf

1 Introduction

Autonomous cyber-physical systems operating in the real world, such as the autonomous drone shown in Fig. 1, typically have a feed-forward information processing pipeline that consists of various modules such as sensing, perception, localization, and mapping, planning, and actuation [1] as shown in Fig. 2. The sensing module contains multiple sensors such as RGB, LiDAR, RADAR, IMU, GPS, etc., to sense data from the real world [2–4]. Perception module processes the data from the sensing unit to extract relevant information such as objects of interest, landmarks, etc. Localization and mapping modules utilize the sensed data and landmarks detected in the perception module to localize the system and optionally build a map of the real world. The planning module utilizes the output of perception and localization modules to plan future actions. The actuation unit converts the output of planning module into signals for the actuators that interact with the real world.

The various modules of an autonomous cyber-physical system operate at different levels of semanticity and dimensionality, e.g., while the early modules such as the perception module may operate on high-dimensional image pixel space, late modules such as the planning module operate on highly semantic but low-dimensional object and landmark space. Traditionally, model-based algorithms were used to extract structure from high-dimensional data, but their accuracy was limited when the system was deployed in the real world. But in recent years machine

K. Samal (✉) · M. Wolf
School of Computing, UNL, Lincoln, NE, USA
e-mail: ksamal2@unl.edu; mwolf@unl.edu

S. Pasricha, M. Shafique (eds.), *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing*, https://doi.org/10.1007/978-3-031-40677-5_9

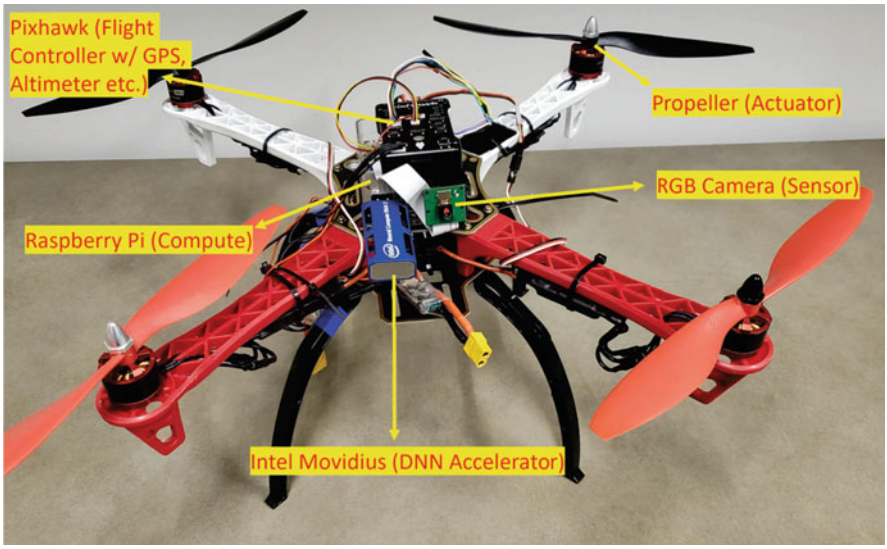


Fig. 1 Autonomous drone setup

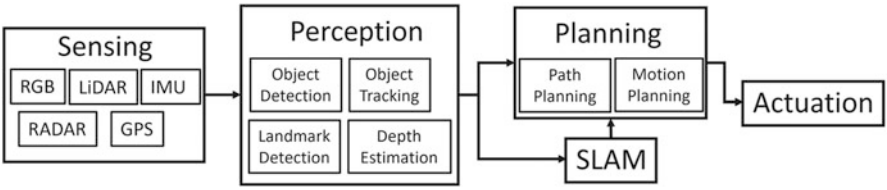


Fig. 2 Information processing pipeline in an autonomous cyber-physical system

learning algorithms have replaced the model-based algorithms as they can learn the features directly from data [5]. When such algorithms are trained on a large volume of data, they can learn robust features that generalize across diverse real-world scenarios. Therefore, adoption of machine learning algorithms has been most pervasive within the perception module of the autonomous cyber-physical system. Additionally, there have been many works that use reinforcement learning for path planning, but most of such works have been tested in simulation and their efficacy in real world is currently under investigation.

The architecture in Fig. 2 shows a clear separation between different modules according to the task performed, but in recent years hybrid modules have been proposed that merge operation of multiple modules to increase accuracy and resource efficiency of the systems. Perception-driven sensing modules (see Sect. 3.5) reduce resource consumption in sensing modules by driving sensor activation and data transfer according to feedback from perception. Similarly, some perception modules

include depth regression unit that directly predicts the 3D map that can be used to simultaneously localize and map (SLAM), thus blurring the boundary between traditional perception and mapping modules. Alternatively, end-to-end learning systems (see Sect. 6) have been proposed that do not have the modular architecture presented above rather such systems directly predict the actuation commands of system from the sensed input such as RGB images. This simplifies the system design but simultaneously obfuscates the system.

2 Sensor Data Fusion

Autonomous cyber-physical systems have multiple sensors depending on the target application. For example, autonomous vehicles have multiple RGB cameras, LiDAR and RADAR sensors to cover the entire field-of-view (FoV), IMU and GPS sensors to aid localization and mapping, and telemetry sensors to support smart connected technologies such as vehicle-to-vehicle (V2V), vehicle-to-cloud (V2C), and vehicle-to-everything (V2X) [6]. Fusing the data collected from multiple sensors is called sensor data fusion [7]. Depending on the stage at which the data sensed by the sensors, the different fusion algorithms can be classified into three categories—early fusion, deep fusion, and late fusion. Figure 3 shows the high-level algorithmic pipeline of different sensor fusion strategies. Early fusion strategies fuse the data from multiple sensors before the processing of the data begins. This requires the sensors to be calibrated and synchronized [8, 9]. Late fusion strategies process the data from different sensors independently and merge the data at the end of the pipeline [10]. This is the most common strategy used in autonomous system software as it has a relaxed constraint on calibration and synchronization of different sensors that typically operate at different frame rate [1, 11]. Finally, deep fusion strategies intermittently merge the features extracted in each sensor processing pipeline [12, 13]. Such strategies have high accuracy as the features extracted in one pipeline can guide the processing of other pipeline thus complementing the information extraction process.

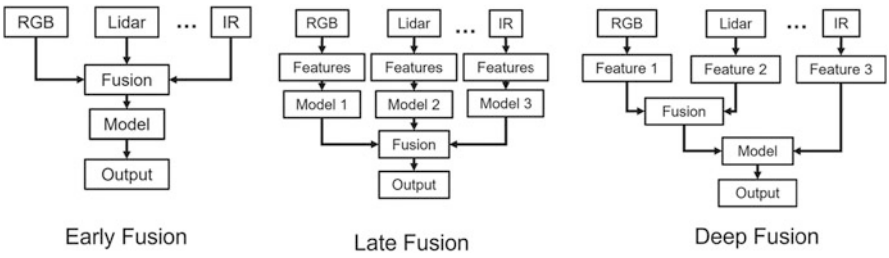


Fig. 3 High-level overview of different sensor fusion strategies. The Model(s) in each strategy can be a hand-crafted or ML based algorithm

3 Perception

Perception module of autonomous cyber-physical systems (ACPS) is responsible for running tasks such as object detection and tracking, path prediction, depth estimation, landmark detection, etc. In modern systems, machine learning algorithms such as deep neural networks (DNNs) are used to solve these tasks. The DNNs are trained on large autonomous driving datasets such as KITTI [2], nuScenes [4], BDD [14], Waymo [3], etc., to detect common on-road obstacles such as pedestrians, bicycles, vehicles, etc. These datasets are collected from vehicles with different sensor configurations, and since there are no standard sensor configuration that is followed by industry or academia, there have been many perception algorithms proposed to process the heterogeneous sensor data. These algorithms can operate on a single modal data such as RGB-only or multi-modal data such as RGB–LiDAR.

3.1 RGB Object Detection

DNN-based RGB object detectors can be divided into three categories depending on their architectures (shown in Fig. 4)—two stage, single stage, or transformer-based. Two-stage detectors such as Faster RCNN [15] have two neural networks, the backbone convolutional neural network (CNN) to extract features from RGB images that is followed by a region proposal network (RPN) that takes the high-level features extracted by the CNN backbone to propose object bounding box hypotheses. One-stage detectors such as YOLO [16, 17] and SSD [18] directly regress the object bounding boxes from within the CNN backbone. Transformer-based object detectors such as DETR [19] replace the RPN module of two-stage detectors with an encoder–decoder module that interprets object proposal as a bipartite matching operation rather than a regression. One-stage detectors have lower computational complexity compared to the two-stage detectors but simultaneously

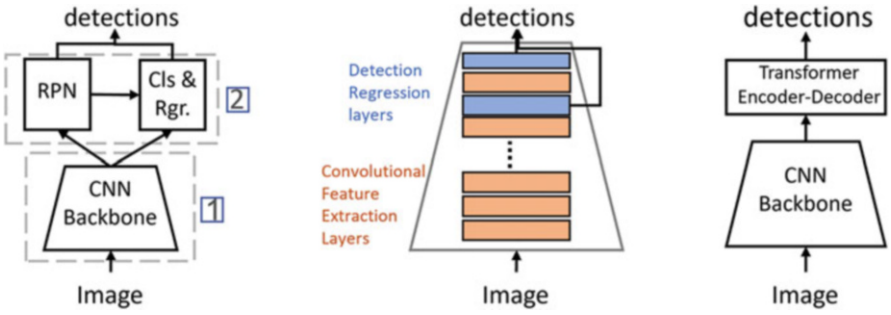


Fig. 4 Object detector architectures. Two-stage detectors (left), one-stage detectors (middle), and transformer-based detectors (right)

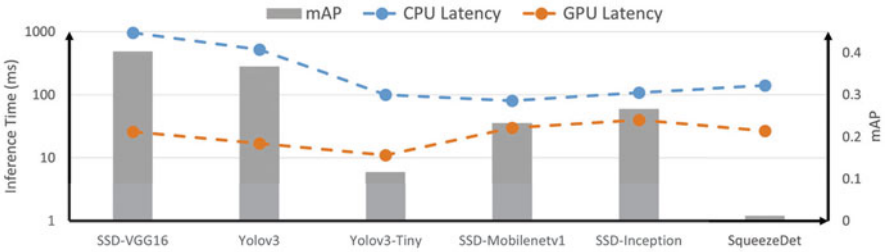


Fig. 5 Performance of DNN-based object detectors on a generic computation platform with a CPU and Nvidia 1080Ti GPU

also suffer from lower accuracy. Typically, autonomous cyber-physical systems use single-stage detectors as they can run at real-time speed (>10 fps) on embedded boards and mid-grade GPUs such as NVidia 1080.

The performance of DNN-based object detectors is typically evaluated on high-performance GPUs such as Nvidia V100 or NVidia Tesla that are meant for datacenters rather than general-purpose computers. Figure 5 shows the latency of several real-time object detectors on a workstation CPU with NVidia 1080Ti GPU from Kim et al. [20]. The SSD and Yolo variants have different backbone CNNs. Compared to YOLOV3, YOLOV3-tiny has a shallower backbone that leads to faster downsampling of the features. This severely affects the small object detection accuracy as the receptive field of intermediate neurons in large. SqueezeDet [21] uses SqueezeNet [22] backbone that removes the fully connected layers of Yolo backbone and uses a fully convolutional network that predicts the bounding box and class probabilities simultaneously. Mobilenets [23] use depthwise separable convolution that breaks the 3D convolution operations within a layer of the DNN into 2D channelwise convolution operation followed by 1D pointwise/channelwide convolution. This reduces the number of FLOPs considerably leading to lower inference time. But it should be noted that such operation increases the number of memory accesses that can be detrimental in memory throughput constrained platforms.

While real-time performance on general-purpose GPU is good for proof of concept and deployment in self-powered systems such as AVs, they are not ideal for battery-powered embedded computing platforms that have limited computing resources and operate under low-power constraints. The detectors showed in Fig. 5 cannot run at real-time speed in low-power embedded platforms such as Raspberry Pi, and even the general-purpose Nvidia 1080Ti GPU consumes 250W of power. Therefore, recently more optimized DNNs have been developed such as SSDLite [24], MobileDet [25], Yolov4 [26], and Yolov5 [27]. The shallow optimized versions of such networks can be deployed in the embedded platforms with edge DNN accelerators such as edge TPUs and Intel Movidius. These accelerators optimize the computational operations within the network by quantizing the weights from 32 bit floating point to 8 bit integer. Simultaneously, they also remove weights

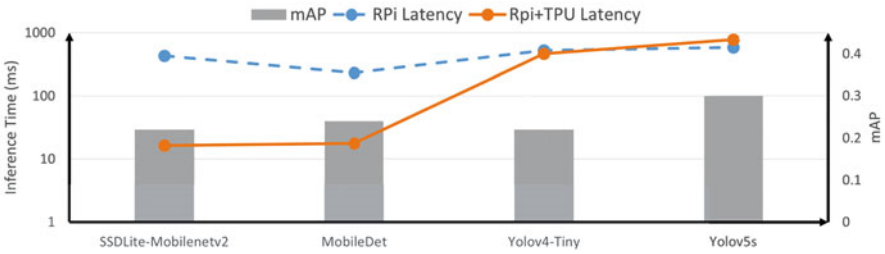


Fig. 6 Performance of DNN-based object detectors on an embedded platform with Raspberry Pi 4 and INT8 TPU accelerator. Tiny-Yolov4 and Yolov5s could not be mapped to the edge TPU

that have low impact on the quality of output thus reducing memory bandwidth and the number of computations. Figure 6 shows the performance of some of the recent light-weight DNN-based object detectors on an embedded platform from Kovács et al. [28]. It can be observed that there are some of these networks that can operate high speed even on low-power resource-constrained embedded platforms.

3.2 LiDAR Object Detection

LiDAR sensors operate on time-of-flight principle. The output of the sensor is a point cloud that captures the 3D map of the world. In addition to position of the objects, the point cloud can also have an intensity channel that captures the reflectance of the objects. Due to the difference in the sensor data, the 2D CNNs developed for RGB images do not work accurately for LiDAR object detection [29, 30]. In the past, geodesic transformation has been proposed that transforms the depth images to capture angle from ground before processing in a 2D CNN [30]. While the transformation improves the accuracy, their efficacy is limited, leading to the development of 3D CNNs that operate on the birds-eye-view (BEV) representation of the point cloud data [29]. But the 3D CNNs had high computational complexity and low accuracy compared to the RGB object detectors. CNNs are not ideal for directly processing point clouds as the filters within the CNN have a regular pattern whereas point cloud has an irregular data representation, e.g., close objects have higher point density than far away objects. Therefore, modern LiDAR object detectors use PointNet [31] to transform the point cloud of the scene into high-level features that are then processed using a CNN. PointNet uses max pooling and multi-layer perceptron to encode point clouds. VoxelNet [32] divided the 3D scene into voxels and used voxel feature extraction (VFE) layer to encode the features of the point cloud in each voxel using a point net. The encoded representation of the scene was then processed using a 3D CNN to detect objects. While Voxelnet had high accuracy, it had high computational complexity. PointPillars [33] proposed to divide the scene into 3D pillars instead, which allowed

Table 1 Performance comparison of LiDAR Object detectors evaluated on the KITTI Dataset [2]. Evaluation metric is Average Precision (AP). Easy, Moderate, Hard represent the occlusion level of the objects

Name	Easy	Moderate	Hard
VeloFCN	40.14	32.08	30.47
MV3D	86.18	77.32	76.33
VoxelNet	89.6	84.81	78.57
PointPillars	88.35	86.1	79.83

Table 2 Performance comparison of different RGB–RADAR fusion-based object detectors in the nuScenes Dataset [4]

Name	AP
RRPN	43
BiraNet	72.3
RADAR-guided visual attention	69

the encoded feature map to be processed using a 2D CNN. This architecture reduced the computational complexity significantly and allowed the object detection to run at real-time speed in an NVidia 1080 GPU. Table 1 shows the performance of different LiDAR object detectors evaluated on the KITTI dataset.

3.3 RADAR Object Detection

The RADAR sensor data represents the depth and velocity information of objects. It is more resilient to sources of interference such as sunlight, fog, and rain that adversely affect the RGB and LiDAR sensors. The raw sensor reading from a RADAR is noisy, and therefore, most RADARs pre-process the sensed data before outputting to the compute engine. But the post-processing step considerably reduces the resolution of the output [34]. This makes it difficult to use machine learning algorithms such as CNNs to process the data as such techniques are not ideal for processing low-dimensional data. Instead, prior works have used RADAR data to augment RGB object detection by using the moving object points from RADAR point cloud to guide the object proposal in the RPN module of a Faster-RCNN RGB object detector [35]. BiraNet projects RADAR points to image plane and extracts the features using a ResNet CNN backbone. The extracted RADAR features are fused with RGB feature maps for object detection. The RADAR-Guided Dynamic Visual Attention [36] uses an RGB network to first generate RoIs and then uses fused RGB–RADAR features of each RoI using a secondary detector for object detection. This reduces complexity of the network compared to BiraNet [37]. Table 2 shows the performance of different RGB–RADAR fusion-based object detectors. Lately, RADAR object detectors have been proposed that operate on the raw RADAR sensor data [38]. While the data is noisy, but the high dimensionality makes it ideal for using CNNs for object detection (Table 3).

Table 3 Performance comparison of RGB–LiDAR object detectors evaluated on the KITTI Dataset [2]. Evaluation metric is average precision (AP)

Name	Easy	Moderate	Hard
MV3D	74.97	63.63	54
AVOD-FPN	76.39	66.47	60.23
FrustumNet	82.19	69.79	60.59
SEG-VoxelNet	86.04	76.13	70.76
VPFNet	88.51	80.97	76.74

3.4 Multi-Modal Object Detection

Multi-modal object detectors use the data collected from multiple sensors operating in different modalities to detect objects [39]. MV3D [40] was one of the first DNN-based RGB–LiDAR-based multi-modal object detectors; it used three feature extraction pipelines one for RGB and two for LiDAR BEV and front view. The LiDAR BEV pipeline was used for creating regions-of-interest (RoIs) that were used to extract features from the LiDAR front view and the RGB pipelines for object localization and classification. AVOD [41] simplified the architecture by using 3D anchors similar to single-shot RGB detectors to directly regress object locations from LiDAR BEV map and RGB images. This allowed AVOD to run in real-time speed (25 fps) in an NVidia 1080 GPU. FrustumNet [42] and F-Convnet [43] used two-stage RGB object detectors to create object proposals in 2D and then projected the 2D bounding boxes on the LiDAR front view map to extract point clouds from the frustum. The extracted frustum was processed using a point net to regress the 3D location of the object. Similarly, SEG-Voxelnet [44] improved the two-stage RGB–LiDAR fusion by using a segmentation network to extract pixel segments from RGB images and then using an improved VoxelNet to extract voxel features from the aligned LiDAR point cloud to regress the 3D location of the objects. Alternatively, VPFNet [45] uses stereo images and LiDAR point cloud to create a dense LiDAR point cloud that is then used for 3D object detection. This architecture is simpler compared to other two track architectures, which leads to low inference latency while maintaining good accuracy.

RGB images capture semantic information of the scene, whereas LiDAR point clouds capture the 3D structure. Therefore, most fusion techniques discussed above try to improve the accuracy of 3D object detection by complementing the LiDAR with semantic RGB information. But these techniques ignore the fact that different modalities can also provide complementary information under different environmental conditions. For example, Infra Red (IR) and LiDAR sensors are better at detecting objects in low light conditions whereas RGB sensors are not. In recent times, there have been some machine-learning-based complementary fusion techniques for multi-spectral data. Guan et al. [46] proposed a two-pipeline network to process RGB and IR data independently and an illumination-aware fully connected network (IFCN) to decide the corresponding weights for fusing the information from the two pipelines. Similarly, Valada et al. [47] and Mees et al. [48]

train a gating network to dynamically adjust the weights for fusion of features extracted from different modalities such as RGB, depth, etc. Such techniques allow the fusion process to adapt to different scenarios.

3.5 Perception-Driven Sensing

The sensor suite of autonomous vehicles has many sensors to sense all the information present in the real world [3, 4]. In some AV systems, sensors of the same modality but with a different sensitivity are deployed to add both precision and redundancy of sensing [3]. For example, in order to get 360° view, multiple RGB cameras can be installed each covering a different section of the scene. Similarly, multiple LiDAR sensors can be installed with each LiDAR having high precision in detecting objects located at different ranges. Such sensor suites have very high energy consumption that limits the mission capability of the system. Furthermore, the high data transfer rate between the sensing and compute engine can cause bandwidth bottleneck that has adverse effect on the reaction time of the system [53]. Therefore, several recent works have proposed creating a feedback from perception to sensing module to control the sensor activation and data transfer. The hysteresis of such systems is based on the state of the perception module and is designed to maximize the utility of the sensor parameters to the end task. The sensor parameters that can be controlled include sensor modality, pixel depth, resolution, etc (Table 4).

Figure 7 shows the meta-architecture of the prior works on closed-loop sensor control. Saha et al. [50] proposed an RGB–IR modality control scheme where RGB is considered the primary/default modality and IR is the secondary modality. An RGB–IR mixed modality object detector is used to detect objects, and the regions where objects are detected are considered RoIs. The modality of the non-RoI regions is switched in the subsequent frames. This process is repeated every frame to create mixed modality images. Mudassar et al. [59] presented the CAMEL adaptive camera system that consists of a 3D stacked visual-IR image sensor with per pixel control. The digital pixel control circuit consists of a light-weight DNN accelerator to run a high-level semantic task. It is used to generate the control signal to control pixel modality (RGB or IR) and spatial and temporal resolution. Mukherjee et al. [60] extended the work by designing a sensor capable of dynamically varying the pixel depth of RoIs based on an external control signal. While the above methods used

Table 4 Performance comparison of different RGB–IR modality control techniques evaluated on the CAMEL dataset [49]

Name	AP	Bandwidth
RGB-only [49]	22.3	61.9
RGB–IR [50]	23.3	52.8
Uncertainty-FP [51]	23.4	53.2
Uncertainty-FN [49]	22.3	53.3
Hybrid [49]	24.4	53.2

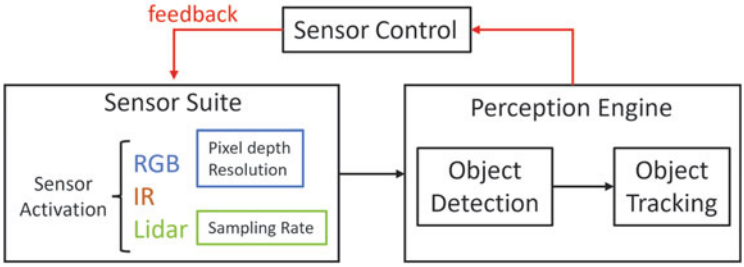


Fig. 7 Perception feedback-driven sensor control. Parameters of sensor control include RGB–IR or RGB–LiDAR multi-modal sensor activation, pixel depth and resolution of RGB sensor, and a sampling rate of LiDAR sensor. The cues used to generate the feedback signal include object detection and tracking

Table 5 Tracking evaluations based on CLEAR MOT metrics from Samal et al. [56]

Name	Rcll↑	Prcn↑	GT	MT	ML	FP↓	FN↓	IDs↓	MOTA↑
RGB-only	72.4	96.0	239	99	16	420	3828	233	67.7
LiDAR-only	77.8	91.5	239	140	15	999	3081	176	69.3
Baseline	83.6	90.2	239	160	7	1269	2276	226	**72.9**
mmMOT [55]*	81.2	86.6	239	165	12	1742	2607	486	65.2
Proposed	81.9	91.9	239	146	6	1004	2521	318	**72.3**

* includes FN and TP ignored in KITTI test server [54] and mmMOT [55] evaluation, such as objects of small heights. $Rcll$=Recall, $Prcn$=Precision, GT=The number of ground truth tracks, MT=Mostly tracked tracks, ML=Mostly lost tracks, FP=False positives, FN=False negatives, IDs=Track ID switches, $MOTA$=Multiple object tracking accuracy

the success of a task, object detection in these cases, to drive the feedback control, but ideally sensor parameters should change according to task failure, for example, IR should be turned on where RGB detection fails. While detecting task failure is a non-trivial task, there have been some prior works that address this issue. Samal et al. [56] proposed a LiDAR sampling strategy in an RGB–LiDAR sensor suite. They used object tracking to detect the regions in the image where there was a temporal inconsistency in object detections from a DNN. Such regions were considered RoIs, and the LiDAR was activated in these regions only. This strategy reduced the overall energy consumption of the entire system. Table 5 shows the performance of the LiDAR sampling strategy compared with methods that used either RGB or both RGB and LiDAR sensors at maximum fidelity. Lee et al. [61] trained a light-weight DNN called Warning Net to predict potential perception task failure. This warning was used to control sensor resolution and operating voltage in the readout integrated circuit (ROIC) of an RGB sensor. This strategy improved the accuracy of perception tasks under noisy conditions. Mudassar et al. [51] used the uncertainty associated with the prediction from an object detection DNN to switch modality between RGB and IR. This improved the accuracy of object detection in difficult visible conditions. The idea was also extended to integrate temporal uncertainty for

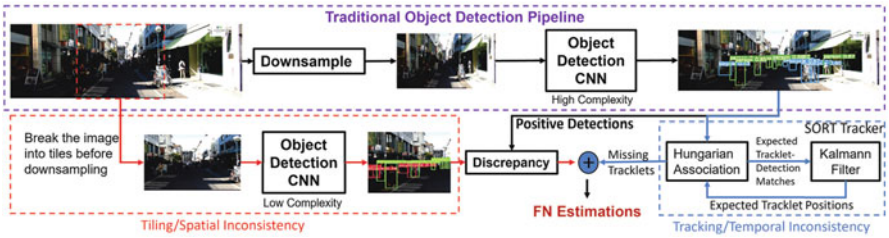


Fig. 8 FN Detection Pipeline [58]. Spatial inconsistency was estimated using tiling and a low complexity detector. Temporal inconsistency was estimated using object tracking from a SORT [57] tracker. Spatial and temporal inconsistencies were concatenated to estimate overall False Negatives

action detection [49]. Table 4 shows the performance of different RGB–IR modality control techniques evaluated on the CAMEL dataset [52].

Samal et al. [58] addressed the perception failure issue directly by creating an false negative (FN) detector, shown in Fig. 8, that used a light-weight secondary object detector and image tiling in addition to temporal inconsistency [56] to detect FNs or perception failures corresponding to the primary object detector. The FNs were considered RoIs that were used to create feedback signal to control sensor parameters such as pixel depth and resolution in an RGB sensor. They created an introspective closed-loop perception system that estimates the perception risk of the RoIs to quantify the possibility of perception failure in such regions and a risk-resource control to drive the sensor control feedback. The closed-loop perception system has less energy consumption than an open-loop system and less marginal cost of prediction than prior active sensor control systems. Figure 9 shows the object detection recall and energy utilization comparison between different closed-loop RGB sensor control systems for different kinds of scenarios.

3.6 Adaptive Computational Load Control

An autonomous CPS system operating in the real world needs to react to the changing dynamics of the scene such as perceptual complexity, power consumption mode, etc. Also, the operating system controlling the system has to perform several tasks outside the autonomy pipeline such as telemetry, networking, memory management, etc. Therefore, it is imperative for the computational load of the autonomy pipeline to be flexible. Since the perception DNN is the most computationally expensive module within the pipeline, the compute graph of the DNN must be dynamic. While there have been several hardware and software techniques to reduce the computational load of the DNNs [62], there are few works that propose dynamically changing the computational load of the DNNs during runtime [63].

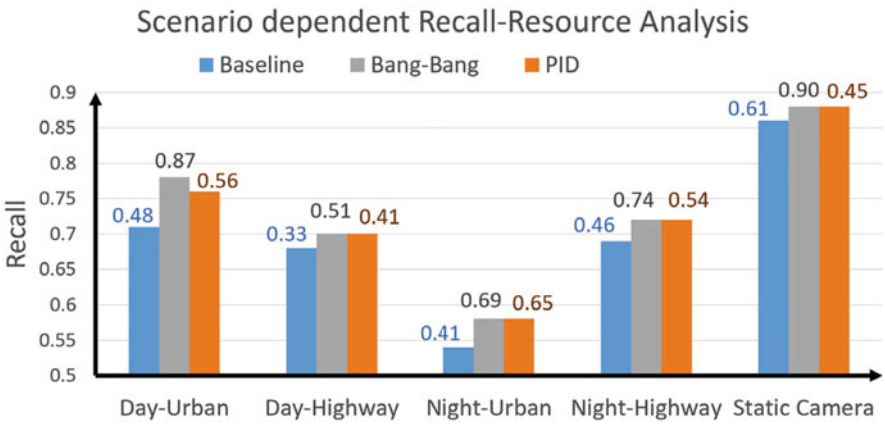


Fig. 9 Recall-resource comparison [58] of closed-loop systems on 5 sequences with different scenarios from the BDD dataset [14]. Here $Baseline$ is from [59], and both $Bang-Bang$ and PID are from [58]. The number on top of a bar represents the corresponding total energy consumption (normalized)

Prior works on dynamic compute control utilize the concept of conditional computation [65] to conditionally deactivate sections of the DNN to reduce the computational load in runtime. Such works include conditionally skipping computation of certain layers [66–68] or channels within the intermediate layers of the network [69–72]. Other works have changed the network architecture to enable early exit [73, 74], i.e., dynamically changing the depth of the network and creating multiple branches within the network each with different computational complexities, and the branch that is executed is determined in runtime [75–77]. There have also been works that use attention to invest the more computational resource to process high-saliency regions compared to the rest [78–81]. But in all the above works, the computational load is conditional on the input rather than an external control signal.

In order to address this issue, Samal et al. [64] proposed a closed-loop perception framework to control the computational load of a DNN-based object detector in an autonomous system according to the end task such as motion planning. The distance-based activation pruning [82], shown in Fig. 10, removes the pixels of the image that are located farther than a particular depth-of-interest (DoI). This operation increases the sparsity within the network activations leading to reduction of computational load in a sparsity-aware DNN accelerator. Similarly, the direction-based activation suppression [64], shown in Fig. 10, uses the direction of motion planning to "suppress" the activations corresponding to pixels outside the region-of-interest (RoI). Table 6 shows the comparison between the both activation pruning and activation suppression w.r.t the open-loop system without compute control. While in both works the DNN computations can be modulated according to an

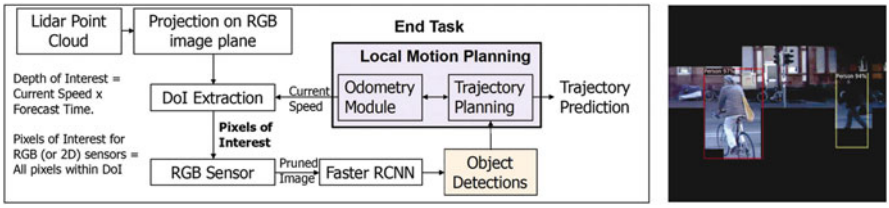


Fig. 10 Distance-driven compute control. Pixels that are located farther away from the sensor are removed

Table 6 Performance comparison of different computation control strategies from [64]. Time steps, avg. speed, and avg. obs. represent the motion planning results from a simulation and energy represents the avg. per frame energy consumption

Name	Time steps	Avg. Speed (m/s)	Avg. Obs.	Energy (J)
Open Loop	42	2.07	5.12	0.17
Activation Pruning	43	1.97	2.76	0.12
Activation Suppression	44	1.96	4.06	0.12

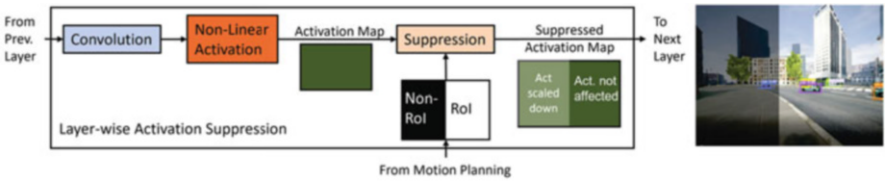


Fig. 11 Direction-driven compute control. Neuron activations in early layers of the DNN corresponding to regions not of interest are "suppressed." Dominant objects outside of RoI are still detected

external signal (DoI or RoI), they still do not take into account the dynamic system requirements such as changes in power mode, operating frequency, task scheduling, reaction time, etc (Fig. 11).

4 Odometery, Localization, and Mapping

In order to operate in the real world, an autonomous cyber-physical system needs to build a map of the environment, localize its position in the map, and finally estimate its motion at every instant. In some systems, the map is built prior to operation; these maps are called HD maps [83]. But the HD maps may not be available for all locations; therefore, many systems build the map online. Since the online mapping depends on the location and pose estimation of the system, simultaneous localization and mapping (SLAM) algorithms have been developed to estimate localization and create 3D map of the world simultaneously in real time. Furthermore, accurate localization requires an estimation of the odometry as well. Traditional algorithms

used sensors such as GPS and IMU for odometry and localization and LiDAR for mapping. But GPS signals have low precision, and IMU sensors suffer from sensor drift that can result in erroneous localization.

4.1 LiDAR Odometry and Mapping

The LiDAR point clouds scanned at any time step are used for registration either by calculating correspondence with previous scans or with an intermittent map that is created online [84]. By minimizing the error during this registration, using algorithms such as iterative closest point (ICP), an estimation of the motion can be derived. Also this registration process can also be used to eliminate erroneous points from the map. But the ICP algorithm is computationally expensive, which increases with the number of points in the point cloud. Therefore, Suma++ uses a DNN-based semantic segmentation network to detect and remove points corresponding to moving objects. This process reduces the complexity of the ICP.

Many cyber-physical systems, such as low-power drones, do not have LiDAR sensors. Therefore, vision-based odometry and SLAM techniques have been developed that can estimate motion, position, and map directly from RGB images. Such techniques are called visual odometry (VO) and visual simultaneous localization and mapping (VSLAM). While traditional feature-based algorithms such as ORB-SLAM [85] have shown decent performance, they do not generalize well across diverse scenarios and require manual parameter optimization that is not ideal for an autonomous system operating in the wild. In recent times, deep-learning-based techniques have been proposed to replace the traditional feature-based VO and VSLAM techniques. Deepvo [86] uses a convolutional neural network (CNN) followed by a recurrent neural network (RNN) to estimate the pose of the system from a sequence of images. The CNN takes two images captured at consecutive time frames as input. This CNN architecture is similar to Flownet [87, 88] that is used to estimate optical flow from monocular images. The long short-term memory (LSTM)-based RNN is used to model the temporal dynamics from sequential data. The network is trained in a supervised setting where the prediction of the network is interpreted as a probabilistic inference and the loss is created from the mean square error (MSE) between the predicted pose and ground truth at every time step. The following equation describes the training process:

$$\theta_{opt} = \underset{\theta}{\operatorname{argmin}}\, MSE(p(\hat{Y}_t | X_{1:t}, \theta), Y_t). \tag{1}$$

Here, θ_{opt} represent the optimal network parameters; Y_t and X_t represent the output pose of the network and a pair of monocular images at time t. Similarly, other methods have been developed that treat pose estimation as a supervised regression problem [89, 90]. Some of these methods also add reprojection loss to augment the

Table 7 Performance comparison of different VSLAM, VO techniques evaluated on sequence 10 from the KITTI Odometry dataset from [93]

Name	ATE	$RPE_{trans}(\%)$	$RPE_{rot}(deg/m)$
ORB-SLAM [85]	19.94	8.65	3.62
LIFT-SLAM [93]	29.87	9.72	2.24
Deepvo [86]	–	8.11	8.83

ATE represents the absolute trajectory error; $RPE_{trans}(\%)$ and $RPE_{rot}(deg/m)$ represent the relative pose error for translation and rotation, respectively. Refer to [92] for details

loss function [91]. Reprojection warps the current image according to current pose estimation to retrieve the previous image.

While DNN-based VSLAM has shown better performance in many autonomous driving datasets, it does not generalize well across datasets. Therefore, hybrid VSLAM techniques merge DNN-based and traditional model-based algorithms. LIFT-SLAM [93] uses a DNN-based orientation estimator similar to LIFT [94] to estimate orientation from image patches. The estimated orientation is fine-tuned according to orientation estimations from a model-based VSLAM algorithm similar to ORB-SLAM [85]. The model-based algorithm operates on the key points and descriptors extracted by the LIFT algorithm to estimate drift and correct the original estimations. Table 7 shows the performance of different VSLAM and VO techniques.

4.2 Unsupervised VSLAM and VO

Mapping, localization, and odometry can also be interpreted as an unsupervised or self-supervised problem [95–97]. Such techniques use DNNs to estimate pose, motion, and/or depth from RGB images and then transform the RGB image according to the estimation and compare the transformed image with actual image to estimate accuracy of the estimation. Figure 12 shows a high-level overview of such techniques. Table 8 shows the performance of unsupervised SLAM techniques. While some techniques simultaneously predict both structure of the world and self-motion, others use sensors such as LiDAR, IMU, etc., to estimate one or the other. In either setting, the estimation of either structure, motion, or both can be validated using several photometric, geometric, and temporal consistencies. Photometric consistency uses motion and/or depth estimation to warp the RGB pixels to create the previous image and pixel-wise error, structural similarity metric (SSIM), etc., to evaluate the accuracy of the estimations. Alternatively, it can also be used to estimate accuracy of optical flow [88] or depth estimations [98, 99] from stereo images. Similarly, ICP algorithm can be used to calculate the relative transformation (Tf_t) between two consecutive depth maps dm_{t-1} and dm_t estimated from RGB images rgb_{t-1} and rgb_t, respectively. This transformation vector can be used to transform the pixels in the RGB image rgb_{t-1} to create warped/estimated RGB

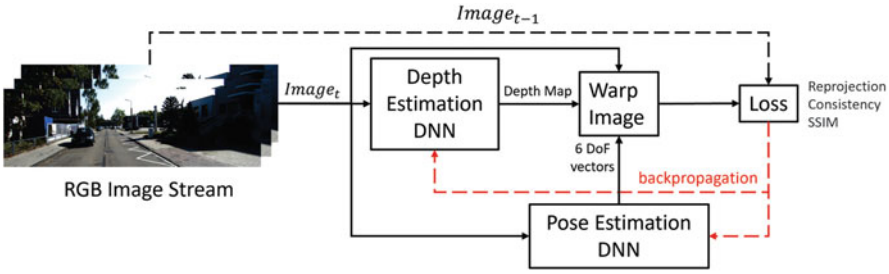


Fig. 12 High-level architecture of VSLAM/VO

Table 8 Performance comparison of different unsupervised SLAM techniques on sequence 10 from the KITTI odometry dataset. Evaluation metric is ATE average per frame

Name	ATE/frame
ORB-SLAM [85]	0.012
Zhou et al. [95]	0.020
Mahjourian et al. [96]	0.012

image $\hat{rgb}_t = T_f(rgb_{t-1})$. The difference between pixel values of $\hat{rgb}_t$ and rgb_t can be considered an indirect estimation of the depth map estimations. Similar consistency losses can be calculated for pose and motion estimations. Finally, the consistency losses are used to train the DNNs responsible for the corresponding estimations using backpropagation.

5 Planning

According to end goal and system configuration, planning in autonomous cyber-physical systems can be divided into many forms such as inverse kinematics, path planning, trajectory planning, etc. Inverse kinematic planning algorithms are used to break down high-level commands such as manipulation of an object into low-level motion commands such as moving the joints in the arm of an industrial robot. Path planning is used in autonomous mobile systems [83], such as AVs, to do long-term planning for, e.g., generating the route from an origin point to an end point, and this route is also known as global waypoints. Trajectory planning is short-term planning that takes the global waypoints, current kinematics, and system constraints into consideration to generate actuation commands such as steering angle, acceleration, etc., in an autonomous mobile system. The most commonly used path planning algorithm such as Djikstra's and A* [100] and trajectory planning algorithms such as Frenet Frame [101] and Spatio-temporal lattice [102] do not require large-scale training. But recently reinforcement learning algorithms have been proposed to do both path and trajectory planning [103]. Performing both path and trajectory planning is also known as motion planning.

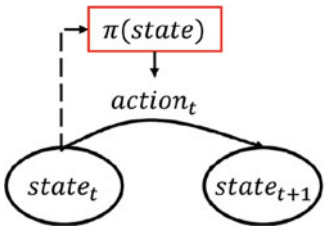


Fig. 13 High-level overview of RL-based motion planning algorithms. The objective of RL algorithms is to learn an optimal action policy ($\pi(state)$)

5.1 Overview of Reinforcement Learning Algorithms

Reinforcement learning algorithms are typically used for motion planning. They learn to predict the ideal motion commands by training on simulation data. Training on real world is not feasible as stochastic actuation of a movable system in the real world for training and exploration can be risky to the environment and also lead to physical damage to the system. Once the RL is learnt in the simulator, it is fine-tuned on real-world scenarios. The reinforcement learning algorithms interpret the motion planning of the system as a Markov decision process (MDP) where the state of the environment is the state of the MDP and the transition between states is performed by taking an action or actuation command [103, 104]. Figure 13 shows the overview of RL. The action at time t is dependent on the state ($state_t$) and is sampled from the action policy π_Q. The objective of RL is to learn an action policy that has the highest return. The return can be divided into short-term reward R and expected long-term value V. Similarly, Q-value is the long-term value V calculated at a state s and action a using the Q-function $Q(s, a)$.

There are many kinds of RL algorithms depending on type of transition function, continuity of action, and state space, etc. Model-based RL algorithms learn the transition probability $T(s_{t+1}|s_t, a_t)$ that represents the probability of entering into state s_{t+1} from s_t by taking action a_t. But since this requires a complete knowledge of process and environment dynamics, such algorithms are not ideal for real-world operations. On the other hand, model-free RL algorithms learn and update their knowledge using trial and error. Similarly, RL algorithms can be classified into on-policy or off-policy algorithms depending on how the Q-function is learnt. On-policy RL algorithms such as state–action–reward–state–action (SARSA) and temporal difference (TD) calculate the Q-value using the current policy only. This leads to a deterministic behavior. On the other hand, off-policy RL algorithms can estimate Q-value with policies that are different from the one that is followed to take actions, and this introduces non-determinism while estimating the action at a state. Therefore, off-policy RL algorithms such as Q-learning are great for exploring the action and state space.

Traditional RL algorithms such as Q-learning and SARSA maintain a table for policy that converts state into action. This table consists of all possible states in one dimension and the expected return of all actions in the other dimension. This table is learned according to the learning policy such as Bellman's equation that maximizes

the return. But such algorithms cannot generalize to unseen states. Therefore, deep Q-networks (DQNs) were developed that use a neural network to estimate the Q-function from training data. It requires two networks namely the target network and the Q-network. The Q-network is used to predict the action in a training episode, and the target network is used to generate the target Q-value and to generate the loss. The loss equation is shown in Eq. (3). This loss is backpropagated through the neural network to adjust the weights of the network using gradient descent.

$$y_t = r_t + \gamma \max_a Q(s_t, a_t, \theta^-)$$
$$Loss = L_f(y_t - Q(s_t, a_t, \theta). \tag{2}$$

Here, y_t is the output of the Q-network, r_t is the short-term reward, and Q is the Q-value from the target network at time t. γ is the discount factor, and L_f is the loss function such as mean square error. The discount factor ($0 \leq \gamma \leq 1$) is used to control the time horizon of an RL algorithm. Low discount factor leads to myopic learning as the algorithm gives high priority to short-term rather than long-term returns. Once the Q-network is trained on a predetermined number of episodes, the weights of the Q-network and the target network are swapped. Such reinforcement learning algorithms that use deep neural networks are called deep reinforcement learning (DRL).

Alternatively, actor–critic RL algorithms, such as A2C and A3C, merge the on-policy and off-policy/value-based learning. Figure 14 shows the framework of such algorithms. The actor network learns a deterministic policy π_θ to convert the state from the environment into action at every time step t, and the critic network evaluates the policy of the actor network by calculating the value $V(s)$ of the action and the time dilation (TD) error δ. The TD error is used to update the actor network.

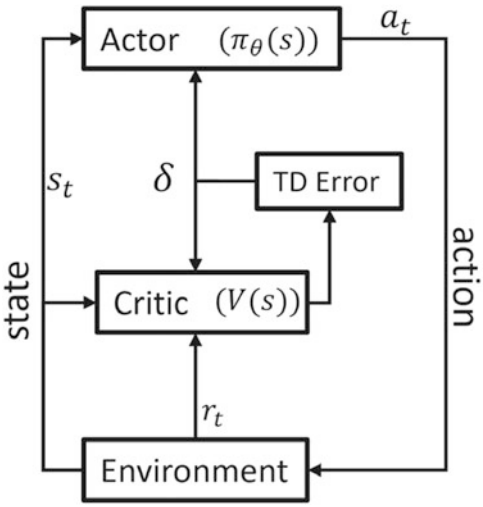


Fig. 14 Actor–critic RL framework with TD error

Simultaneously, the TD error (δ), shown in equation below, is also used by the critic to adjust the value function.

$$\delta = r_{t+1} + \gamma V(s_{t+1}) - V(s_t). \tag{3}$$

Here, r_t and $V(s_t)$ are the reward and value for state s at $t = t$. γ is the discount factor.

Most algorithms presented above require discrete state and action space. But many robotic applications have continuous environmental state and action spaces that if discretized with high precision can lead to exponential increase in memory and computational complexity. Deep deterministic policy gradient (DDPG) [105] on the other hand is ideal for continuous environments with a large continuous action space. They are similar to DQN and actor–critic RL algorithms with certain modifications such as adding stochasticity by introducing noise to the network parameters and/or the predicted action.

5.2 *Applications of Deep Reinforcement Learning Algorithms for Planning*

DRL techniques have been proposed for motion planning in both static and dynamic environments [106]. Lei et al. [107] used a variant of DQN, called double DQN (DDQN) [108], with CNNs for the Q-networks to learn local path planning from LiDAR scans in a simulated environment. Ohnishi et al. [109] used a constrained DQN to learn robot navigation in a simulated environment. A constrained DQN dynamically decides the frequency of swapping of weights between the Q-network and the target network to reduce the number of training samples required. DRL algorithms have also been extended to power systems control. Zhang et al. proposed a DQN to control dynamic voltage and frequency scaling (DVFS) in an edge device. Similarly, Yan et al. [110] used DRL methods to learn load frequency control (LFC) in power systems for renewable energy sources. Such techniques are trained offline and typically extract features in an unsupervised manner using autoencoders.

DRL algorithms have been proposed for active SLAM where an autonomous CPS system creates the map of an unseen environment and navigates it simultaneously. Such algorithms leverage the knowledge of the map of the environment to learn navigation. Botteghi et al. [111] augmented the reward function of a DDPG algorithm to make it map-aware. This led to substantial reduction in collisions and faster convergence. Wen et al. [112] proposed to use Q-learning to learn path planning in a SLAM environment that is created using traditional EKF-SLAM framework from a LiDAR sensor. Later, the work was extended by using a convolutional residual network to predict depth from monocular images and a dueling architecture-based DDQN (D3QN) to plan a path to avoid collision with obstacles [113]. This algorithm was trained in a simulator and eventually transferred

to a robot operating in a controlled real-world scenario. This work was extended to create a two-track robot navigation and obstacle avoidance algorithm [114]. By default, the system navigates the environment using FastSLAM [115], and once an obstacle is detected, D3QN is used to plan a path to avoid the obstacle. A fully convolutional residual network (FCRN) was used to convert RGB images into depth images for mapping and obstacle detection.

Some of the major drawbacks of RL-based planners in general are the lack of determinism and generalization to unseen scenarios, difficulty in tuning the reward function that affects both convergence and optimality and requirement of a large amount of training data. Inverse reinforcement learning (IRL) uses RL to learn from human subjects [116]. Such algorithms address the handcrafting of cost/reward functions by using a neural network to directly map sensor readings to reward map [117]. Recently, these algorithms were extended to DRL paradigm to create deep inverse reinforcement learning (DIRL) algorithms [118]. Typically, such algorithms use the maximum entropy criteria to select the ideal decision from a distribution of sub-optimal decisions collected by observing human subjects. Wulfmeier et al. [119] proposed a DIRL to learn motion planning from a large-scale driving dataset. Rosbach et al. [120] integrated DIRL with a model predictive control (MPC)-based planner to create a behavior-aware motion planner for autonomous driving. Furthermore, DIRL algorithms have also been extended to prediction tasks such as off-road vehicle motion and pedestrian trajectory prediction by incorporating additional task-specific features such as kinematics and social affinity maps, respectively.

While DIRL addresses the handcrafting of reward function issue of RL, it does not address the safety issue that results from a lack of generalizability to unseen scenarios. It should be noted that high-risk scenarios that can lead to an accident are difficult to capture in a dataset, and therefore, algorithms that learn directly from data may not make the optimal decisions in such scenarios. Therefore, Cao et al. [121] proposed a hierarchical RL plus IL algorithm for driving in high risk near accident-driving scenarios. This algorithm uses IL for low-level driving policy decision and RL for high-level driving mode decision. RL algorithms are typically trained in simulation environment, but transferring these learned networks to real world is non-trivial due to shift in data distribution. To address such issues, constrained proximal policy optimization [122] techniques have been proposed that add constraints during the RL formulation and training. Constraints enforced can be kinematic constraints to emulate practical robot mechanics, environmental constraints to emulate the physical constraints of the real world, etc.

6 End-to-End Learning Systems

The meta-architecture of the autonomous cyber-physical systems shown in Fig. 2 is common for most open-source autonomous vehicle and drone control software. This modular structure allows independent development and diagnosis that is critical of

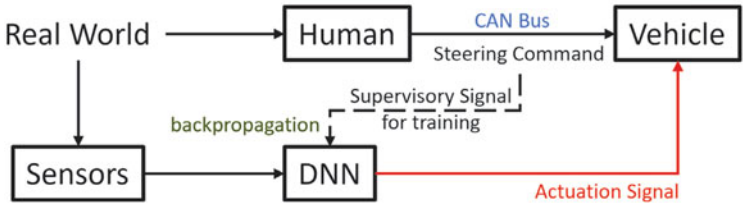


Fig. 15 High-level architecture of an end-to-end learning system for AV. During test time, the actuation signals are generated directly from the DNN

safe operation of systems operating in the wild. But lately with the success of DNNs and availability of powerful embedded platforms, a new category of algorithms have been proposed that replace the various modules in the information processing pipeline with a single DNN [123]. A high-level architecture of an end-to-end self-driving control system is shown in Fig. 15. The primary motivation of this line of thought is the superiority of DNN for low-level feature extraction over hand-crafted feature design for perception tasks. The input of end-to-end learning systems is the raw sensor data, such as images from an RGB camera, and the output is the actuation commands such as steering and acceleration commands. These systems are trained on large volumes of training data from diverse conditions. The supervisory signal for training is provided by human drivers. This is similar to imitation learning and can be formulated as the equation below [124].

$$\underset{\theta}{argmin} \sum_{i} \ell(F(obs_i, \theta), act_i). \tag{4}$$

Here, θ represents the parameters of the network, obs_i and act_i are sensor observations from a scene i and corresponding actions taken by the human supervisor, F represents the neural network, and ℓ represents the loss function used to train the network.

The first work on end-to-end learning system was ALVINN [125] in 1989. It used a fully connected neural network with a single hidden layer to predict the steering command from sensor data. The input layer had three channels or "retina," one for a 8×32 range finder image, second for the blue channel of a 30×32 RGB image, and third for a road intensity feedback unit that represented the relative change in intensity between frames. This light-weight neural network was trained to follow lanes in off-roads and was able to drive a vehicle across a 400 meters in a wooded road at a speed of 0.5 m/s. Subsequently, in 2004, DARPA Autonomous Vehicle (DAVE) [126] project implemented an end-to-end learning system to drive an off-road vehicle through a junk filled alleyway. The input of DAVE was two RGB cameras placed in the right and left sides of the car. The training data for this system was collected by observing the images from RGB cameras and steering commands from CAN bus while manually driving the RC car through several off-road environments. Muller et al. [127] improved upon ALVINN by replacing the

fully connected neural network with a 6-layer convolutional neural network (CNN) that was trained on obstacle avoidance rather than road following and used stereo RGB cameras rather than low-resolution RGB and range finder. This system could drive itself at a maximum speed of 2 m/s. The above works laid the ground work and showed the feasibility of an end-to-end learning system. But these systems were not tested on public roads where they have to avoid many dynamic obstacles while obeying numerous traffic laws.

Recently, inspired by the success of CNNs in solving perception tasks such as image classification, object detection etc., Bojarski et al. [128] proposed a CNN-based end-to-end learning system with 9 layers—1 input normalization layer, 7 convolutional layers, and 2 fully connected layers. The output of the system is the steering angle. The system was trained on roughly 72 h of human driving data collected from cars equipped with three cameras placed on the front left, front center, and front right sides of the car. The system could operate at 30 fps on an NVidia Self-drive PX [129] system. But his system was trained to follow road lanes and was not capable of making complex maneuvers such as lane change, etc. A major limitation of such systems is the lack of controllability by external agents such as a human driver or a behavior planning module. Therefore, Codevilla et al. [124] proposed a conditional end-to-end learning system that is trained with external commands as an additional input, operating as a switch to activate different parts of the network, such that the overall system learns how to respect external commands during inference. But this does not take the planning module into account; therefore, Gao et al. [130] proposed a two-stage end-to-end learning system with path planner at high-level issuing "intention" commands to a DNN-based motion planner at the low level that can respect the intentions. The intentions can be in the form of explicit directional commands such as forward, backward, etc., or they can be in image-like form such as occupancy grid, floor maps, etc. A major drawback of imitation learning is the lack of ability for long-term planning as the learning is carried out on low-level decisions from the human supervisor, and therefore, the system is ignorant of the high-level "motivation" and the uncertainty associated with the decisions. Amini et al. [131] proposed a probabilistic end-to-end learning model that used a Gaussian Mixture Model (GMM) for predicting steering angles with uncertainty. They used noisy GPS signal as a prior for localization uncertainty from RGB images such that the system could issue deterministic steering commands in the presence of a map (Table 9).

The end-to-end learning-based ACPS systems discussed above sample training samples from driving episodes assuming that the original distribution is independent and identically distributed (IID). But real world is causal in nature, and the temporal relationship between the samples cannot be captured using memory-less DNNs that were discussed above. Therefore, recently end-to-end learning systems have been proposed that use DRL instead of CNNs to learn the mapping from sensor data to actuation signals. Sadeghi and Levine proposed CAD^{2}RL [135] that used a fully convolutional network (FCN) and Q-learning to convert RGB images into direction of motion for navigating an autonomous drone while avoiding obstacles. The system was trained on a simulator with randomized ray casting to create stochastic training

Table 9 Performance of different end-to-end learning systems on different autonomous driving datasets from [133]

Dataset	Model	MAE in degree [SD]
Comma.ai [132]	CNN+FCN [128]	2.54 [3.19]
	CNN+LSTM	2.58 [3.44]
	Kim and Canny [133]	2.44 [3.20]
HCE	CNN+FCN [128]	1.27 [1.57]
	CNN+LSTM	1.57 [2.27]
	Kim and Canny [133]	1.20 [1.66]
Udacity [134]	CNN+FCN [128]	4.12 [4.83]
	CNN+LSTM	4.15 [4.93]
	Kim and Canny [133]	4.19 [4.93]

MAE stands for mean absolute error measured in degrees and SD stands for the standard deviation in the measured error

data that helped the learning system to generalize to real-world scenarios. Sallab et al. [136] proposed an end-to-end learning system with 3 stages—CNN-based spatial feature extraction stage, RNN/LSTM-based state estimation stage, and DRL-based policy selection stage. This system was trained and tested for lane keeping in a car racing simulator. Kendall et al. [137] proposed a DDPG-based DRL algorithm to learn lane following in an end-to-end learning-based autonomous vehicle. Contrary to prior works that required an expert/human response for each sample, the DRL system in this work was trained on sparse supervisory signal. The system utilized the distance travelled by the vehicle before safety driver had to intervene as the reward criteria. DNNs are black box in nature due to a lack of insight into the reasoning behind their predictions, and with end-to-end learning, the entire ACPS system becomes similarly obfuscated. Kim and Canny [133] proposed a two-stage method to determine the regions of the scene that have the most impact on the decision of a CNN-based end-to-end learning system. The first stage consists of a visual attention decoder for the CNN-based feature extraction module, and the second stage takes the saliency map generated in the first stage and executes a causality test by removing the pixels at different high salient regions to measure their impact on the predicted decision.

7 Conclusion

Machine learning algorithms, especially deep neural networks, are used extensively in modern cyber-physical systems to support autonomous operation. Due to an abundance of training data and availability of powerful compute hardware, such algorithms have achieved very high task accuracy compared to traditional algorithms, and therefore, DNNs have permeated almost every module of the system stack from low-level sensing to high-level planning. It can also be observed that while CNNs

are better at filtering pertinent information from high-dimensional data in the early processing layers, RL algorithms are better at high-level decision-making. This architecture is not ideal as there is considerable correlation between sensory data as well that cannot be captured by CNNs. Furthermore, the neural-network-based machine learning algorithms today, while very accurate, are essentially black-box entities that cannot explain their predictions. This has led to a schism among the designers of ACPS systems with one section supporting replacement of the modular architecture of the system with a single neural network due to design simplicity and high accuracy. On the other hand, another section within the community is skeptical of using such "black box" algorithms and prefers balkanizing their usage to perception tasks such as object detection only. Since CPS systems are deployed in real-world scenarios, it is paramount that the system should be able to explain itself especially in the event of a failure. And while there are a number of works on attribution, i.e., detecting input regions that are responsible for a particular prediction, there is a lack of work on prediction failures and post-hoc diagnosis. Therefore, a better understanding of a DNN is required for widespread adoption of such algorithms in autonomous cyber-physical systems.

References

1. Kato, S., Tokunaga, S., Maruyama, Y., Maeda, S., Hirabayashi, M., Kitsukawa, Y., Monrroy, A., Ando, T., Fujii, Y., Azumi, T.: Autoware on board: Enabling autonomous vehicles with embedded systems. In: 2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS), pp. 287–296 (2018)
2. Geiger, A., Lenz, P., Stiller, C., Urtasun, R.: Vision meets robotics: The KITTI dataset. Int. J. Robot. Res. **32**, 1231–1237 (2013)
3. Sun, P., Kretzschmar, H., Dotiwalla, X., Chouard, A., Patnaik, V., Tsui, P., Guo, J., Zhou, Y., Chai, Y., Caine, B., Vasudevan V.: Scalability in perception for autonomous driving: Waymo open dataset. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2446–2454 (2020)
4. Caesar, H., Bankiti, V., Lang, A., Vora, S., Liong, V., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., Beijbom, O.: nuScenes: A multimodal dataset for autonomous driving. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 11621–11631 (2020)
5. O'Mahony, N., Campbell, S., Carvalho, A., Harapanahalli, S., Hernandez, G., Krpalkova, L., Riordan, D., Walsh, J.: Deep learning vs. traditional computer vision. In: Science and Information Conference, pp. 128–144 (2019)
6. MIPI White Paper: Driving the Wires of Automotive. https://www.mipi.org/mipi-white-paper-driving-wires-automotive
7. Boulahia, S., Amamra, A., Madi, M., Daikh, S.: Early, intermediate and late fusion strategies for robust deep learning-based multimodal action recognition. Mach. Vis. Appl. **32**, 1–18 (2021)
8. Yeong, D., Velasco-Hernandez, G., Barry, J., Walsh, J.: Sensor and sensor fusion technology in autonomous vehicles: A review. Sensors **21**, 2140 (2021)
9. Fayyad, J., Jaradat, M., Gruyer, D., Najjaran, H.: Deep learning sensor fusion for autonomous vehicle perception and localization: A review. Sensors, **20**, 4220 (2020)
10. Cho, H., Seo, Y., Kumar, B., Rajkumar, R.: A multi-sensor fusion system for moving object detection and tracking in urban driving environments. In: 2014 IEEE International Conference on Robotics and Automation (ICRA), pp. 1836–1843 (2014)

11. Liang, P., Chondro, P., Wu, J., Lai, W., Sun, Y., Lai, Y., Chen, T.: Deep fusion of heterogeneous sensor modalities for the advancements of ADAS to autonomous vehicles. In: 2018 International Symposium on VLSI Design, Automation and Test (VLSI-DAT), pp. 1–4 (2018)
12. Shivakumar, S., Nguyen, T., Miller, I., Chen, S., Kumar, V., Taylor, C.: DFuseNet: Deep fusion of RGB and sparse depth information for image guided dense depth completion. In: 2019 IEEE Intelligent Transportation Systems Conference (ITSC), pp. 13–20 (2019)
13. Kim, J., Choi, J., Kim, Y., Koh, J., Chung, C., Choi, J.: Robust camera LiDAR sensor fusion via deep gated information fusion network. In: 2018 IEEE Intelligent Vehicles Symposium (IV), pp. 1620–1625 (2018)
14. Yu, F., Chen, H., Wang, X., Xian, W., Chen, Y., Liu, F., Madhavan, V., Darrell, T.: BDD100K: A diverse driving dataset for heterogeneous multitask learning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2636–2645 (2020)
15. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards real-time object detection with region proposal networks. Adv. Neural Inf. Proces. Syst. **28**, 1137–1149 (2015). https://ieeexplore.ieee.org/document/7485869
16. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You Only Look Once: Unified, Real-Time Object Detection. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 779–788 (2016)
17. Redmon, J., Farhadi, A.: YOLOV3: An incremental improvement. ArXiv Preprint ArXiv:1804.02767 (2018)
18. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C., Berg, A.: SSD: Single shot multibox detector. In: European Conference on Computer Vision, pp. 21–37 (2016)
19. Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., Zagoruyko, S.: End-to-end object detection with transformers. In: European Conference on Computer Vision, pp. 213–229 (2020)
20. Kim, C., Oghaz, M., Fajtl, J., Argyriou, V., Remagnino, P.: A comparison of embedded deep learning methods for person detection. ArXiv Preprint ArXiv:1812.03451 (2018)
21. Wu, B., Iandola, F., Jin, P., Keutzer, K.: SqueezeDet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 129–137 (2017)
22. Iandola, F., Han, S., Moskewicz, M., Ashraf, K., Dally, W., Keutzer, K.: SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size. ArXiv Preprint ArXiv:1602.07360 (2016)
23. Howard, A., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: MobileNets: Efficient convolutional neural networks for mobile vision applications. ArXiv Preprint ArXiv:1704.04861 (2017)
24. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.: MobileNetV2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4510–4520 (2018)
25. Xiong, Y., Liu, H., Gupta, S., Akin, B., Bender, G., Wang, Y., Kindermans, P., Tan, M., Singh, V., Chen, B.: MobileDets: Searching for object detection architectures for mobile accelerators. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 3825–3834 (2021)
26. Bochkovskiy, A., Wang, C., Liao, H.: YOLOV4: Optimal speed and accuracy of object detection. ArXiv Preprint ArXiv:2004.10934 (2020)
27. Al., G.: Ultralytics/yolov5: v6.0—YOLOv5n 'Nano' models, Roboflow integration, TensorFlow export, OpenCV DNN support. (Zenodo,2021,10). https://doi.org/10.5281/zenodo.5563715
28. Kovács, B., Henriksen, A., Stets, J., Nalpantidis, L.: Object Detection on TPU Accelerated Embedded Devices. Computer Vision Systems. **12899**, 82–92 (2021). https://link.springer.com/chapter/10.1007/978-3-030-87156-7_7#citeas
29. Li, B.: 3D fully convolutional network for vehicle detection in point cloud. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1513–1518 (2017)

30. Gupta, S., Arbelaez, P., Malik, J.: Perceptual organization and recognition of indoor scenes from RGB-D images. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 564–571 (2013)
31. Qi, C., Su, H., Mo, K., Guibas, L.: PointNet: Deep learning on point sets for 3D classification and segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 652–660 (2017)
32. Zhou, Y., Tuzel, O.: VoxelNet: End-to-end learning for point cloud based 3D object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4490–4499 (2018)
33. Lang, A., Vora, S., Caesar, H., Zhou, L., Yang, J., Beijbom, O.: PointPillars: Fast encoders for object detection from point clouds. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 12697–12705 (2019)
34. Schiementz, M.: Postprocessing architecture for an automotive RADAR network. Cuvillier Verlag, Göttingen (2005)
35. Nabati, R., Qi, H.: RRPN: RADAR region proposal network for object detection in autonomous vehicles. In: 2019 IEEE International Conference on Image Processing (ICIP), pp. 3093–3097 (2019)
36. Kumawat, H., Mukhopadhyay, S.: RADAR Guided Dynamic Visual Attention for Resource-Efficient RGB Object Detection. ArXiv Preprint ArXiv:2206.01772 (2022)
37. Yadav, R., Vierling, A., Berns, K.: RADAR + RGB attentive fusion for robust object detection in autonomous vehicles. ArXiv Preprint ArXiv:2008.13642 (2020)
38. Major, B., Fontijne, D., Ansari, A., Teja Sukhavasi, R., Gowaikar, R., Hamilton, M., Lee, S., Grzechnik, S., Subramanian, S.: Vehicle detection with automotive RADAR using deep learning on Range-Azimuth-Doppler tensors. In: Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops, pp. 0–0 (2019)
39. Feng, D., Haase-Schütz, C., Rosenbaum, L., Hertlein, H., Glaeser, C., Timm, F., Wiesbeck, W., Dietmayer, K.: Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges. IEEE Trans. Intell. Transp. Syst. **22**, 1341–1360 (2020)
40. Chen, X., Ma, H., Wan, J., Li, B., Xia, T.: Multi-view 3D object detection network for autonomous driving. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1907–1915 (2017)
41. Ku, J., Mozifian, M., Lee, J., Harakeh, A., Waslander, S.: Joint 3D proposal generation and object detection from view aggregation. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1–8 (2018)
42. Qi, C., Liu, W., Wu, C., Su, H., Guibas, L.: Frustum PointNets for 3D object detection from RGB-D data. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 918–927 (2018)
43. Wang, Z., Jia, K.: Frustum ConvNet: Sliding frustums to aggregate local point-wise features for amodal 3D object detection. In: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1742–1749 (2019)
44. Dou, J., Xue, J., Fang, J.: SEG-VoxelNet for 3D vehicle detection from RGB and LiDAR data. In: 2019 International Conference on Robotics and Automation (ICRA), pp. 4362–4368 (2019)
45. Wang, C., Chen, H., Fu, L.: VPFNet: Voxel-Pixel Fusion Network for Multi-class 3D Object Detection. ArXiv Preprint ArXiv:2111.00966 (2021)
46. Guan, D., Cao, Y., Yang, J., Cao, Y., Yang, M.: Fusion of multispectral data through illumination-aware deep neural networks for pedestrian detection. Information Fusion. **50**, 148–157 (2019)
47. Valada, A., Vertens, J., Dhall, A., Burgard, W.: AdapNet: Adaptive semantic segmentation in adverse environmental conditions. In: 2017 IEEE International Conference on Robotics and Automation (ICRA), pp. 4644–4651 (2017)

48. Mees, O., Eitel, A., Burgard, W.: Choosing smartly: Adaptive multimodal fusion for object detection in changing environments. In: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 151–156 (2016)
49. Mudassar, B., Saha, P., Wolf, M., Mukhopadhyay, S.: A Task-Driven Feedback Imager with Uncertainty Driven Hybrid Control. Sensors **21**, 2610 (2021)
50. Saha, P., Mudassar, B., Mukhopadhyay, S.: Adaptive control of camera modality with deep neural network-based feedback for efficient object tracking. In: 2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), pp. 1–6 (2018)
51. Mudassar, B., Saha, P., Mukhopadhyay, S.: Uncertainty characterization in active sensor systems with DNN-based feedback control. In: 2020 IEEE SENSORS, pp. 1–4 (2020)
52. Gebhardt, E., Wolf, M.: Camel dataset for visual and thermal infrared multiple object detection and tracking. In: 2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), pp. 1–6 (2018)
53. Zhao, H., Zhang, Y., Meng, P., Shi, H., Li, L., Lou, T., Zhao, J.: Towards safety-aware computing system design in autonomous vehicles. ArXiv Preprint ArXiv:1905.08453 (2019)
54. The KITTI Vision Benchmark Suite. http://www.cvlibs.net/datasets/kitti/eval_tracking.php. Cited 17 July 2023
55. Zhang, W., Zhou, H., Sun, S., Wang, Z., Shi, J., Loy, C.: Robust multi-modality multi-object tracking. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2365–2374 (2019)
56. Samal, K., Kumawat, H., Saha, P., Wolf, M., Mukhopadhyay, S.: Task-driven RGB-LiDAR fusion for object tracking in resource-efficient autonomous system. IEEE Transactions on Intelligent Vehicles **7**(1), 102–112 (2021)
57. Bewley, A., Ge, Z., Ott, L., Ramos, F., Upcroft, B.: Simple online and realtime tracking. In: 2016 IEEE International Conference on Image Processing (ICIP), pp. 3464–3468 (2016)
58. Samal, K., Wolf, M., Mukhopadhyay, S.: Introspective Closed-Loop Perception for Energy-efficient Sensors. In: 2021 17th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), pp. 1–8 (2021)
59. Mudassar, B., Saha, P., Long, Y., Amir, M., Gebhardt, E., Na, T., Ko, J., Wolf, M., Mukhopadhyay, S.: CAMEL: An adaptive camera with embedded machine learning-based sensor parameter control. IEEE J. Emerging Sel. Top. Circuits Syst. **9**, 498–508 (2019)
60. Mukherjee, M., Mudassar, B., Lee, M., Mukhopadhyay, S.: Algorithm-circuit cross-layer control for digital pixel image sensors. In: 2020 IEEE SENSORS, pp. 1–4 (2020)
61. Lee, M., Mudassar, B., Mukhopadhyay, S.: Adaptive Camera Platform Using Deep Learning-Based Early Warning of Task Failures. IEEE Sensors J. **21**, 13794–13804 (2021)
62. Liang, T., Glossner, J., Wang, L., Shi, S., Zhang, X.: Pruning and quantization for deep neural network acceleration: A survey. Neurocomputing **461**, 370–403 (2021)
63. Liu, D., Kong, H., Luo, X., Liu, W., Subramaniam, R.: Bringing AI to edge: From deep learning's perspective. Neurocomputing, **485**, 297–320 (2022). https://www.sciencedirect.com/science/article/pii/S0925231221016428
64. Samal, K., Wolf, M., Mukhopadhyay, S.: Closed-loop Approach to Perception in Autonomous System. In: 2021 Design, Automation and Test in Europe Conference and Exhibition (DATE), pp. 463–468 (2021)
65. Bengio, Y., Léonard, N., Courville, A.: Estimating or propagating gradients through stochastic neurons for conditional computation. ArXiv Preprint ArXiv:1308.3432 (2013)
66. Wang, X., Yu, F., Dou, Z., Darrell, T., Gonzalez, J.: SkipNet: Learning dynamic routing in convolutional networks. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 409–424 (2018)
67. Wu, Z., Nagarajan, T., Kumar, A., Rennie, S., Davis, L., Grauman, K., Feris, R.: BlockDrop: Dynamic inference paths in residual networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 8817–8826 (2018)
68. Veit, A., Belongie, S.: Convolutional networks with adaptive inference graphs. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 3–18 (2018)

69. Lin, J., Rao, Y., Lu, J., Zhou, J.: Runtime neural pruning. Adv. Neural Inf. Proces. Syst., **30** (2017)
70. Gao, X., Zhao, Y., Dudziak, Ł., Mullins, R., Xu, C. Dynamic channel pruning: Feature boosting and suppression. ArXiv Preprint ArXiv:1810.05331 (2018)
71. Bejnordi, B., Blankevoort, T., Welling, M.: Batch-shaping for learning conditional channel gated networks. ArXiv Preprint ArXiv:1907.06627 (2019)
72. Yu, J., Yang, L., Xu, N., Yang, J., Huang, T.: Slimmable neural networks. ArXiv Preprint ArXiv:1812.08928 (2018)
73. Bolukbasi, T., Wang, J., Dekel, O., Saligrama, V.: Adaptive neural networks for efficient inference. In: International Conference on Machine Learning, pp. 527–536 (2017)
74. Teerapittayanon, S., McDanel, B., Kung, H.: BranchyNet: Fast inference via early exiting from deep neural networks. In: 2016 23rd International Conference on Pattern Recognition (ICPR), pp. 2464–2469 (2016)
75. Mullapudi, R., Mark, W., Shazeer, N., Fatahalian, K.: HydraNets: Specialized dynamic architectures for efficient inference. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 8080–8089 (2018)
76. Yang, B., Bender, G., Le, Q., Ngiam, J.: CondConv: Conditionally parameterized convolutions for efficient inference. Adv. Neural Inf. Proces. Syst. **32**, 1305–1316 (2019). https://dblp.org/rec/conf/nips/YangBLN19.html?view=bibtex
77. Liu, L., Deng, J.: Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. Proceedings of the AAAI Conference on Artificial Intelligence **32**(1), 3675–3682 (2018). https://dblp.org/rec/conf/aaai/LiuD18.html?view=bibtex
78. Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., Bengio, Y.: Show, attend and tell: Neural image caption generation with visual attention. In: International Conference on Machine Learning, pp. 2048–2057 (2015)
79. Ren, M., Pokrovsky, A., Yang, B., Urtasun, R.: SBNet: Sparse blocks network for fast inference. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 8711–8720 (2018)
80. Figurnov, M., Collins, M., Zhu, Y., Zhang, L., Huang, J., Vetrov, D., Salakhutdinov, R.: Spatially adaptive computation time for residual networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1039–1048 (2017)
81. Hua, W., Zhou, Y., De Sa, C., Zhang, Z., Suh, G.: Channel gating neural networks. Adv. Neural Inf. Proces. Syst. **32**, 1884–1894 (2019). https://researchr.org/publication/HuaZSZS19-0/bibtex
82. Samal, K., Wolf, M., Mukhopadhyay, S.: Attention-based activation pruning to reduce data movement in real-time AI: A case-study on local motion planning in autonomous vehicles. IEEE J. Emerging Sel. Top. Circuits Syst. **10**, 306–319 (2020)
83. Raju, V., Gupta, V., Lomate, S.: Performance of Open Autonomous Vehicle Platforms: Autoware and Apollo. In: 2019 IEEE 5th International Conference for Convergence in Technology (I2CT), pp. 1–5 (2019)
84. Mendes, E., Koch, P., Lacroix, S.: ICP-based pose-graph SLAM In: 2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), pp. 195–200 (2016)
85. Mur-Artal, R., Montiel, J., Tardós, J.: ORB-SLAM: A Versatile and Accurate Monocular SLAM System. IEEE Trans. Robot. **31**, 1147–1163 (2015)
86. Wang, S., Clark, R., Wen, H., Trigoni, N.: DeepVO: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. In: 2017 IEEE International Conference on Robotics and Automation (ICRA), pp. 2043–2050 (2017)
87. Dosovitskiy, A., Fischer, P., Ilg, E., Hausser, P., Hazirbas, C., Golkov, V., Van Der Smagt, P., Cremers, D., Brox, T.: FlowNet: Learning optical flow with convolutional networks. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2758–2766 (2015)
88. Ilg, E., Mayer, N., Saikia, T., Keuper, M., Dosovitskiy, A., Brox, T.: FlowNet 2.0: Evolution of optical flow estimation with deep networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2462–2470 (2017)

89. Melekhov, I., Ylioinas, J., Kannala, J., Rahtu, E.: Relative camera pose estimation using convolutional neural networks. In: International Conference on Advanced Concepts for Intelligent Vision Systems, pp. 675–687 (2017)
90. Parisotto, E., Singh Chaplot, D., Zhang, J., Salakhutdinov, R.: Global pose estimation with an attention-based recurrent network. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 237–246 (2018)
91. Kendall, A., Cipolla, R.: Geometric loss functions for camera pose regression with deep learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 5974–5983 (2017)
92. Schubert, D., Goll, T., Demmel, N., Usenko, V., Stückler, J., Cremers, D.: The TUM VI benchmark for evaluating visual-inertial odometry. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1680–1687 (2018)
93. Bruno, H., Colombini, E.: LIFT-SLAM: A deep-learning feature-based monocular visual SLAM method. Neurocomputing **455**, 97–110 (2021)
94. Yi, K., Trulls, E., Lepetit, V., Fua, P.: LIFT: Learned invariant feature transform. In: European Conference on Computer Vision, pp. 467–483. Springer, Cham (2016)
95. Zou, Y., Luo, Z., Huang, J.: DF-Net: Unsupervised joint learning of depth and flow using cross-task consistency. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 36–53 (2018)
96. Mahjourian, R., Wicke, M., Angelova, A.: Unsupervised learning of depth and ego-motion from monocular video using 3D geometric constraints. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 5667–5675 (2018)
97. Li, R., Wang, S., Long, Z., Gu, D.: UnDeepVO: Monocular visual odometry through unsupervised deep learning. In: 2018 IEEE International Conference on Robotics and Automation (ICRA), pp. 7286–7291 (2018)
98. Godard, C., Mac Aodha, O., Brostow, G.: Unsupervised monocular depth estimation with left-right consistency. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 270–279 (2017)
99. Godard, C., Mac Aodha, O., Firman, M., Brostow, G.: Digging into self-supervised monocular depth estimation. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 3828–3838 (2019)
100. Karur, K., Sharma, N., Dharmatti, C., Siegel, J.: A Survey of Path Planning Algorithms for Mobile Robots. Vehicles. **3**, 448–468 (2021)
101. Werling, M., Ziegler, J., Kammel, S., Thrun, S.: Optimal trajectory generation for dynamic street scenarios in a Frenet frame. In: 2010 IEEE International Conference on Robotics and Automation, pp. 987–993 (2010)
102. McNaughton, M., Urmson, C., Dolan, J., Lee, J.: Motion planning for autonomous driving with a conformal spatiotemporal lattice. In: 2011 IEEE International Conference on Robotics and Automation, pp. 4889–4895 (2011)
103. Kober, J., Bagnell, J., Peters, J.: Reinforcement learning in robotics: A survey. Int. J. Robot. Res. **32**, 1238–1274 (2013)
104. Sutton, R., Barto, A.: Reinforcement learning: An introduction. MIT Press, New York (2018)
105. Lillicrap, T., Hunt, J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. ArXiv Preprint ArXiv:1509.02971 (2015)
106. Yu, J., Su, Y., Liao, Y.: The path planning of mobile robot by neural networks and hierarchical reinforcement learning. Front. Neurorobot., **14**, 63 (2020)
107. Lei, X., Zhang, Z., Dong, P.: Dynamic path planning of unknown environment based on deep reinforcement learning. J. Robot. **2018**, 1–10 (2018). https://www.researchgate.net/publication/327750234_Dynamic_Path_Planning_of_Unknown_Environment_Based_on_Deep_Reinforcement_Learning
108. Van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double Q-learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 30 (2016)

109. Ohnishi, S., Uchibe, E., Yamaguchi, Y., Nakanishi, K., Yasui, Y., Ishii, S.: Constrained deep Q-learning gradually approaching ordinary Q-learning. Front. Neurorobot. **13**, 103 (2019)
110. Yan, Z., Xu, Y.: Data-driven load frequency control for stochastic power systems: A deep reinforcement learning method with continuous action search. IEEE Trans. Power Syst. **34**, 1653–1656 (2018)
111. Botteghi, N., Sirmacek, B., Mustafa, K., Poel, M., Stramigioli, S.: On reward shaping for mobile robot navigation: A reinforcement learning and SLAM based approach. ArXiv Preprint ArXiv:2002.04109 (2020)
112. Wen, S., Chen, X., Ma, C., Lam, H., Hua, S.: The Q-learning obstacle avoidance algorithm based on EKF-SLAM for NAO autonomous walking under unknown environments. Robot. Auton. Syst. **72**, 29–36 (2015)
113. Xie, L., Wang, S., Markham, A., Trigoni, N.: Towards monocular vision based obstacle avoidance through deep reinforcement learning. ArXiv Preprint ArXiv:1706.09829 (2017)
114. Wen, S., Zhao, Y., Yuan, X., Wang, Z., Zhang, D., Manfredi, L.: Path planning for active SLAM based on deep reinforcement learning under unknown environments. Intell. Serv. Robot. **13**, 263–272 (2020)
115. Montemerlo, M., Thrun, S., Koller, D., Wegbreit, B.: FastSLAM: A factored solution to the simultaneous localization and mapping problem. AAAI/IAAI, **593598** (2002). https://dl.acm.org/doi/10.5555/777092.777184
116. Ziebart, B.D., Maas, A.L., Bagnell, J.A., Dey, A.K.: Maximum entropy inverse reinforcement learning. AAAI **8**, 1433–1438 (2008)
117. Wulfmeier, M., Wang, D., Posner, I.: Watch this: Scalable cost-function learning for path planning in urban environments. In: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2089–2095 (2016)
118. Wulfmeier, M., Ondruska, P., Posner, I.: Maximum entropy deep inverse reinforcement learning. ArXiv Preprint ArXiv:1507.04888 (2015)
119. Wulfmeier, M., Rao, D., Wang, D., Ondruska, P., Posner, I.: Large-scale cost function learning for path planning using deep inverse reinforcement learning. Int. J. Robot. Res. **36**, 1073–1087 (2017)
120. Rosbach, S., James, V., Großjohann, S., Homoceanu, S., Roth, S.: Driving with style: Inverse reinforcement learning in general-purpose planning for automated driving. In: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 2658–2665 (2019)
121. Cao, Z., Bıyık, E., Wang, W., Raventos, A., Gaidon, A., Rosman, G., Sadigh, D.: Reinforcement learning based control of imitative policies for near-accident driving. ArXiv Preprint ArXiv:2007.00178 (2020)
122. Gangapurwala, S., Mitchell, A., Havoutis, I.: Guided constrained policy optimization for dynamic quadrupedal robot locomotion. IEEE Robotics and Automation Letters **5**, 3642–3649 (2020)
123. Xu, H., Gao, Y., Yu, F., Darrell, T.: End-to-end learning of driving models from large-scale video datasets. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2174–2182 (2017)
124. Codevilla, F., Müller, M., López, A., Koltun, V., Dosovitskiy, A.: End-to-end driving via conditional imitation learning. In: 2018 IEEE International Conference on Robotics and Automation (ICRA), pp. 4693–4700 (2018)
125. Pomerleau, D.: ALVINN: An autonomous land vehicle in a neural network. Adv. Neural Inf. Proces. Syst. **1**, 305–313 (1988). https://dl.acm.org/doi/10.5555/89851.89891
126. Net-Scale Technologies, Inc.: Autonomous off-road vehicle control using end-to-end learning. Final technical report. http://net-scale.com/doc/net-scale-dave-report.pdf (2004)
127. Muller, U., Ben, J., Cosatto, E., Flepp, B., Cun, Y.: Off-road obstacle avoidance through end-to-end learning. Adv. Neural Inf. Proces. Syst. **18**, 739–746 (2005). https://dl.acm.org/doi/10.5555/2976248.2976341
128. Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L., Monfort, M., Muller, U., Zhang, J.: End to end learning for self-driving cars. ArXiv Preprint ArXiv:1604.07316 (2016)

129. NVIDIA DRIVE PX. https://www.nvidia.com/content/nvidiaGDC/zz/en_ZZ/self-driving-cars/drive-px/ (2015)
130. Gao, W., Hsu, D., Lee, W., Shen, S., Subramanian, K.: Intention-Net: Integrating planning and deep learning for goal-directed autonomous navigation. In: *Conference on Robot Learning*, pp. 185–194 (2017)
131. Amini, A., Rosman, G., Karaman, S., Rus, D.: Variational end-to-end navigation and localization. In: 2019 International Conference on Robotics and Automation (ICRA), pp. 8958–8964 (2019)
132. Comma.ai. Public driving dataset. https://github.com/commaai/research. Cited 17 July 2023
133. Kim, J., Canny, J.: Interpretable learning for self-driving cars by visualizing causal attention. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2942–2950 (2017)
134. The Udacity Dataset. https://github.com/udacity/self-driving-car. Cited 17 July 2023
135. Sadeghi, F., Levine, S.: CAD2RL: Real single-image flight without a single real image. ArXiv Preprint ArXiv:1611.04201 (2016)
136. Sallab, A., Abdou, M., Perot, E., Yogamani, S.: Deep reinforcement learning framework for autonomous driving. Electronic Imaging **2017**, 70–76 (2017)
137. Kendall, A., Hawke, J., Janz, D., Mazur, P., Reda, D., Allen, J., Lam, V., Bewley, A., Shah, A.: Learning to drive in a day. In: *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8248–8254 (2019)

Machine Learning for Efficient Perception in Automotive Cyber-Physical Systems

Joydeep Dey and Sudeep Pasricha

1 Introduction

In 2021, it was reported that an estimated 31,730 people died in motor vehicle traffic crashed in the United States, representing an estimated increase of about 12% compared with 2020 [1]. By eliminating the possibility of human driving errors through automation, advanced driver assistance systems (ADAS) are becoming a critical component in modern vehicles, to help save lives, improve fuel efficiency, and enhance driving comfort. ADAS systems typically involve a four-stage pipeline involving sequential execution of functions related to *perception*, decision, control, and actuation. An incorrect understanding of the environment by the perception system can make the entire system prone to erroneous decision making, which can result in accidents due to imprecise real-time control and actuation. This motivates the need for a reliable *perception architecture* that can mitigate errors at the source of the pipeline and improve safety in emerging semiautonomous vehicles.

The standard SAE-J3016 effectively classifies the capabilities of a perception architecture supported by a vehicle according to their targeted level of autonomy. In general, an optimal vehicle perception architecture should consist of carefully defined location and orientation of each sensor selected from a heterogeneous suite of sensors (e.g., cameras and radars) to maximize environmental coverage in the combined field of view obtained from the sensors. In addition to ensuring accurate sensing via appropriate sensor placement, a high object detection rate and low false positive detection rate need to be maintained using efficient deep learning-based object detection and sensor fusion techniques.

J. Dey · S. Pasricha (✉)
Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO, USA
e-mail: Joydeep.Dey@colostate.edu; sudeep@colostate.edu

S. Pasricha, M. Shafique (eds.), *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing*, https://doi.org/10.1007/978-3-031-40677-5_10

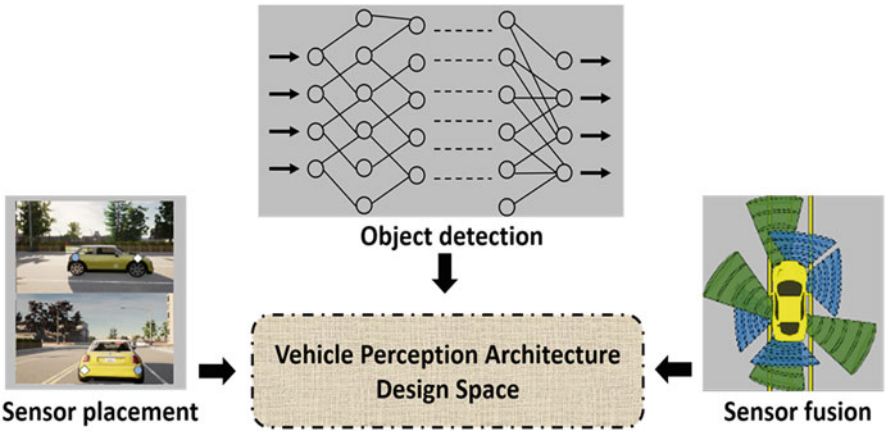


Fig. 1 Breakdown of perception architecture design space

State-of-the-art deep learning-based object detection models are built with different network architectures, uncertainty modeling approaches, and test datasets over a wide range of evaluation metrics [2]. Object detectors that are capable of real-time perception are resource constrained by latency requirements, onboard memory capacity, and computationally complexity. Optimizations performed to meet any one of these constraints often results in a trade-off with the performance of others [3]. As a result, comparison and selection from among the best set of deep learning-based object detectors for perception applications remains a challenge.

In real-world driving scenarios, the position of obstacles and traffic are highly dynamic, so after detection of an object, tracking is necessary to predict its new position. Due to noise from various sources, there is an inherent uncertainty associated with the measured position and velocity. This uncertainty is minimized by using sensor fusion algorithms [4]. An important challenge with sensor fusion algorithms is that the complexity of tracking objects increases as the objects get closer, due to a much lower margin for error (uncertainty) in the vicinity of the vehicle.

As summarized in Fig. 1, the design space of a vehicular perception architecture involves determining appropriate sensor selection and placement, object detection algorithms, and sensor fusion techniques. The possible configurations for each of these decisions are nontrivial and can easily lead to a combinatorial explosion of the design space, making exhaustive exploration impractical. Conversely, an optimization of each of these decisions individually before composing a final solution can lead to solutions that are suboptimal and perform poorly in real environments. Perception architecture design depends heavily on the target features and use cases to be supported in the vehicle, making the already massive design space addressing the problem even larger and harder to traverse. Consequently, today, there are no generalized rules for the synthesis of perception architectures for vehicles.

In this chapter, we describe a novel framework called *P*erception *A*rchitecture *S*earch *T*echnique for *A*DAS (PASTA) that was first presented in [5] to perform perception architecture synthesis for emerging semiautonomous vehicles. Our experimental results indicate that the proposed framework is able to optimize perception performance across multiple ADAS metrics, for different vehicle types.

The main contributions in this chapter include the following:

- A global co-optimization framework capable of synthesizing robust vehicle-specific perception architecture solutions that include heterogeneous sensor placement, deep learning-based object detector design, and sensor fusion algorithm selection
- An exploration of various design space search algorithms tuned for the vehicle perception architecture search problem
- A fast and efficient method for co-exploration of the deep learning object detector hyperparameters, through adaptive and iterative environment- and vehicle-specific transfer learning
- A comparative analysis of the framework efficiency across different vehicle models (Audi-TT and BMW-Minicooper)

2 Related Work

State-of-the-art semiautonomous vehicles require robust perception of their environment, for which the choice of sensor placement, object detection algorithms, and sensor fusion techniques are the most important decisions. These decisions are carefully curated to support ADAS features (e.g., blind spot warning and lane keep assist) that characterize the autonomy level to be supported by a vehicle under design.

Many prior works have explored vehicle perception system design with different combinations of sensor types to overcome limitations that plague individual sensor types. The work in [6] used a single camera-radar pair for perception of headway distance using a continental radar mounted on the geometric center of the front bumper and a Nextbase 512G monocular camera behind the windscreen. Vehicle detection was performed on the collected camera frames, by sorting potential candidates in a fixed trapezoidal region of interest in the horizontal plane. In [6], a camera-radar fusion-based perception architecture was proposed for target acquisition with the well-known single shot detection (SSD) object detector on consecutive camera frames. This allowed their perception system to differentiate vehicles from pedestrians in real time. The detection accuracy was optimized with the use of a Kalman filter and Bayesian estimation, which reduced computational complexity compared with [6]. In [7], a single neural network was used for fusion of all camera and radar detections. The proposed neural fusion model (CRF-Net) used an optimized training strategy similar to the "dropout" technique, where all input neurons for the camera data are simultaneously deactivated in random training steps,

forcing the network to rely more on the radar data. The training focus toward radar overcame the bias introduced by starting with pretrained weights from the feature extractor that was trained from the camera data. The work in [8] optimized merging camera detection with LiDAR processing. An efficient clustering technique inspired by the DBSCAN algorithm allowed for a better exploitation of features from the raw LiDAR point cloud. A fusion scheme was then used to sequentially merge the 2-D detections made by a YOLOv3 object detector using cylindrical projection with the detections made from clustered LiDAR point cloud data. In [9], an approach to fuse LiDAR and stereo camera data was proposed, with a post-processing method for accurate depth estimation based on a patch-wise depth correction approach. In contrast to the cylindrical projection of 2-D detections in [8], the work in [9] uses a projection of 3-D LiDAR points into the camera image frame instead, which upsamples the projection image, creating a more dense depth map.

All of the prior works discussed above optimize vehicle perception performance for rigid combinations of sensors and object detectors, without any design space exploration. Only a few prior works have (partially) explored the design space of sensors and object detectors for vehicle perception. An approach for optimal positioning and calibration of a three LiDAR system was proposed in [10]. The approach used a neural network to learn and qualify the effectiveness of different LiDAR location and orientations. The work in [11] proposed a sensor selection and exploration approach based on factor graphs during multisensor fusion. The work in [12] heuristically explored a subset of backbone networks in the Faster R-CNN object detector for perception systems in vehicles. The work in [13] presented a framework that used a genetic algorithm to optimize sensor orientations and placements in vehicles.

Efficient energy management strategies for ADAS use reliable detections enabled by the optimized perception techniques discussed in [6–13]. The work in [14] derives a prediction mechanism for optimal energy management for ADAS using a nonlinear autoregressive artificial neural network (NARX). Multiple sources are used as input to the neural network such as data from drive cycle information, current vehicle state, global positioning system, travel time data, and detected obstacles. In addition, dynamic programming is used to derive an optimal energy management control strategy, which shows significant fuel economy improvements compared with highly accurate predictive baseline models. The work in [14] proposes a predictive optimal energy management strategy that leverages sensor data aggregation and dynamic programming to achieve vehicle fuel economy improvement for ADAS compared with existing vehicle control strategies. The work discussed in [14, 15] leverage existing ADAS technology in modern vehicles to realize prediction-based optimal energy management, which enables fuel economy improvements for ADAS with minor modifications.

Unlike prior works that fine-tune specific perception architectures, e.g., [6–9], *or explore the sensing and object detector configurations separately, e. g.* [10–13], *this chapter proposes a holistic framework that jointly co-optimizes heterogeneous sensor placement, object detection algorithms, and sensor fusion techniques.* To the best of our knowledge, this is the first effort that performs co-optimization across

such a comprehensive decision space to optimize ADAS perception, with the ability to be tuned and deployed across multiple vehicle types.

3 Background

3.1 ADAS Level 2 Autonomy Features

In this chapter, our exploration of perception architectures on a vehicle, henceforth referred to as an *ego vehicle*, targets four ADAS features that have varying degrees of longitudinal (i.e., in the same lane as the ego vehicle) and lateral (i.e., in neighboring lanes to the ego vehicle lane) sensing requirements. The SAE-J3016 standard [16] defines adaptive cruise control (ACC) and lane keep assist (LKA) individually as level 1 features, as they only perform the dynamic driving task in either the latitudinal or longitudinal direction of the vehicle. Forward collision warning (FCW) and blind spot warning (BW) are defined in SAE-J3016 as level 0 active safety systems, as they only enhance the performance of the driver without performing any portion of the dynamic driving task. However, when all four features are combined, the system can be described as a level 2 autonomy system. Figure 2 shows an overview of the four features we focus on for level 2 autonomy, which are discussed next.

While different ACC implementations may have varying sensing strategies, they all take over longitudinal control from the driver (Fig. 2). The challenge in ACC is to maintain an accurate track of the lead vehicle (immediately ahead of the ego vehicle in the same lane) with a forward facing sensor and using longitudinal control to maintain the specified distance while maintaining driver comfort (e.g., avoiding sudden velocity changes). Lane keep assist (*LKA*) systems determine whether the ego vehicle is drifting toward any lane boundaries and are an evolution of lane departure warning systems. LKA systems have been known to overcompensate, creating a "ping-pong" effect where the vehicle oscillates back and forth between the lane lines [17]. The main challenges in LKA are to reduce this ping-pong effect and the accurate detection of lane lines on obscured (e.g., snow covered) roads. Forward

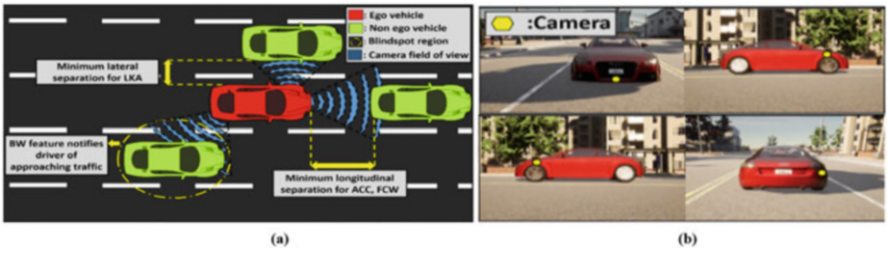


Fig. 2 Visualization of common scenarios in ACC, FCW, LKA, and BW

collision warning (*FCW*) systems are used for real-time prediction of collisions with a lead vehicle.

An important requirement for these systems is that they avoid false positives and false negatives to improve driver comfort and safety and reduce rear-end accidents [18]. Finally, blind spot warning (*BW*) systems use lateral sensor data to determine whether there is a vehicle toward the rear on either side of the ego vehicle (Fig. 2) in a location the driver cannot see with their side mirrors. *A perception architecture designed to support level 2 autonomy in a vehicle should support all four of these critical features.*

3.2 Sensor Placement and Orientation

In order to capture data most relevant to each feature, a strategic sensor placement strategy must be used on the ego vehicle such that their chosen position and orientation maximize coverage (of the vehicle environment). Figure 2 shows an example of field of view coverage (in blue) corresponding to three unique placements of camera sensors on the body of the ego vehicle (in yellow, lower images) to meet coverage goals. For the ACC and FCW features, the ego vehicle is responsible for slowing down to maintain a minimum separation between the ego and lead vehicle. The camera must be positioned somewhere on the front bumper to measure minimum longitudinal separation accurately while keeping the lead vehicle in the desired field of view. For LKA, there is a need to maintain a safe minimum lateral distance between nonego vehicles in neighboring lanes. Here, a front camera is needed to extract lane line information, while side cameras are required for tracking this minimum lateral separation. As BW requires information about a specific area near the rear of the vehicle, it is a challenge to find an optimal sensor placement that maximizes the view of the blind spot. If the sensor is too far forward or too far back, it will miss key portions of the blind spots. Beyond placement, the orientation of sensors can also significantly impact coverage for all features [18]. Thus, sensor placement and orientation remains a challenging problem.

3.3 Object Detection for Vehicle Environment Perception

There are two broad goals associated with deep learning-based object detectors: determining spatial information (relative position of an object in the image) via *localization* followed by identifying which category that object instance belongs to via *classification* [19]. As an example, Fig. 3 shows object detection of multiple car instances (using the YOLOv3 deep learning based object detector [20]) by creating a bounding box around the “car” object instances and predicting the object class as “car.” The pipeline of traditional object detection models can be divided into informative region selection, feature extraction, and classification [21]. Depending

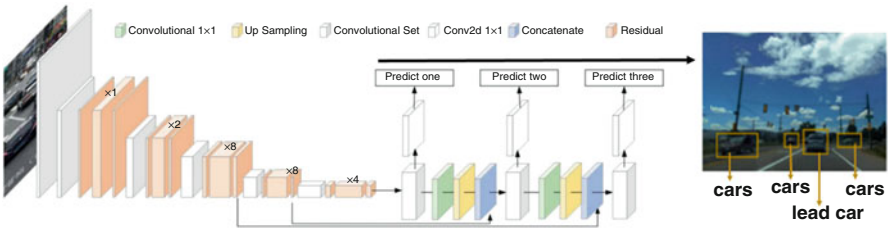


Fig. 3 Example of vehicle (object) detection with YOLOv3

on which subset of these steps is used to process an input image frame, object detectors are classified as single stage or two stage.

Modern *single-stage* detectors are typically composed of a feed-forward fully convolutional network that outputs object classification probabilities and box offsets (w.r.t. predefined anchor/bounding boxes) at each spatial position. The YOLO family of object detectors is a popular example of single-stage detectors [21]. Single shot detection (SSD) is another example, based on the VGG-16 backbone [22]. An advantageous property of single-stage detectors is their very high detection throughput (e.g., ~40 frames per second with YOLO) that makes them suitable for real time scenarios. *Two-stage* detectors divide the detection process into separate region proposal and classification stages. The first stage involves identifying several regions in an image that have a high probability to contain an object using a region proposal network (RPN). In the second stage, proposals of identified regions are fed into convolutional networks for classification. Region-based CNN (R-CNN) is an example of a two-stage detector [23]. R-CNN divides an input image into 2000 regions generated through a selective search algorithm, after which the selected regions are fed to a CNN for feature extraction followed by a support vector machine (SVM) for classification. Fast R-CNN [24] and subsequently faster R-CNN [25] improved the speed of training and detection accuracy compared with R-CNN by streamlining the stages.

Two-stage detectors have high localization and object recognition accuracy, whereas one-stage detectors achieve higher inference speed [26]. *In this chapter, we considered both types of object detectors to exploit the latency/accuracy trade-offs during perception architecture synthesis.*

3.4 Sensor Fusion

Perception architectures that use multiple sensors in their sensing framework often must deal with errors due to imprecise measurements from one or more of the sensors. Conversely, errors can also arise when only a single sensor is used due to measurement uncertainties from insufficient spatial *(occlusion)* or temporal (*delayed sensor response time*) coverage of the environment. The Kalman filter is

one of the most widely used sensor fusion state estimation algorithms that enables error-resilient tracking of targets [27]. The Kalman filter family is a set of recursive mathematical equations that provides an efficient computational solution of the least-squares method for estimation. The filters in this family have the ability to obtain optimal statistical estimations when the system state is described as a linear model and the error can be modeled as Gaussian noise. If the system state is represented as a nonlinear dynamic model as opposed to a linear model, a modified version of the Kalman filter known as the extended Kalman filter (EKF) can be used, which provides an optimal approach for implementing nonlinear recursive filters [28]. However, for real-time ADAS operations, the computation of the Jacobian (matrix describing the system state) in EKF can be computationally expensive and contribute to measurement latency. Further, any attempt to reduce the cost through techniques like linearization makes the performance unstable [29]. The unscented Kalman filter (UKF) is another alternative that has the desirable property of being more amenable to parallel implementation [30]. *In our perception architecture exploration, we explore the family of Kalman filters as candidates for sensor fusion.*

4 PASTA Architecture

4.1 Overview

Figure 4 presents a high-level overview of our proposed *PASTA* framework. The heterogeneous sensors, object detection model library, sensor fusion algorithm library, and physical dimensions of the vehicle model are inputs to the framework. An algorithmic design space exploration is used to generate a perception architec-

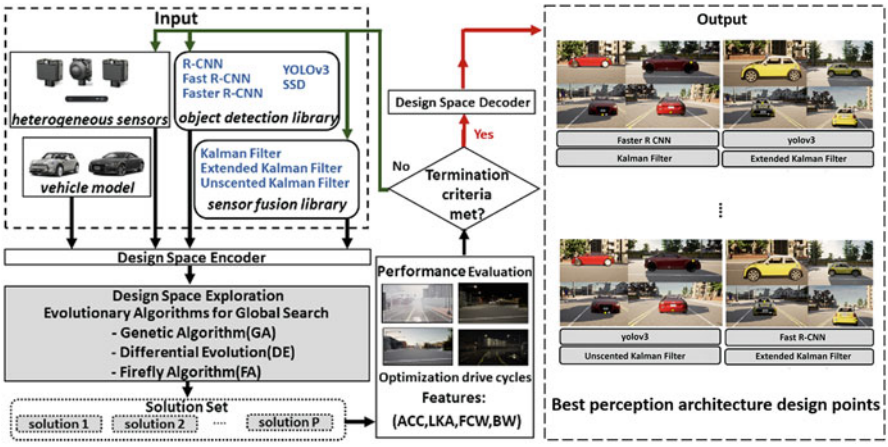


Fig. 4 An overview of the proposed PASTA framework

ture solution, which is subsequently evaluated based on a cumulative score from performance metrics relevant to the ADAS autonomy level being targeted. As part of the framework, we evaluate three design space search exploration algorithms: genetic algorithm (GA), differential evolution (DE), and the firefly algorithm (FA). The process of perception architecture generation and evaluation iterates until an algorithm-specific stopping criteria is met, at which point the best design points are output. The following subsections describe each component of our framework in detail.

4.2 Problem Formulation and Metrics

In our framework, for a given vehicle, we define a *design point* as a perception architecture that is a combination of three components: a *sensor configuration*, which involves the fixed deployment position and orientation of each sensor selected for the vehicle, an *object detector algorithm*, and a *sensor fusion algorithm*. The goal is to find an optimal design point for the given vehicle that minimizes the cumulative error across eight metrics that are characterized of the ability to track and detect nonego vehicles across road geometries and traffic scenarios.

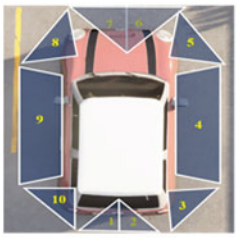
The eight selected metrics are related to our goal of supporting level 2 autonomy with the perception architecture. In the descriptions of the metrics below, the ground truth refers to the actual position of the nonego vehicles (traffic in the environment of the ego vehicle). The metrics can be summarized as follows: *(1) Longitudinal position error and (2) lateral position error,* deviation of the detected positional data from the ground truth of nonego vehicle positions along the y and x axes, respectively. *(3) Object occlusion rate,* the fraction of passing nonego vehicles that go undetected in the vicinity of the ego vehicle. *(4) Velocity uncertainty,* the fraction of times that the velocity of a nonego vehicle is measured incorrectly. *(5) Rate of late detection,* the fraction of the number of "late" nonego vehicle detections made over the total number of nonego vehicles. Late detection is one that occurs after a nonego vehicle crosses the minimum safe longitudinal or lateral distance, as defined by Intel RSS safety models for precrash scenarios [31]. This metric directly factors in the trade-off between latency and accuracy for object detector and fusion algorithms. *(6) False positive lane detection rate,* the fraction of instances when a lane marker is detected but there exists no ground truth lane. *(7) False negative lane detection rate,* the fraction of instances when a ground truth lane exists but is not detected. *(8) False positive object detection rate,* the fraction of total vehicle detections, which were classified as nonego vehicle detections but did not actually exist.

4.3 Design Space Encoder/Decoder

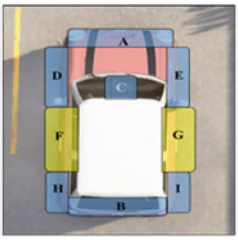
The design space encoder receives a set of random initial design points as input, which are arranged into a vector format. This format is best suited for various kinds of rearrangement and splitting operations during design space exploration. The encoder adapts the initial selection of inputs for our design space such that a design point is defined by the location and orientation of each sensor's configuration (consisting of six parameters: x, y, z, roll, pitch, and yaw), together with the object detector and fusion algorithm. The design space decoder converts the solutions into the same format as the input so that the output perception architecture solution(s) found can be visualized with respect to the real world coordinate system.

4.4 Design Space Exploration

The goal of a design space exploration algorithm in our framework is to generate perception architectures (design points), which are aware of feature to field of view (FOV) zone correlations around an ego vehicle. Figure 5a shows the ten primary FOV zones around the ego vehicle. These zones of interest are defined as the most important perception areas in the environment for a particular ADAS feature. Figure 5b shows the regions on the vehicle on which sensors can be mounted (in blue). Regions F and G (in yellow) are exempt from sensor placement due to the mechanical instability of placing sensors on the door of a vehicle. The correlation between ADAS features, zones, and regions, is shown in Fig. 5c. For exploration of possible locations within a region, a fixed step size of 2 cm in two dimensions across the surface of the vehicle is considered, which generates a 2-D grid of possible positions in each zone shown in Fig. 5b. The orientation exploration of each sensor involves rotation at a fixed step size of 1° between an upper and lower bounding limit for roll, pitch, and yaw, respectively, at each of these possible positions within the 2-D grid. The orientation exploration limits were chosen with caution with the



(a)



(b)

Feature	Region	Zone
BW	B, H, I	1, 2, 3,10
LKA	E, I	3,4,5
	D, H	8, 9, 10
ACC, FCW	A, B,C	6, 7

(c)

Fig. 5 (**a**) Field of view (FOV) zones, (**b**) sensor placement regions, and (**c**) feature, region, and zone relationship

caveat that some sensors, such as long-range radars, have an elevated number of recorded false positives with extreme orientations.

To get a sense of the design space, consider four sensors (e.g., two cameras and two radars). Just the determination of the optimal placement and orientation of these sensors involve exploring ${}^{1.24e+26}C_4$ and ${}^{7.34e+25}C_4$ configurations for the Audi-TT and BMW-Minicooper vehicles, respectively. Coupled with the choice of different object detectors and sensor fusion algorithms, the resulting massive design space cannot be exhaustively traversed in a practical amount of time, necessitating the use of intelligent design space search algorithms that support hill climbing to escape local minima. In our framework, we explored three evolutionary algorithms: (1) *genetic algorithm (GA)*, (2) *differential evolution (DE)*, and the (3) *Firefly algorithm (FA)*. As shown in Fig. 4, each algorithm generates a solution set of size *"P"* at every iteration until the termination criteria is met. The algorithms simultaneously co-optimize sensor configuration, object detection, and sensor fusion and proceed to explore new regions of the design space when the termination (perception) criteria is not met. We briefly describe the three algorithms below.

4.4.1 Genetic Algorithm (GA)

GA is a popular evolutionary algorithm that can solve optimization problems by mimicking the process of natural selection [32]. GA repeatedly selects a population of candidate solutions and then improves the solutions by modifying them. Initially, the GA randomly selects a solution set of fixed size referred to as the population and then improves the quality of the candidate solutions by modifying them in each iteration. GA has the ability to optimize problems where the design space is discontinuous and also if the cost function is nondifferentiable. In our GA implementation, in the selection stage, the cost function values are computed for 50 design points at a time, and a roulette wheel selection method is used to select which set of chromosomes will be involved in the crossover step based on their cost function probability value (fraction of the cumulative cost function sum of all chromosomes considered in the selection). In the crossover stage, the crossover parameter is set to 0.5, allowing half of the 50 chromosomes to produce offspring. The mutation parameter is set to 0.2, which determines the new genes allowed for mutation in each iteration.

4.4.2 Differential Evolution (DE)

Differential Evolution (DE) [33] is another stochastic population-based evolutionary algorithm that takes a unique approach to mutation and recombination. An initial solution population of fixed size is selected randomly, and each solution undergoes mutation and then recombination operations. DE generates new parameter vectors

by adding the weighted difference between two population vectors to a third vector to achieve difference vector-based mutation. Next, crossover is performed, where the mutated vector's parameters are mixed with the parameters of another predetermined vector, the target vector, to yield a trial vector. If the trial vector yields a lower cost function value than the target vector, the trial vector replaces the target vector in the next generation. To ensure that better solutions are selected only after generation of all trial vectors, greedy selection is performed between the target vector and the trial vector at every iteration. Unlike GA where parents are selected based on fitness, every solution in DE takes turns to be one of the parents [34]. In our DE implementation, we set initial population size to 50 and use a crossover probability of 0.8 to select candidates participating in crossover.

4.4.3 Firefly Algorithm (FA)

FA is a swarm-based metaheuristic [35] that has shown superior performance compared with GA for certain problems [36]. In FA, a solution is referred to as a firefly. The algorithm assumes that the attractiveness of a firefly is directly proportional to its luminosity, which depends on the fitness function value. Further, irrespective of gender all fireflies can be attracted to each other in the design space. Initially, a random solution set is generated, and the fitness (brightness) of each candidate solution is measured. In the design space, a firefly is attracted to another with higher brightness (more fit solution), with brightness decreasing exponentially over distance. FA is significantly different from DE and GA, as both exploration of new solutions and exploitation of existing solutions to find better solutions is achieved using a single position update step.

4.5 Performance Evaluation

Each iteration of the design space exploration involves performance evaluation of the generated solution set where each design point undergoes multiple drive cycles. A drive cycle here refers to a virtual simulation involving an ego vehicle (with a perception architecture under evaluation) following a fixed set of waypoint coordinates, while performing object detection and sensor fusion on the environment and other nonego vehicles. A total of 20 different drive cycles were considered, with five drive cycles customized for each ADAS feature. As an example, drive cycles for ACC and FCW involve an ego vehicle following different lead vehicles at different distances, velocities, weather conditions, and traffic profiles. The fitness of the perception architectures generated by the framework are computed using the cumulative metric scores (Sect. 4.2) across the drive cycles.

5 Experiments

5.1 Experimental Setup

To evaluate the efficacy of the PASTA framework, we performed experiments in the open-source simulator Car Learning to Act (CARLA) implemented as a layer on Unreal Engine 4 (UE4) [37]. The UE4 engine provides state-of-the-art physics rendering for highly realistic driving scenarios. We leveraged this tool to design a variety of drive cycles that are roughly 5 minutes long and contain scenarios that commonly arise in real driving environments, including adverse weather conditions (rain and fog) and a few overtly aggressive/conservative driving styles observed with vehicles. To ensure generalizability, we consider a separate set of test drive cycles to evaluate solution quality, which are different from the optimization drive cycles used iteratively by the framework to generate optimized perception architecture solutions.

We target generating perception architectures to meet level 2 autonomy goals for two vehicle models: Audi-TT and BMW-Minicooper (Fig. 6). A maximum of four mid-range radars and four RGB cameras are considered in the design space, where each sensor can be placed in any zone (Fig. 5a and b). Using a greater number of these sensors led to negligible improvements for the level 2 autonomy goal. The RGB cameras possess 90° field of view, 200 fps shutter speed, and image resolution of 800 × 600 pixels. The mid-range radars selected generate a maximum of 1500 measurements per second with a horizontal and vertical field of view of 30° and a maximum detection distance of 100 m. We considered five different object detectors (YOLOv3, SSD, R-CNN, Fast R-CNN, and Faster R-CNN) and three sensor fusion algorithms (Kalman filter, Extended Kalman filter, and Unscented Kalman filter). For the design space exploration algorithms, the cost function was a weighted sum across the eight metrics discussed in Sect. 4.2, with the weight factor for each metric chosen on the basis of their total feature-wise cardinality across all zones shown in Fig. 5c. During design space exploration, if the change in average cost function value was <5% over 250 iterations, the search was terminated. All algorithmic exploration was performed on an AMD Ryzen 7 3800X 8-Core CPU desktop with an NVIDIA GeForce RTX 2080 Ti GPU.

BMW Minicooper

Length	Width	Height	Wheelbase
3.835	1.727	1.414	2.495

(a)

Audi TT

Length	Width	Height	Wheelbase
4.177	1.832	1.353	2.505

(b)

Fig. 6 (**a**) BMW-Minicooper (left) and (**b**) Audi-TT (right)

5.2 Experimental Results

In the first experiment, we explored the inference latency and accuracy in terms of mean average precision (mAP) for the five different object detectors considered in this chapter. Table 1 summarizes the inference latency on a CPU and GPU, as well as the accuracy in mAP for the object detectors on images from our analyzed drive cycles, with all detectors trained on the MS-COCO dataset. It can be observed that the two-stage detectors (R-CNN, Fast R-CNN, and Faster R-CNN) have a higher accuracy than the single stage detectors (SSD and YOLOv3). However, the inference time for the two-stage detector is significantly higher than for the single stage detectors. For real-time object detection in vehicles, it is crucial to be able to detect objects with low latency, typically less than 100 ms [38]. As a result, single-stage detectors are preferable, with YOLOv3 achieving slightly better accuracy and lower inference time than SSD. However, in some scenarios, delayed detection can still be better than not detecting or wrongly detecting an object (e.g., slightly late blind spot warning is still better than receiving no warning) in which case the slower but more accurate two-stage detectors may still be preferable. Our PASTA framework is aware of this inherent trade-off and factors in the detection accuracy and rate of late detection in performance evaluation metrics (Sect. 4.2) to explore both single-stage and two-stage detectors. Also, detectors with a higher mAP value sometimes did not detect objects that other detectors with a lower mAP were able to; thus, we consider all five detectors in our exploration.

Next, we explored the importance of global co-optimization for our problem. We select the genetic algorithm (GA) variant of our framework to explore the entire design space (GA-PASTA) and compared it against five other frameworks. Frameworks GA-PO and GA-OP use the GA but perform a local (sequential) search for sensor design. In GA-PO, sensor position is explored before orientation, while in GA-OP the orientation for fixed sensor locations (based on industry best practices) is explored before adjusting sensor positions. For both frameworks, the object detector used was fixed to YOLOv3 due to its sub-100 ms inference latency and reasonable accuracy, while the extended Kalman filter (EKF) was used for sensor fusion due to its ability to efficiently track targets following linear or nonlinear trajectories. The framework GA-VESPA is from prior work [13] and uses GA for exploration across sensor positions and orientations simultaneously, with the YOLOv3 object detector and EKF fusion algorithm. Frameworks GA-POD and GA-POF use GA for a more comprehensive exploration of the design space. GA-POD simultaneously explores the sensor positioning, orientation, and object detectors, with a fixed EKF fusion

Table 1 Object detector latency and accuracy comparison

Object detector	R-CNN	Fast R-CNN	Faster R-CNN	SSD	YOLOv3
Latency GPU (ms)	48956.18	1834.71	176.99	53.25	24.03
Latency CPU (ms)	66090.83	2365.86	286.72	70.32	32.92
mAP (%)	73.86	76.81	79.63	70.58	71.86

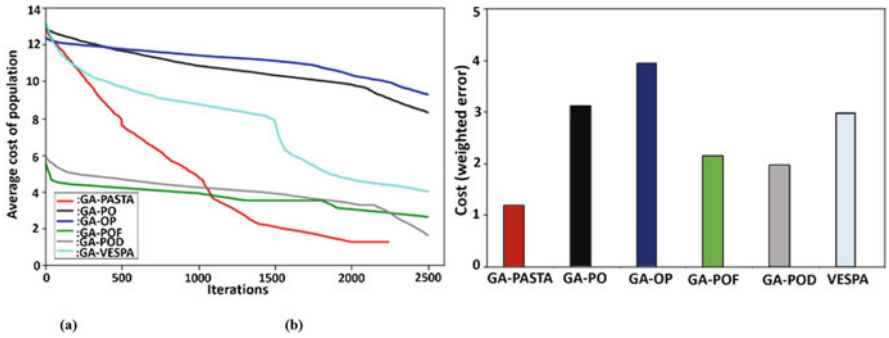


Fig. 7 (**a**) Comparison of perception architecture exploration frameworks. (**b**) Cost of best solution from each framework

algorithm. GA-POF simultaneously explores the sensor positioning, orientation, and sensor fusion algorithm, with a fixed YOLOv3 fusion algorithm.

Figure 7a depicts the average cost of solution populations (lower is better) for the BMW-Minicooper across the different frameworks plotted against the number of iterations, with each exploration lasting between 80 and 100 hours. It can be observed that GA-PO performs better than GA-OP, which confirms the intuitive importance of exploring sensor positioning before adjusting sensor orientations. GA-VESPA outperforms both GA-PO and GA-OP, highlighting the benefit of co-exploration of sensor position and orientation over a local sequential search approach used in GA-PO and GA-OP. GA-POD and GA-POF in turn outperform these frameworks, indicating that decisions related to object detection and sensor fusion can have a notable impact on perception quality. GA-POD terminates with its solution set having a lower average cost than GA-POF, which indicates that co-exploration of object detection and sensor placement/orientation is slightly more effective than co-exploration of sensor fusion and sensor placement/orientation. Our proposed GA-PASTA framework achieves the lowest average cost solution, highlighting the tremendous benefit that can be achieved from co-exploring sensor position/orientation, object detection, and sensor fusion algorithms. Figure 7b summarizes the objective function cost of the best solution found by each framework, which aligns with the population-level observations from Fig. 7a.

The comparative analysis for the BMW-Minicooper was repeated three times with different initializations for all six frameworks, and the results for the other two runs show a consistent trend with the one shown in Fig. 7. Note also that the relative trend across frameworks observed for the Audi-TT is similar to that observed for the BMW-Minicooper, and thus, the results for the Audi-TT are omitted for brevity.

In the next experiment, we explored the efficacy of different design space exploration algorithms (GA, DE, and FA; see Sect. 4.4) to determine which algorithm can provide optimal perception architecture solutions across varying vehicle models. Figure 8 shows the results for the three variants of the PASTA framework, for the Audi-TT and BMW-Minicooper vehicles. The best solution

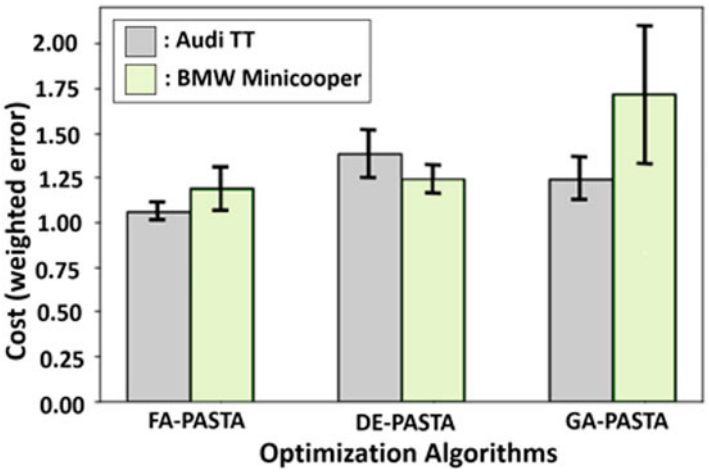


Fig. 8 Comparison of three variants of PASTA framework with genetic algorithm (GA), differential evolution (DE), and Firefly algorithm (FA)

was selected across three runs of each algorithmic variant (variations for the best solution across runs are highlighted with confidence intervals, with bars indicating the median). It is observed that for both considered vehicle models, the FA algorithm outperforms the DE and GA algorithms. For Audi-TT, the best solution found by FA improves upon the best solution found with DE and GA by 18.34% and 14.84%, respectively. For the BMW-Minicooper, the best solution found by FA outperforms the best solution found by DE and GA by 3.16% and 13.08%, respectively. Figure 9a depicts the specific sensor placement locations for each vehicle type, with a visualization of sensor coverage for the best solutions found by each algorithm shown in Fig. 9b.

Finally, in our quest to further improve perception architecture synthesis in PASTA, we focused on a more nuanced exploration of the object detector design space. We selected the FA search algorithm due to its superior performance over GA and DE and modified FA-PASTA to integrate a neural architecture search (NAS) for the YOLOv3 object detector, with the aim of further improving YOLOv3 accuracy across drive cycles while maintaining its low detection latency. Our NAS for YOLOv3 involved transfer learning to retrain network layers with a dataset consisting of 6000 images obtained from the KITTI dataset, using the open source tool CADET [39]. The NAS hyperparameters that were explored involved the number of layers to unfreeze and retrain (from a total of 53 layers in the Darknet-53 backbone used in YOLOv3; Fig. 10a), along with the optimizer learning rate, momentum, and decay. The updated variant of our framework, FA-NAS-PASTA, considered these YOLOv3 hyperparameters along with the sensor positions and orientations, and sensor fusion algorithms, during iterative evolution of the population of candidate solutions in the FA algorithm.

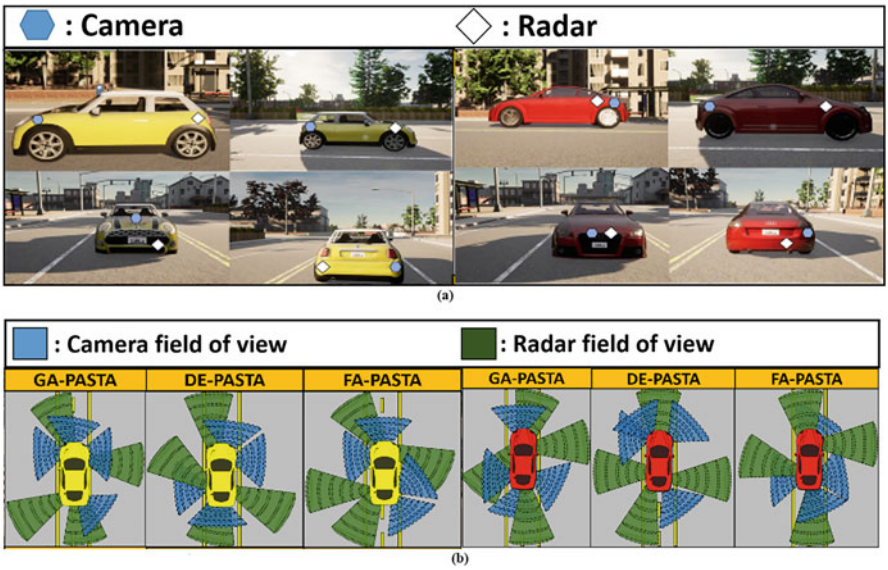


Fig. 9 (**a**) Sensor placement for best solution found with FA algorithm (top yellow vehicle: BMW-Minicooper, bottom red vehicle: Audi-TT) (top) and (**b**) Sensor coverage for best solutions found by GA, DE, and FA search algorithms (bottom)

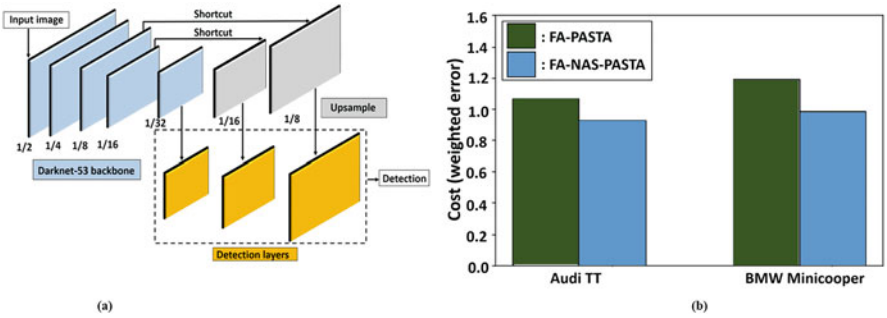


Fig. 10 (**a**) YOLOv3 object detector architecture with Darknet-53 backbone network that was fine-tuned using neural architecture search (NAS) and (**b**) results of integrating object detector NAS with PASTA

Figure 10b shows the results of this analysis for the two vehicles considered. FA-PASTA is the best performing variant of our framework (from Fig. 8), while FA-NAS-PASTA is the modified variant that integrates NAS for YOLOv3. It can be observed that fine tuning the YOLOv3 object detector during search space exploration in FA-NAS-PASTA leads to notable improvements in the best perception architecture solution, with up to 14.43% and 21.13% improvement in performance for the Audi-TT and BMW-Minicooper, compared with PASTA-FA.

6 Conclusion

In this chapter, we propose an automated framework called *PASTA* that is capable of generating perception architecture designs for modern semiautonomous vehicles. *PASTA* has the ability to simultaneously co-optimize locations and orientations for sensors, optimize object detectors, and select sensor fusion algorithms for a given target vehicle. Our experimental analysis showed how PASTA can synthesize optimized perception architecture solutions for the Audi-TT and BMW-Minicooper vehicles, while outperforming multiple semiglobal exploration techniques. Integrating neural architecture search for the object detector in PASTA shows further promising improvements in solution quality.

Acknowledgments This work was supported by the National Science Foundation (NSF), through grant CNS-2132385.

References

1. NHTSA (National Highway Traffic Safety Administration), National Center for Statistics and Analysis: Data estimates indicate traffic fatalities continued to rise at record pace in first nine months of 2021 (2022)
2. Gupta, A., Anpalagan, A., Guan, L., Khwaja, A.S.: Deep learning for object detection and scene perception in self-driving cars: survey, challenges, and open issues. Array. **10**, 100057 (2021)
3. Feng, D., Harakeh, A., Waslander, S.L., Dietmayer, K.: A review and comparative study on probabilistic object detection in autonomous driving. IEEE Trans. Intell. Transp. Syst. **23**, 9961 (2021)
4. Carazo, J., Rufas, D., Bimepica, E., Carrabina, J.: Resource-constrained machine learning for ADAS: a systematic review. IEEE Access. **8**, 40573–40598 (2020)
5. Dey, J., Pasricha, S.: Robust perception architecture design for automotive cyber-physical systems. In: IEEE Computer Society Annual Symposium on VLSI (ISVLSI). IEEE (2022)
6. Zhexiang, Y., Jie, B., Sihan, C., Libo, H., Xin, B.: Camera-radar data fusion for target detection via Kalman filter and Bayesian estimation. SAE Technical Paper (2018)
7. Nobis, F., Geisslinger, M., Weber, M., Betz, J., Lienkamp, M.: A deep learning-based radar and camera sensor fusion architecture for object detection. In: IEEE Sensor Data Fusion: Trends, Solutions, Applications (SDF). IEEE (2020)
8. Verucchi, M., Bartoli, L., Bagni, F., Gatti, F., Burgio, P., Bertogna, M.: Real-time clustering and LiDAR-camera fusion on embedded platforms for self-driving cars. In: IEEE International Conference on Robotic Computing (IRC). IEEE (2020)
9. Meng, L., Yang, L., Tang, G., Ren, S., Yang, W.: An optimization of deep sensor fusion based on generalized intersection over union. In: International Conference on Algorithms and Architectures for Parallel Processing. Springer (2020)
10. Meadows, W., Hudson, C., Goodin, C., Dabbiru, L., Powell, B., Doude, M., Carruth, D., Islam, M., Ball, J.E., Tang, B.: Multi-LIDAR placement, calibration, co-registration, and processing on a Subaru Forester for off-road autonomous vehicles operations. In: Autonomous Systems: Sensors, Processing and Security for Vehicles and Infrastructure. SPIE (2019)
11. Chen, H., Ling, P., Danping, Z., Kun, L., Yexuan, L., Yu, C.: An optimal selection of sensors in multi-sensor fusion navigation with factor graph. In: Ubiquitous Positioning, Indoor Navigation and Location-Based Services (UPINLBS). IEEE (2018)

12. Ji-Qing, L., Sheng Fang, H., Shao, F., Zhong, Y., Hua, X.: Multi-scale traffic vehicle detection based on faster R–CNN with NAS optimization and feature enrichment. Def. Technol. **17**, 1542–1554 (2021)
13. Dey, J., Taylor, W., Pasricha, S.: VESPA: a framework for optimizing heterogeneous sensor placement and orientation for autonomous vehicles. IEEE Consum. Electron. Mag. **10**, 16–26 (2020)
14. Asher, Z., Tunnell, J., Baker, D.A., Fitzgerald, R.J., Banaei-Kashani, F., Pasricha, S., Bradley, T.H.: Enabling prediction for optimal fuel economy vehicle control. SAE International (2018)
15. Tunnell, J., Asher, Z., Pasricha, S., Bradley, T.H.: Towards Improving Vehicle Fuel Economy with ADAS. SAE International (2018)
16. SAE International Standard J3016: Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles (2018)
17. Kirchner, C.: Lane keeping assist explained. Motor Review [Online]. Available: https://motorreview.com/lane-keeping-assist-explained (2014)
18. Li, H., Zhao, G., Qin, L., Aizeke, H., Zhao, X., Yang, Y.: A survey of safety warnings under connected vehicle environments. IEEE Trans. Intell. Transp. Syst. **22**, 2572–2588 (2020)
19. Zhao, Z.Q., Zheng, P., Xu, S.T., Wu, X.: Object detection with deep learning: A review. IEEE Trans. Neural. Netw. Learn. Syst. **30**, 3212–3232 (2019)
20. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: unified, realtime object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. IEEE (2016)
21. Han, J., Zhang, D., Cheng, G., Liu, N., Xu, D.: Advanced deep-learning techniques for salient and category-specific object detection: a survey. IEEE Signal Process. Mag. **35**, 84 (2018)
22. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C.: SSD: single shot multibox detector. In: European Conference on Computer Vision. Springer (2016)
23. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. IEEE (2014)
24. Girshick, R.: Fast R-CNN. In: Proceedings of the IEEE International Conference on Computer Vision. IEEE (2015)
25. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: towards real-time object detection with region proposal networks. In: Advances in Neural Information Processing systems (NIPS) (2015)
26. Fayyad, J., Jaradat, M.A., Gruyer, D., Najjaran, H.: Deep learning sensor fusion for autonomous vehicle perception and localization: a review. Sensors. **20**, 4220 (2020)
27. Kalman, R.E.: A new approach to linear filtering and prediction problems. In: Transactions of the American Society of Mechanical Engineers (ASME) –Journal of Basic Engineering. IEEE (1960)
28. Simon, J., Uhlmann, J.: New extension of the Kalman filter to nonlinear systems. In: Signal Processing, Sensor Fusion, and Target Recognition. International Society for Optics and Photonics (1997)
29. Yeong, D., Hernandez, G., Barry, J., Walsh, J.: Sensor and sensor fusion technology in autonomous vehicles: a review. Sensors. **21**, 2140 (2021)
30. Wan, A., Merwe, R.: The unscented kalman filter for nonlinear estimation. In: Proceedings of the IEEE Adaptive Systems for Signal Processing, Communications, and Control Symposium. IEEE (2000)
31. NHTSA (National Highway Traffic Safety Administration): Implementing RSS model on NHTSA pre-crash scenarios. Intel (2017)
32. Reeves, C.: Genetic Algorithms: Handbook of Metaheuristics. International Series in Operations Research & Management Science (2003)
33. Storn, R., Price, K.: Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. J. Glob. Optim. **11**, 341 (1997)
34. Kachitvichyanukul, V.: Comparison of three evolutionary algorithms: GA, PSO, and DE. Ind. Eng. Manag. Syst. **11**, 215 (2012)

35. Yang, X.S.: Firefly algorithms for multimodal optimization. In: Stochastic Algorithms: Foundations and Applications. Springer (2009)
36. Zhou, G.D., Yi, T.H., Zhang, H., Li, H.N.: A comparative study of genetic and firefly algorithms for sensor placement in structural health monitoring. Shock. Vib. **2015**, 1–10 (2015)
37. Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V.: CARLA: an open urban driving simulator. In: 1st Annual Conference on Robot Learning Conference on Robot Learning. PMLR (2017)
38. Brekke, Å., Vatsendvik, F., Lindseth, F.: Multimodal 3d object detection from simulated pretraining. In: Symposium of the Norwegian Artificial Intelligence Society. Springer (2019)
39. Lin, S., Zhang, Y., Hsu, C., Skach, M., Haque, M., Tang, L., Mars, J.: The architectural implications of autonomous driving:constraints and acceleration. In: Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems. ACM (2018)

Machine Learning for Anomaly Detection in Automotive Cyber-Physical Systems

Vipin Kumar Kukkala, Sooryaa Vignesh Thiruloga, and Sudeep Pasricha

1 Introduction

Today's vehicles are sophisticated cyber-physical systems (CPS) that consists of multiple interconnected embedded systems known as electronic control units (ECUs). The ECUs control various vehicular functions and communicate with each other using the in-vehicle network. In recent years, the number of ECUs along with the complexity of software running on these ECUs has been increasing rapidly, to enable advanced driver assistance systems (ADAS) features such as adaptive cruise control, collision avoidance, lane keep assist, and blind spot warning. This has resulted in an increase in the complexity of the in-vehicle network over which huge volumes of automotive sensor and real-time decision data, and control directives are communicated. This in turn has led to various challenges related to the reliability [1–4], security [5–9], and real-time control of automotive applications [10–13].

Recent developments in ADAS resulted in increased interaction with various external systems using advanced communication standards such as 5G technology and Vehicle-to-X (V2X) [14]. Unfortunately, this makes automotive embedded systems highly susceptible to various cybersecurity threats that can have catastrophic consequences. The vehicular attacks in [15–17] have presented different ways to gain access to the in-vehicle network and override vehicle controls by injecting anomalous messages. With the connected and autonomous vehicles (CAVs) on the horizon, these security concerns will get further aggravated. Therefore, it is crucial to prevent unauthorized access to in-vehicle networks by external attackers to ensure the security of automotive CPS.

V. K. Kukkala (✉) · S. V. Thiruloga · S. Pasricha
Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO, USA
e-mail: kvipin@rams.colostate.edu; sooryaa@colostate.edu; sudeep@colostate.edu

S. Pasricha, M. Shafique (eds.), *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing*, https://doi.org/10.1007/978-3-031-40677-5_11

Traditional computer networks utilized firewalls to defend the networks from external attackers. However, no firewall is flawless, and no network can be completely secure. Therefore, there is a need for an active monitoring system that continuously monitors the network to identify malicious messages in the system. An anomaly detection system (ADS) can be used to continuously monitor the in-vehicle network traffic and trigger alerts when suspicious messages or known threats are detected, which is typically the last line of defense in automotive CPSs.

At a high level, ADSs are categorized into two types: *(i) rule-based* and (ii) *machine learning-based*. Rule-based ADSs observe for traces of previously observed attack signatures whereas machine learning-based ADSs observe for the deviation from the known normal system behavior to detect the presence of an attacker. Rule-based ADS can have faster detection rates and very few false alarms (false positive rate) but are limited to detecting only known attacks. On the contrary, machine learning-based ADS can detect both previously observed and novel attacks but can suffer from comparatively slower detection times and higher false alarm rate. An efficient ADS needs to be robust, scalable, and incur minimal overhead (lightweight). Moreover, practical ADSs need to have a wide attack coverage (being able to detect both known and unknown attacks) with high confidence in detection and low false alarms as recovering from false alarms can be costly.

Obtaining the signature of every possible attack is impractical and would limit us to only detecting known attacks. Hence, we believe that machine learning-based ADSs provide a more pragmatic solution to this problem. Additionally, due to the ease of acquiring in-vehicle network data, large volumes of in-vehicle message data can be collected, which facilitates the use of advanced deep learning models for detecting anomalies in automotive CPS [9].

In this chapter, we propose a novel ADS framework called *INDRA*, first presented in [6], that monitors the messages in controller area network (CAN)-based automotive CPS for anomalies. During the offline phase, *INDRA* uses a deep learning-based model to learn the normal system behavior in an unsupervised manner. At runtime, *INDRA* continuously scans the network for anomalous messages in the network. *INDRA* aims to maximize the detection accuracy with minimal false alarms and overhead on the ECUs.

Our novel contributions in this work are as follows:

1. We introduced a gated recurrent unit (GRU)-based recurrent autoencoder network to learn the normal system behavior during the offline phase.
2. We presented an anomaly score (AS) metric to measure deviation from the normal system behavior.
3. We conducted a comprehensive analysis toward the selection of thresholds for the anomaly score metric.
4. We compare our proposed *INDRA* framework with the best-known prior works in the area, to show its effectiveness.

2 Related Work

Several techniques have been proposed to design ADS for protecting time-critical automotive CPS. These works try to detect multiple attacks by monitoring the in-vehicle network data.

Rule-based ADS detects known attacks by using the information about previously observed attack signatures. A language theory-based model [18] was proposed to derive attack signatures. However, this technique fails to detect attacks when it misses the packets transmitted during the early stages of an attack. The authors in [19], used transition matrices to detect attacks in a CAN bus. They were able to achieve a low false-positive rate for simple attacks but failed to detect advanced replay attacks. In [20], the authors identify notable attack signatures such as an increase in message frequency and missing messages to detect attacks. In [21], the authors proposed a specification-based approach to detect attacks; they analyze the behavior of the system and compare it with the predefined attack patterns to detect anomalies. However, their system fails to detect unknown attacks. The authors in [22] propose an ADS technique using the Myers algorithm [23] under the map-reduce framework. In [24], a time-frequency analysis of CAN messages is used to detect multiple anomalies. The authors analyzed message frequency at design time in [25] to derive regular operating mode region. This region is observed for deviations at runtime to detect anomalies. The sender ECU's clock skew and the messages are used to detect attacks [26] by observing for variations in the clock-skew at runtime. The authors in [27] performed a formal analysis on clock-skew-based ADS and evaluated on a real vehicle. In [28], a memory heat map is used to characterize the memory behavior of the operating system to detect anomalies. In [29], an entropy-based ADS is proposed, which observes for change in system entropy to detect anomalies. Nonetheless, the technique fails to detect small scale attacks for which the entropy change is minimal. In conclusion, rule-based ADSs offer a solution to the intrusion detection problem with lower false positive rates but cannot detect more complex and novel attacks. Moreover, obtaining signatures of every possible attack pattern is not practical.

Machine learning-based ADSs aim to learn the normal system behavior in an offline phase and observe for any deviation from the learned normal behavior to detect anomalies at runtime. In [30], the authors proposed a sensor-based ADS that utilizes attack detection sensors to monitor various system events to observe for deviations from normal behavior. However, this approach is expensive and suffers from poor detection rates. In [31], a one-class support vector machine (OCSVM)-based ADS was introduced, but it suffers from poor detection latency. An ensemble of different nearest neighbor classifiers was used in [32] to distinguish between a normal and an attack-induced CAN payload. The authors in [33] proposed a decision-tree-based detection model to monitor the physical features of the vehicle to detect attacks. However, this model is not practical and suffers from high anomaly detection latencies. In [34], a hidden Markov model (HMM)-based technique was proposed to monitor the temporal relationships between messages to detect attacks.

Table 1 Performance metrics comparison between our proposed *INDRA* framework and state-of-the-art machine learning-based anomaly detection works

	Performance metrics			
Technique	Lightweight model	Low false positive rate	High detection accuracy	Fast inference time
PLSTM [38]	X	✓	X	X
RepNet [39]	✓	X	X	✓
CANet [36]	X	✓	✓	X
INDRA	✓	✓	✓	✓

A deep neural network-based approach was proposed to scan the payload in the in-vehicle network in [35]. This approach is not scalable as it is fine-tuned for a low priority tire pressure monitoring system (TPMS), which makes it hard to adapt to high priority powertrain applications. In [36] a long-short-term memory (LSTM)-based ADS for multi-message ID detection was proposed. However, the model architecture is highly complex and incurs high overhead on the ECUs. An LSTM-based ADS to detect insertion and dropping attacks (explained in Sect. 4.3) is proposed in [37]. In [38], an LSTM-based predictor model is proposed to predict the subsequent time step message value at a bit level and observe for large variations to detect anomalous messages. A recurrent neural network (RNN)-based ADS to learn the normal CAN message pattern in the in-vehicle network is proposed in [39]. In [40], a hybrid ADS was proposed which utilizes a specification-based system in the first stage and an RNN-based model in the second stage to detect anomalies in time-series data. Several other machine models such as the stacked LSTMs and temporal convolutional neural networks (TCNs)-based techniques were proposed in [7, 8], respectively. However, none of these techniques provides a complete system-level solution that is scalable, reliable, and lightweight to detect various attacks for in-vehicle networks.

In this chapter, we introduce a lightweight recurrent autoencoder-based ADS using gated recurrent units (GRUs) that monitors the in-vehicle network messages at a signal level to detect multiple types of attacks with higher efficiency than various state-of-the art works in this area. A summary of some of the state-of-the-art works' performance under different metrics and our proposed *INDRA* framework is presented in Table 1. An exhaustive analysis of each metric and evaluation results are presented later in Sect. 6.

3 Sequence Learning Background

The availability of increased computing power from GPUs and custom accelerators training deep neural networks with many hidden layers became feasible and has led to the creation of powerful models for solving difficult problems in many domains.

One such problem is detecting anomalies in automotive CPS. In an automotive CPS, the communication between ECUs occurs in a time-dependent manner. Therefore, there is temporal relationship between the messages, which can be exploited in order to detect anomalies. However, this cannot be achieved using typical feedforward neural networks where the output of a specific input at an instance is independent of the other inputs. Sequence models can be an appropriate approach for such problems, as they inherently handle sequences and time-series data.

3.1 Sequence Models

A sequence model is a function that ensures that the outcome is reliant on both the current and prior inputs. The recurrent neural network (RNN), which was introduced in [41], is an example of such a sequence model. Moreover, other sequence models such as gated recurrent unit (GRU) and long-short-term memory (LSTM) have also been developed.

3.1.1 Recurrent Neural Networks (RNNs)

An RNN is a form of artificial neural network that takes the sequential data as input and tries to learn the relationships between the elements in the sequence. The hidden state in RNNs allows learned information from previous time steps to persist over time. An RNN unit with feedback is shown in Fig. 1a, and an unrolled RNN in time is shown in Fig. 1b.

The output h_t of an RNN unit is a function of both the input x_t and the previous output h_{t-1}:

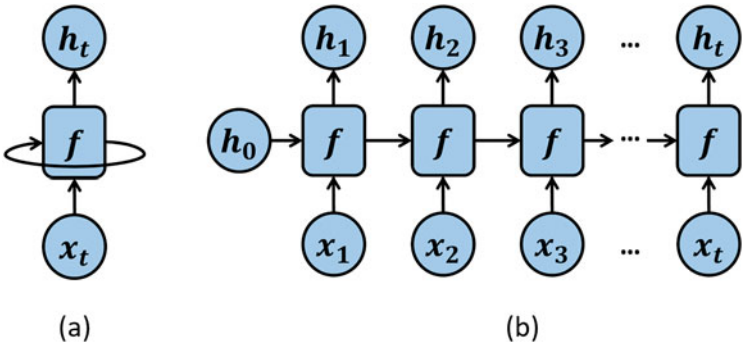


Fig. 1 (**a**) A single RNN unit and (**b**) RNN unit unrolled in time, where f is the RNN unit, x is the input, and h represents hidden states

$$h_t = f\left(W x_t + U h_{t-1} + b\right) \tag{1}$$

where f is a nonlinear activation function (e.g., sigmoid or tanh), U and W are weight matrices, and b is the bias term. One of the major limitations of RNNs is that they are very hard to train. Since RNNs and other sequence models handle sequences or time-series inputs, backpropagation occurs through various time steps (known as backpropagation through time). During this process, the feedback loop in RNNs causes the errors to expand or shrink rapidly, thereby creating exploding or vanishing gradients, respectively, which in turn destroys the information in backpropagation. This vanishing gradient problem prohibits RNNs from learning *long-term dependencies*. To solve this problem, additional states and gates were introduced in the RNN unit in [42] to remember long-term dependencies, which led to the development of LSTM Networks.

3.1.2 Long-/Short-Term Memory (LSTM) Networks

LSTMs unlike RNNs uses cell state and hidden state information along with multiple gates to remember long-term dependencies between messages. The cell state can be imagined as a freeway that carries relevant information throughout the processing of a sequence. The state stores information from previous time steps so that it can be used in subsequent time steps, reducing the effects of short-term memory. The gates modify the information in the cell state. As a result, the gates in LSTM assist the model in determining which information should be retained and which should be ignored.

An LSTM unit contains three gates: (i) input gate (f_t) (ii) forget gate (i_t), and (iii) output gate (o_t) as shown in Fig. 2a. The forget gate is a binary gate that determines which information from the previous cell state (c_{t-1}) to retain. The input gate adds relevant information to the cell state (c_t). Finally, the output gate uses information from the previous two gates to produce an output. An LSTM unit unrolled in time is shown in Fig. 2b.

LSTMs learn long-term dependencies in a sequence by using a combination of different gates and hidden states. However, they are not computationally efficient due to the addition of multiple gates, as the sequence path is more complicated than in RNNs, which in turn requires more memory at runtime. Moreover, training LSTMs have a high computation overhead even when the advanced training methods such as truncated backpropagation are employed. To overcome abovementioned limitations, a simpler recurrent neural network called gated recurrent unit (GRU) network was introduced in [43]. GRUs can be trained faster than LSTMs and also can remember dependencies in long sequences with minimal overhead (in both memory and runtime), while solving the vanishing gradient problem.

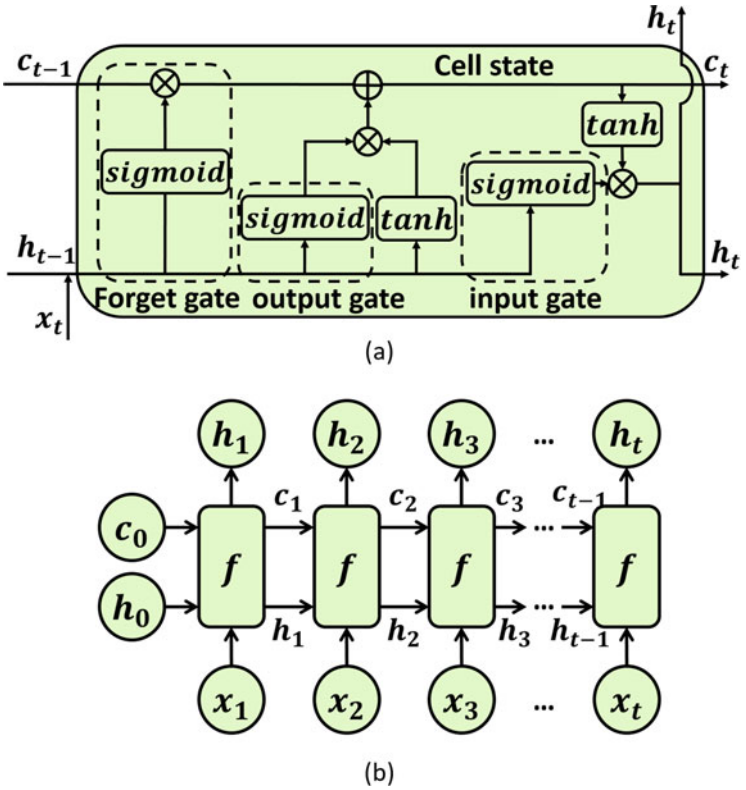


Fig. 2 (**a**) A single LSTM unit with different gates and (**b**) unrolled LSTM unit in time, where *f* is an LSTM unit, *x* is input, *c* is cell state, and *h* is the hidden state

3.1.3 Gated Recurrent Unit (GRU)

Unlike LSTMs, a GRU unit takes a different route for gating information. The input and forget gate of the LSTM is combined into a solitary *update* gate and in addition combines hidden and cell state, as shown in Fig. 3a, b.

A typical GRU unit contains two gates: (i) reset gate and (ii) update gate. The reset gate combines new input with previous memory, while the update layer determines how much relevant data should be stored. Thus, a GRU unit controls the data stream similar to an LSTM by uncovering its hidden layer contents. Moreover, GRUs are computationally more efficient than LSTMs as they achieve this using fewer gates and states, with low memory overhead. It is crucial to use lightweight machine learning models as real-time automotive ECUs are highly resource-constrained embedded systems with strict energy and power budgets. Thus, GRU-based networks are an ideal fit for inference in automotive systems. Hence, *INDRA* chose to use a lightweight GRU-based model to implement an ADS (explained in detail in Sect. 5).

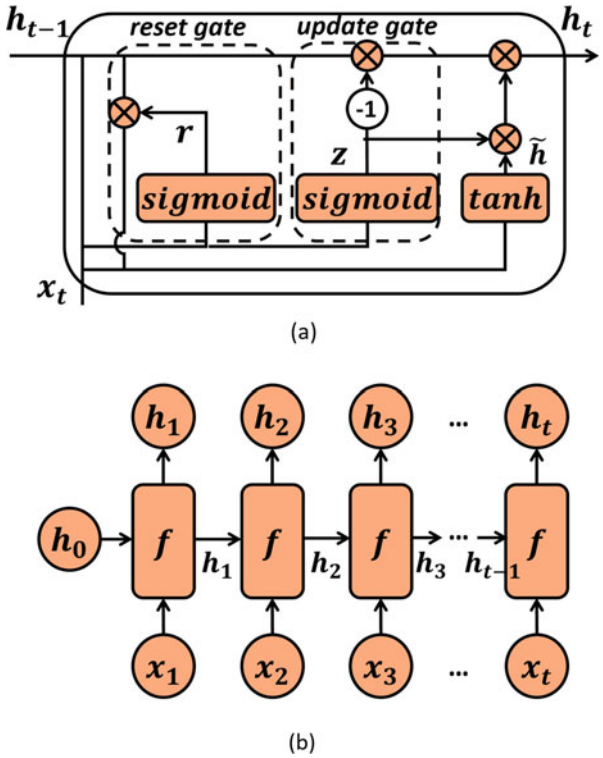


Fig. 3 (**a**) A single GRU unit with different gates and (**b**) GRU unit unrolled in time, where f is a GRU unit, x is input, and h is the hidden state

The major advantage of sequence models is that they can be trained in both supervised and unsupervised learning fashion. Due to the large volume of CAN message data in a vehicle, labeling all that data can become very tedious. Additionally, the variability in the messages between vehicle models from the same manufacturer and the proprietary nature of this information makes it even more challenging to label messages correctly. Nonetheless, due to the ease of obtaining CAN message data via onboard diagnostics (OBD-II), large amounts of unlabeled data can be collected easily. Thus, *INDRA* uses GRUs in an unsupervised learning setting.

3.2 Autoencoders

Autoencoders are unsupervised learning-based artificial neural networks who try to reconstruct the input by learning the latent input features. They accomplish this by encoding the input data (x) to a hidden layer and finally decoding it to produce a reconstruction $\tilde{x}$ (as shown in Fig. 4). This encoded information at the hidden

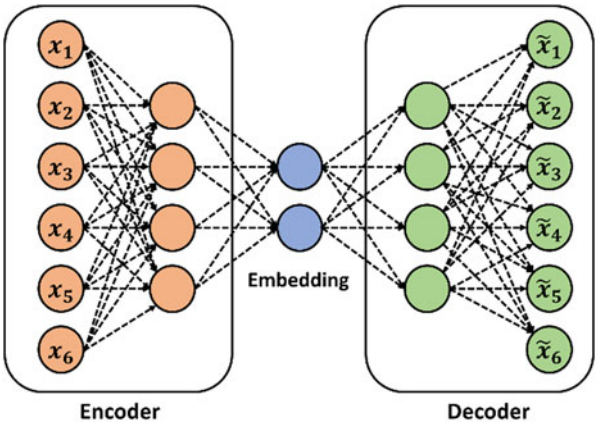


Fig. 4 A simple autoencoder network with encoder, decoder, and embedding layers of the network

layer is called an embedding. The layers that are used to create this embedding are called the encoder, and the layers that are used in reconstructing the embedding into the original input (decoding) are called the decoder. During the training process, the encoder attempts to learn a nonlinear mapping of the inputs, while the decoder tries to learn the nonlinear mapping of the embedding to the inputs. The encoder and decoder accomplish this with the help of nonlinear activation functions such as tanh and rectified linear unit (ReLU). Moreover, the autoencoder aims to recreate the input as closely as possible by extracting important features from the inputs with a goal of minimizing reconstruction loss. The most used loss functions in autoencoders include mean squared error (MSE) and Kullback-Leibler (KL) divergence.

Since autoencoders aim to reconstruct the input by learning the underlying distribution of the input data, they are an excellent choice for efficiently learning and reconstructing highly correlated time-series data by learning the temporal relations between messages. *Hence, our proposed INDRA framework uses lightweight GRUs in an autoencoder to learn latent representations of CAN message data in an unsupervised learning setting*.

4 Problem Definition

4.1 System Model

In this chapter, we consider a generic automotive *system* consisting of multiple ECUs connected using a CAN-based in-vehicle network, as shown in Fig. 5. Each ECU connected in the network is responsible for running a specific set of automotive applications that are hard real time in nature (i.e., have strict timing and deadline

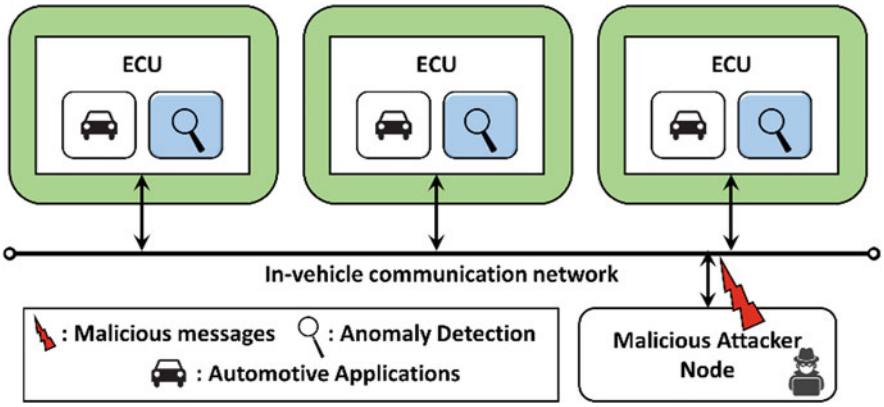


Fig. 5 Overview of the automotive system model considered in *INDRA*

constraints). Moreover, we assume that each ECU also runs anomaly detection applications (ADS), which are responsible for monitoring and detecting anomalies in the in-vehicle network. *INDRA* considers a distributed ADS approach (anomaly detection application is collocated with automotive applications) as opposed to a centralized ADS approach in which one central ECU handles all anomaly detection tasks due to the following reasons:

- A centralized ADS approach is susceptible to single-point failures, which can completely expose the system to the attacker.
- In the worst-case scenarios such as during a flooding attack (explained in Sect. 4.3), the centralized system might not be able to communicate with the victim ECUs due to highly congested in-vehicle network.
- If an attacker successfully tricks the centralized ADS ECU, the attacks can go undetected by the other ECUs, compromising the entire system; however, in a distributed ADS scenario, it requires fooling multiple ECUs (which is more difficult) to compromise the system. Moreover, in a distributed ADS scenario, even if one of the ECU is compromised, the attacks can still be detected by the decentralized intelligence.
- In a distributed ADS, ECUs can stop accepting messages as soon as an anomaly is detected rather than having to wait for a centralized system to notify them, resulting in faster reaction times.
- With a distributed ADS, the computation load of ADS is split among the ECUs and monitoring can be limited to only required messages. As a result, multiple ECUs can independently monitor a subset of messages with lesser overhead.

For the abovementioned reasons, many prior works such as [18, 25] also consider a distributed ADS approach. Furthermore, with increasing computation power of automotive ECUs, the collocation of ADS applications with real-time automotive

applications in a distributed manner should not be a problem, if the ADS has a minimal overhead. *INDRA* framework is not only lightweight but also scalable, and achieves high anomaly detection performance, as discussed in Sect. 6.

An ideal ADS should have low susceptibility to noise, low cost, and a low power/energy footprint. The following are some of the key characteristics of an efficient IDS, which were taken into consideration when designing our *INDRA* ADS:

- *Lightweight*: Anomaly detection tasks can incur additional overhead on ECU, which could result in poor application performance and missed deadlines for real-time applications, which is catastrophic. Therefore, *INDRA* aims to have a lightweight ADS that incurs minimal overhead on the ECU.
- *Few false positives*: This is a highly desired quality in any type of ADS (even outside of the automotive domain), as dealing with false positives can quickly become costly. Thus, a good ADS is expected to have few false positives or false alarms.
- *Coverage*: This defines the range of attacks that an ADS can detect. A good ADS must be capable of detecting more than one type of attack. Moreover, a high coverage for ADS will make the system resistant to multiple attack surfaces.
- *Scalability*: This is an important requirement as the number of ECUs in emerging vehicles is growing along with software and network complexity. A good ADS should be highly scalable and capable of supporting multiple system sizes.

4.2 Communication Model

This subsection discusses vehicle communication model that was considered for *INDRA* framework. *INDRA* primarily focuses on detecting anomalies in controller area network (CAN) bus-based automotive CPS. CAN is the most commonly used in-vehicle network protocol in modern automotive systems. CAN offers a low cost, lightweight, event-triggered communication where messages are transmitted in the form of frames. A typical standard CAN frame structure is shown in Fig. 6, and the length of each field (in bits) is shown on the top. The standard CAN frame consists of a header, payload, and trailer segment. The header contains information of the message identifier (ID) and the length of the message, whereas the payload segment contains the actual data that needs to be transmitted. The trailer section is mainly used for error checking at the receiver. A variation of the CAN protocol, called CAN-extended or CAN 2.0B, is also being deployed increasingly in modern vehicles. The key difference being that CAN extended has a 29-bit identifier, which allows for a greater number of messages IDs.

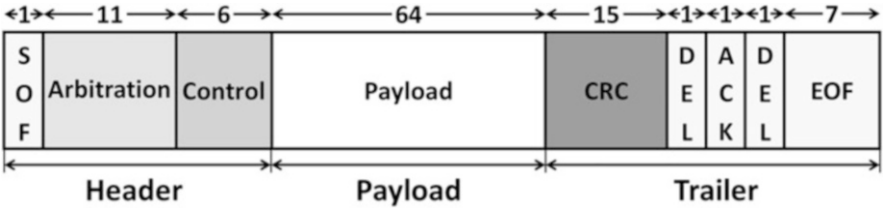


Fig. 6 Standard frame format of a CAN message

Signal Name	Message	Start bit	Length	Byte Order	Value Type
Battery_Current	Status	0	16	Intel	Signed
Battery_Voltage	Status	16	16	Intel	Unsigned
Motor_Current	Status	32	16	Intel	Signed
Motor_Speed	Status	48	8	Intel	Signed
Motor_Direction	Status	56	8	Intel	Unsigned

Fig. 7 An example real-world CAN message with signal information [44]

Our proposed *INDRA* ADS focuses on monitoring the payload segment of the CAN frame and observes for anomalies within the payload to detect cyberattacks. This is because most modern-day attacks involve an attacker modifying the payload to accomplish malicious activities. An attacker can also target the header or trailer segments, but the message would get rejected at the receiver. The payload segment comprises of multiple data entities called signals. An example real-world CAN message with the list of signals within the message is shown in Fig. 7. Each signal has a fixed size (in bits), assigned a particular data type, and a start bit that specifies its location in the 64-bit payload segment of the CAN message.

INDRA focuses on monitoring individual signals within CAN payload to observe for anomalies and detect attacks. During training, *INDRA* learns the temporal dependencies between the messages at a signal level and observes for deviations at runtime to detect attacks. The ability to detect attacks at a signal level enables *INDRA* to not only detect the presence of an attacker but also help in identifying the signal within the message that is under attack. This can be valuable information for understanding the intentions of the attacker, which can be used for developing appropriate countermeasures. The details about the signal level monitoring of *INDRA* ADS are discussed in Sect. 5.2. *Note*: Even though our proposed *INDRA* framework focuses on detecting attacks by monitoring CAN messages, our approach is protocol-agnostic and can be used with other in-vehicle network protocols (such as FlexRay and LIN) with minimal changes.

4.3 Attack Model

Our proposed *INDRA* ADS aims to protect the vehicle from various types of attacks that are most commonly seen and difficult to detect attacks in the domain of automotive CPS. Moreover, these attacks have been widely used in literature to evaluate ADSs.

1. *Flooding attack*: This is the most common and simple to launch attack, and it requires no knowledge of the system. In this attack, the attacker continuously floods the in-vehicle network with a random or specific message with the goal of preventing other ECUs from accessing the bus and rendering the bus unusable. These attacks are typically detected by the vehicle's network bridges and gateways and often do not reach the last line of defense (the ADS). However, it is crucial to consider these attacks as they can have serious consequences when not handled correctly.
2. *Plateau attack*: In this attack, an attacker overwrites a signal value with a constant value for the entirety of the attack interval. The severity of this attack is determined by the magnitude of the jump (increase in signal value) and the duration for which it is held. Larger jumps in signal values are easier to detect compared with shorter jumps.
3. *Continuous attack*: In this attack, an attacker gradually overwrites the signal value with the goal of achieving some target value while avoiding the activation of an ADS. This attack is difficult to detect and can be sensitive to the ADS parameters (discussed in Sect. 5.2).
4. *Suppress attack*: In this attack, the attacker suppresses the signal value(s) by either disabling the target ECU's communication controller or shutting down the ECU. These attacks are easy to detect because they disrupt message transmission for long durations but are harder to detect for shorter durations.
5. *Playback attack*: In this attack, the attacker attempts to trick the ADS by replaying a valid series of message transmissions from the past. This attack is hard to detect if the ADS lacks the ability to capture the temporal relationships between messages and detect when they are violated.

Moreover, in this work, we assume that the attacker can gain access to the vehicle using the most common attack vectors such as connecting to V2X systems that communicate with the outside world (e.g., infotainment and connected ADAS systems), connecting to the OBD-II port, probe-based snooping on the in-vehicle bus, and by replacing an existing ECU. We also assume that the attacker has access to the network parameters (such as parity, flow control, and BAUD rate) that can further assist in gaining access to the in-vehicle network.

Problem objective: The goal of our proposed *INDRA* framework is to implement a lightweight ADS that can detect a variety of attacks (mentioned above) in a CAN-based automotive CPS, with a high detection accuracy and low false positive rate while maintaining a large attack coverage.

5 *INDRA* Framework Overview

INDRA framework enables a machine learning-based signal level ADS for monitoring CAN messages in automotive embedded CPS. An overview of the proposed framework is depicted in Fig. 8. The *INDRA* framework is divided into design-time and runtime steps. During design time, *INDRA* uses trusted CAN message data to train a recurrent autoencoder-based model that learns the normal system behavior. At runtime, the trained recurrent autoencoder model is used to detect anomalies based on deviations from normal system behavior computed using the proposed anomaly score metric. These steps are described in greater detail in the subsequent subsections.

5.1 *Recurrent Autoencoder*

Recurrent autoencoders are powerful neural networks that are designed to behave similar to an encoder–decoder structure but can handle time-series or sequence data as inputs. They can be represented as regular feed-forward neural network-based autoencoders, with neurons that are RNN, LSTM, or GRU units (discussed in Sect. 3). Recurrent autoencoders, like regular autoencoders, have an encoder and a decoder stage. The encoder generates a latent representation of the input data in an n-dimensional space. The decoder uses this latent representation from the encoder output and attempts to reconstruct the input data with minimal reconstruction loss. In *INDRA*, we propose a new lightweight recurrent autoencoder model, which is tailored for the design of ADS to detect cyberattacks in the in-vehicle network data.

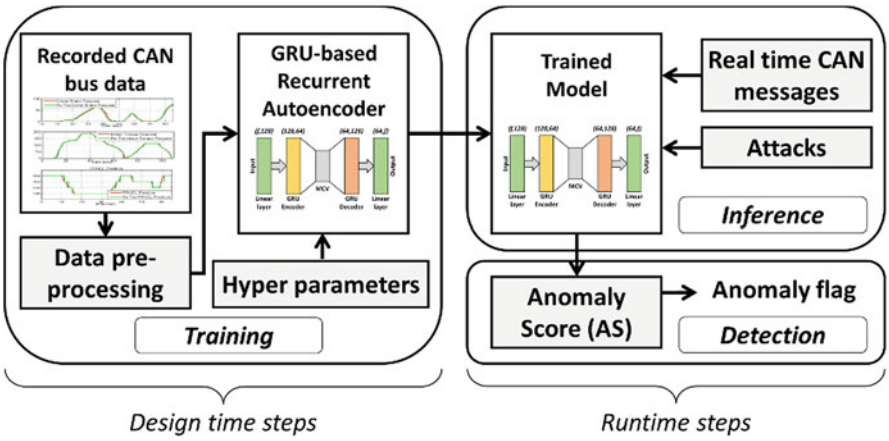


Fig. 8 Overview of *INDRA* ADS framework

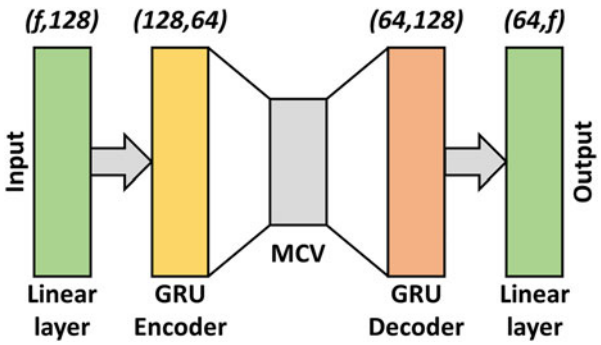


Fig. 9 Proposed model architecture of the recurrent autoencoder used in *INDRA* (f is number of features, i.e., number of signals in the input CAN message, and MCV is message context vector)

The details of the proposed model architecture and the various steps involved in its training and evaluation are discussed in the subsequent sections.

5.1.1 Model Architecture

Our proposed recurrent autoencoder model architecture with the input and output dimensions of each layer is shown in Fig. 9. The model comprises of a linear layer at the input, a GRU-based encoder, a GRU-based decoder, and a linear layer before the final output. The first linear layer receives the input time-series CAN message data with signal level values with f features (where f is the number of signals in the message). The linear layer output is passed to the GRU-based encoder, which generates the latent representation of the time-series signal inputs. This latent representation is referred to as a message context vector (MCV). The MCV captures the context of various signals in the input message in the form of a vector. Each value in the MCV can be viewed as a point in an n-dimensional space containing the context of the series of signal values provided as input. The MCV is fed into a GRU-based decoder, which is then followed by a linear layer to produce the reconstruction of the input CAN message data with individual signal values. The loss between the input and the reconstructed input is calculated using mean square error (MSE), and the weights are updated using backpropagation through time. *INDRA* designs a recurrent autoencoder model for each message ID.

5.1.2 Training Process

The training procedure starts with the preprocessing of the recorded CAN message data from a trusted vehicle. Each sample in the dataset consists of a message ID and the corresponding signal values contained within that message ID. In some cases, the range of signal values can be very large, which can make the training process

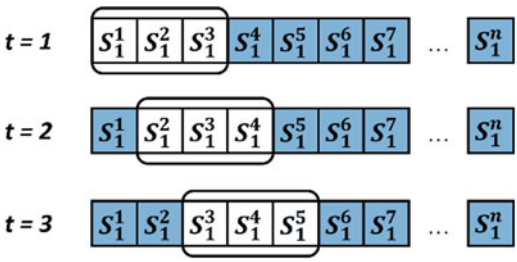


Fig. 10 Example of a rolling window approach

extremely slow or unstable. To prevent this, we scale the signal values between 0 and 1 for each signal type. Moreover, scaling signal values also helps to avoid the problem of exploding gradients (as discussed in Sect. 3).

The preprocessed CAN data is divided into training data (85%) and validation data (15%), which is then prepared for training using a rolling window-based approach. This involves choosing a fixed size window and rolling it to the right by one sample every time step. Figure 10 illustrates a rolling window of size three samples and its movement for the three consecutive time steps. The term S_i^j represents the *i*th signal value at *j*th sample. The elements in the rolling window are referred to as a subsequence, and the size of the subsequence is equal to the size of the rolling window. Our proposed recurrent autoencoder model attempts to learn the temporal relationships that exist between the series of signal values because each subsequence consists of a set of signal values over time. These signal level temporal relationships aid in detecting more complex attacks such as continuous and playback (as discussed in Sect. 4.3). The process of training using subsequences is done iteratively until the end of the training data.

Each iteration during the training process consists of a forward pass and a backward pass (using backpropagation through time to update the weights and biases of the neurons-based on the error value (as discussed in Sect. 3)). The model's performance is evaluated (forward pass only) at the end of the training using the validation data, which was not seen by the model during the training. The model has seen the complete dataset once by the end of validation, which is known as an epoch. The model is trained for a set number of epochs until the model reaches convergence. Moreover, the process of training and validation using subsequences is sped up by training the input subsequences data in groups known as mini-batches. Each mini-batch is made up of several consecutive subsequences that are given as the input to the model in parallel. The size of each mini-batch is referred to as batch size. Finally, a learning rate is defined to control the rate of update of the model parameters during backpropagation phase. These hyperparameters such as subsequence size, batch size, and learning rate are covered in detail in Sect. 6.1.

5.2 Inference and Detection

The trained model is set to evaluation mode at runtime, meaning that only forward passes are performed, and the weights are not updated. During this phase, the trained model is tested under multiple attack scenarios (mentioned in Sect. 4.3), by simulating appropriate attack condition in the CAN message dataset.

Every data sample that passes through the model is reconstructed, and the reconstruction loss is sent to the detection module, which then computes a metric called *anomaly score* (AS). The AS helps in determining whether a signal is anomalous or normal. The AS is calculated at a signal level to predict which signal is under attack. AS is computed as a squared error during each iteration of the inference to estimate the prediction deviation from the input signal value, as shown in (2).

$$\mathrm{AS}_i = \left(S_i^j - \hat{S}_i^j\right)^2 \forall i \in [1, m] \tag{2}$$

where, S_i^j denotes the ith signal value at jth sample, $\hat{S}_i^j$ represents its reconstruction, and m is the number of signals in the message. We observe a large deviation for predicted value from the input signal value (i.e., large AS value), when the current signal pattern is not seen during the training phase and a minimal AS value otherwise. This serves as the foundation for our detection phase.

Since the dataset lacks a signal level anomaly label information, *INDRA* combines the signal level AS information into a message-level AS, by calculating the maximum AS of the signals in that message as shown in (3).

$$\mathrm{MIS} = \max\left(\mathrm{AS}_1, \mathrm{AS}_2, \ldots, \mathrm{AS}_m\right) \tag{3}$$

To achieve adequate detection accuracy, the anomaly threshold (AT) for flagging messages is carefully chosen. *INDRA* investigates multiple choices for AT, using the best model from the training process. The model with the lowest running validation loss, from the training process, is defined as the best model. From this model, multiple metrics such as maximum, mean, median, 99.99%, 99.9%, 99%, and 90% validation loss are logged across all iterations as the choices for the AT. The analysis for selection of the AT metric is presented in detail in Sect. 6.2.

A working snapshot of *INDRA* ADS working in an environment with attacks is illustrated in Fig. 11a, b, with a plateau attack on a message with three signals, between time 0 and 50. Figure 11a compares the input (true) vs ADS predicted signal value comfort three signals. The attack interval is represented by the blue highlighted area. It can be observed that the reconstruction is close for almost all signals except during the attack interval for majority of the time. Signal 3 is subjected to a plateau attack in which the attacker maintains a constant value until the end of attack interval as illustrated in the third subplot of Fig. 11a (note the larger difference between the predicted and actual input signal values in that subplot,

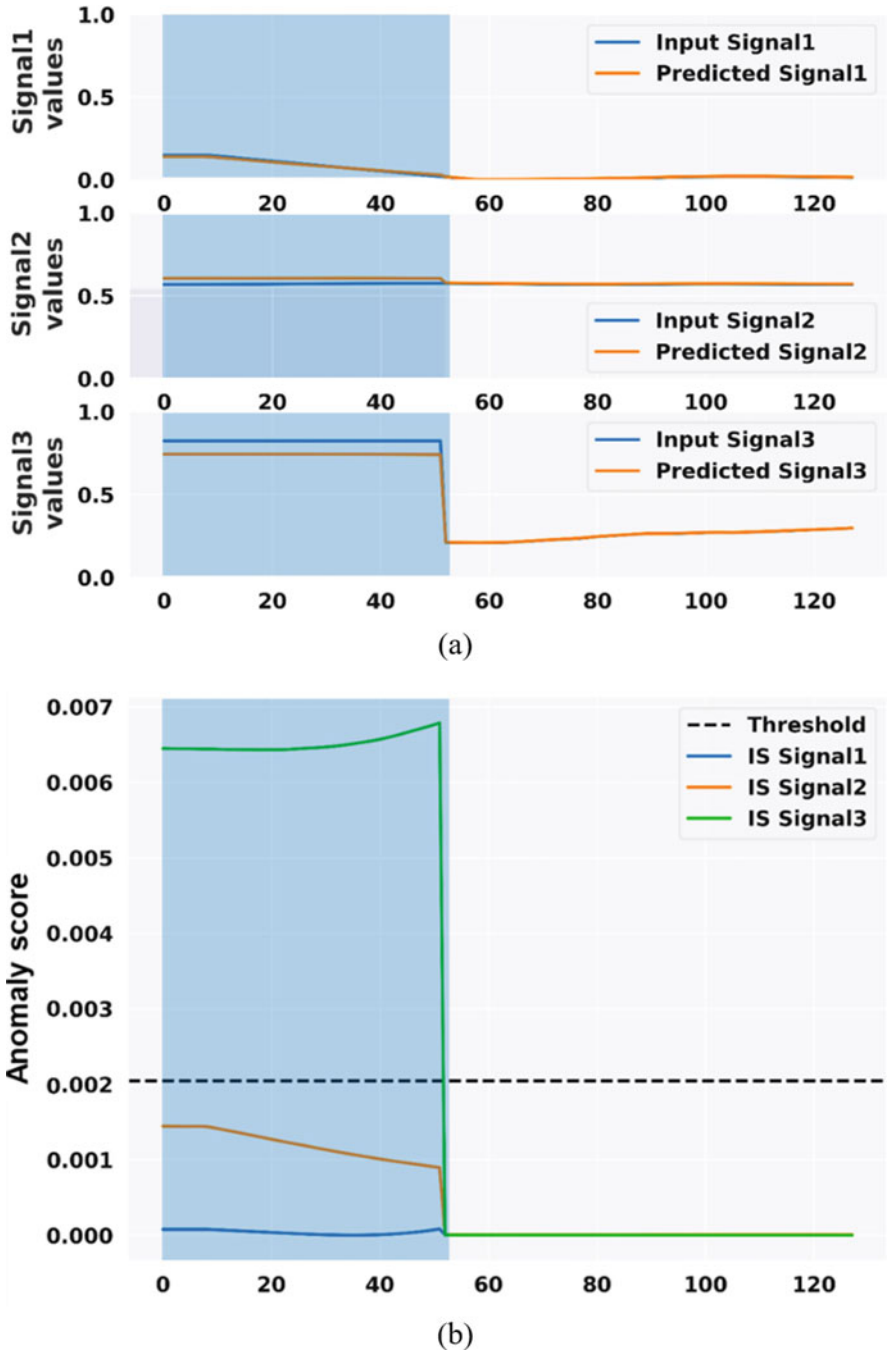


Fig. 11 Working of *INDRA* ADS checking a message with three signals under a plateau attack, where (**a**) shows the signal comparisons and (**b**) shows IS for signals and IS for the message and Anomaly flag

compared to for signals 1 and 2). Figure 11b depicts multiple signal anomaly scores for the three signals. The dotted black line represents the anomaly threshold (AT). As previously stated, the maximum of signal anomaly scores is chosen as message anomaly score (MAS), which in this case is the AS of signal 3. The anomaly score of signal 3 is above the AT, for the entire duration of the attack interval as shown in Fig. 11b, highlighting *INDRA's* ability to detect such attacks. The value of AT (equal to 0.002) in Fig. 11b is calculated using the method described in Sect. 6.2. It is important to note that this value is specific to the example case shown in Fig. 11 and is not the threshold value used for our remaining experiments. The details of AT selection technique are discussed in detail in Sect. 6.2.

6 Experiments

6.1 Experimental Setup

A series of experiments have been conducted to evaluate the performance of our proposed *INDRA* ADS. We begin by presenting an analysis for the selection of anomaly threshold (AT). The derived AT is used to contrast against two variants of the same framework known as *INDRA*-LED and *INDRA*-LD. The former removes the linear layer before the output, essentially leaving the task of decoding the context vector to GRU-based decoder. The abbreviation LED stands for (L) linear layer, (E) encoder GRU, and (D) decoder GRU. The second variation substitutes a series of linear layers for the GRU and the linear layer at the decoder (LD stands for linear decoder). These experiments were carried to assess the importance of different layers in the network. However, the encoder side of the network is not changed because it is required to generate an encoding of the time-series data. *INDRA* investigates other variants as well, but they were not included in the discussion as their performance was lesser compared with that of LED and LD variants.

Subsequently, the best *INDRA* variant is compared with three prior works: predictor LSTM (PLSTM [38]), replicator neural network (RepNet [39]), and CANet [36]. The first comparison work (PLSTM) employs an LSTM-based network that has been trained to predict the signal values in the following message transmission. PLSTM accomplishes this by taking the 64-bit CAN message payload as the input and learning to predict the signal at a bit-level granularity by minimizing prediction loss. The bit level deviations between the real and the predicted next signal values are monitored using a log loss or binary cross-entropy loss function. PLSTM uses the prediction loss values during runtime to decide whether a particular message is anomalous or not. The second comparison work (RepNet) employs a series of RNN layers to increase the dimensionality of the input data and reconstruct the signal values by decreasing back to the original dimensionality. RepNet accomplishes this by reducing the mean squared error between the input and the reconstructed signal values. At runtime, large deviations between the input received signal and

the reconstructed signal values are used to detect attacks. Finally, CANet uses a quadratic loss function to minimize the signal reconstruction error by combining multiple LSTMs and linear layers in an autoencoder architecture. All experiments conducted with *INDRA* and its variants and prior works are discussed in subsequent subsections.

The SynCAN dataset developed by ETAS and Robert Bosch GmbH [36] was used to evaluate INDRA framework with its variants and against prior works. The dataset contains CAN message data for ten different IDs that have been modeled after real-world CAN message data. Furthermore, the dataset consists of both training and test data with multiple attacks (discussed in Sect. 4.3). Each row in the dataset contains a timestamp, message ID, and individual signal values. In addition, the test data contains a label column with either 0 or 1 values indicating normal or anomalous messages. The label information is available per message basis and does not specify which signal within the message is under attack. This label information is used to evaluate the proposed ADS over several metrics such as detection accuracy and false positive rate and is discussed in detail in the next subsections. Moreover, to simulate a more realistic attack scenario in the in-vehicle networks, the test data also contains normal CAN traffic between the attack injections. *Note*: The label information in the training data is not used to train *INDRA* model, as INDRA model learns the patterns in the input data in an unsupervised manner.

All the machine learning-based frameworks including the *INDRA* framework and its variants as well as comparison works are implemented using Pytorch 1.4. *INDRA* conducts various experiments to select the best performing model hyperparameters (number of layers, hidden unit sizes, and activation functions). The final model discussed in Sect. 5.1 was trained using the SynCAN data set, with 85% of train data used for training and the remaining for validation. The validation data is primarily used to assess the model performance at the end of each epoch. The model is trained for 500 epochs, using a rolling window approach (as discussed in Sect. 5.1.2) with the subsequence size of 20 messages and the batch size of 128. Moreover, an early stopping mechanism is implemented to monitor the validation loss across epochs and stop the training process if there is no improvement after 10 (patience) epochs. The initial learning rate is chosen as 0.0001, and tanh activations are applied after each linear and GRU layers. Furthermore, ADAM optimizer is used with the mean squared error (MSE) as the loss criterion. The trained model parameters were used during testing and considered multiple test data inputs to simulate attack scenarios. The anomaly score metric (as stated in Sect. 5) was used to calculate the anomaly threshold to flag the message as anomalous or normal. To evaluate the model performance, several performance metrics such as detection accuracy and false positive rate were considered. All the simulations were executed on an AMD Ryzen-9 3900X server with an Nvidia GeForce RTX 2080Ti GPU.

Finally, before the experimental results section, we present the following definitions in the context of ADS:

- True positive (TP) – when the ADS detects an actual anomalous message as an anomaly

- False negative (FN) – when the ADS detects an actual anomalous message as normal
- False positive (FP) – when the ADS detects a normal message as an anomaly (aka false alarm)
- True negative (TN) – when the ADS detects an actual normal message as normal.

INDRA framework focuses on two key performance metrics: (i) *detection accuracy*, a measure of ADS ability to detect anomalous messages correctly, and (ii) *false positive rate*, also known as false alarm rate. These metrics are computed as shown in (4) and (5):

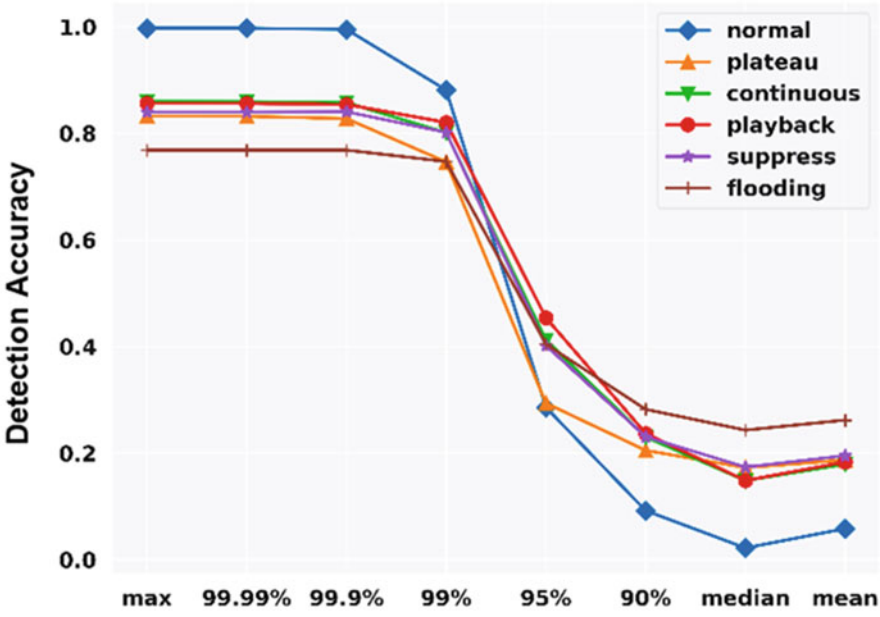
$$\text{Detection accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FN} + \text{FP} + \text{TN}} \tag{4}$$

$$\text{False positive rate} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{5}$$

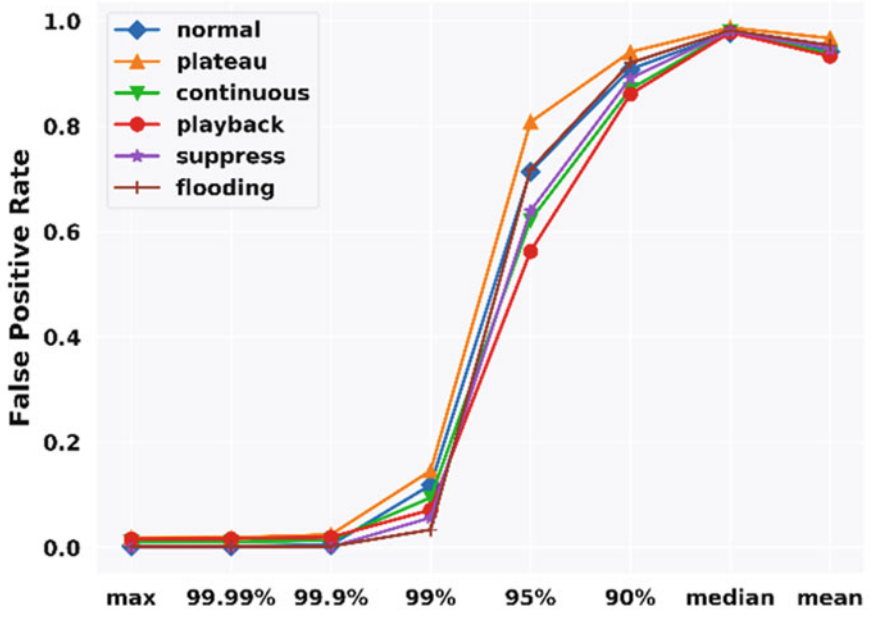
6.2 Anomaly Threshold Selection

This subsection presents a detailed analysis on the selection of anomaly threshold (AT) by considering various options such as max, median, mean, and different quantile bins of validation loss of the final model. The idea is that the model reconstruction error for the normal message should be much smaller than the error for anomalous messages. Hence, *INDRA* explores several candidate options to achieve this goal that would work across multiple attack and no-attack scenarios. A high threshold value can make it harder for the model to detect the attacks that change the input pattern minimally (e.g., continuous attack). On the other hand, having a small threshold value can cause multiple false alarms, which is highly undesirable. Hence, it becomes crucial to select an appropriate threshold value to optimize the performance of the model.

Figure 12a, b shows the detection accuracy and false positive rate, respectively, for various candidate options to calculate AT under different attack scenarios. The results from the Fig. 12 indicate that selecting higher validation loss as the AT can lead to a high accuracy and low false alarm rate. However, selecting a very high value (e.g., "max" or "99.99 percentile") may result in missing small variations in the input patterns that are found in more sophisticated attacks. We empirically conclude that the maximum and 99.99 percentile values to be very close. To capture attacks that produce small deviations, a slightly smaller threshold value is selected that would still perform similar to max and 99.99 percentile thresholds on all of the current attack scenarios. Therefore, *INDRA* chooses the 99.9th percentile value of the validation loss as the value of the anomaly threshold (AT) and uses the same AT value for the remainder of the experiments discussed in the next subsections.



(a)



(b)

Fig. 12 Comparison of (**a**) detection accuracy and (**b**) false positive rate for various choices of anomaly threshold (AT) as a function of validation loss under different attack scenarios (% refers to percentile not percentage)

6.3 Comparison of INDRA Variants

After selecting the correct anomaly threshold from the previous subsection, we use that same criterion for evaluating against two other variants: *INDRA*-LED and *INDRA*-LD. The main intuition behind evaluating different variants of *INDRA* is to investigate the impact of different types of layers in the model on the performance metrics discussed in Sect. 6.1.

Figure 13a illustrates the detection accuracy for *INDRA* framework and its variants on *y*-axis with multiple types of attacks and for a no-attack scenario (normal) on the *x*-axis. It can be clearly seen that *INDRA* outperforms the other two variants and has high accuracy in most of the attack scenarios. It should be noted that the high accuracy is achieved by monitoring at a signal level as opposed to prior works that monitors at the message level.

Figure 13b illustrates the false positive rate or false alarm rate of *INDRA* and other variants under different attack scenarios. When compared with other variants, *INDRA* has the lowest false positive rate and highest detection accuracy. Moreover, *INDRA*-LED, which is just short of a linear layer at the decoder end, is the second-best performing model after *INDRA*. The ability of *INDRA*-LED to use a GRU-based decoder helps in reconstructing the MCV back to original signals. It can be clearly seen in both Fig. 13a, b that the absence of GRU layers on the output decoder end for *INDRA*-LD results in significant performance degradation. As a result, *INDRA* is chosen as the candidate model for subsequent experiments.

6.4 Comparison with Prior Works

Our proposed *INDRA* framework is compared with some of the best-known prior works in the ADS area such as PLSTM [38], RepNet [39], and CANet [36]. Figure 14a, b shows the detection accuracy and false positive rate, respectively, for the various techniques under different attack scenarios.

From the Fig. 14a, b, it is evident that *INDRA* achieves high accuracy for each attack scenario while also achieving low positive rates. The ability to monitor signal level variations combined with more cautious selection of anomaly threshold gives *INDRA* an advantage over comparison works. PLSTM and RepNet use the maximum validation loss in the final model as the threshold, whereas CANet uses interval-based monitoring to detect anomalous messages. Choosing a higher threshold helped PLSTM to achieve slightly lower false positive rates for some scenarios, but it hurt the ability of both PLSTM and RepNet to detect attacks with minor variations in the input data. This is because the deviations produced by some of the complex attacks are small and the attacks go undetected due to the large thresholds. Moreover, CANet's interval-based monitoring struggles to find an optimal value for the thresholds. Lastly, the false positive rates of *INDRA* remain significantly low with the maximum of 2.5% for plateau attacks. It should be noted

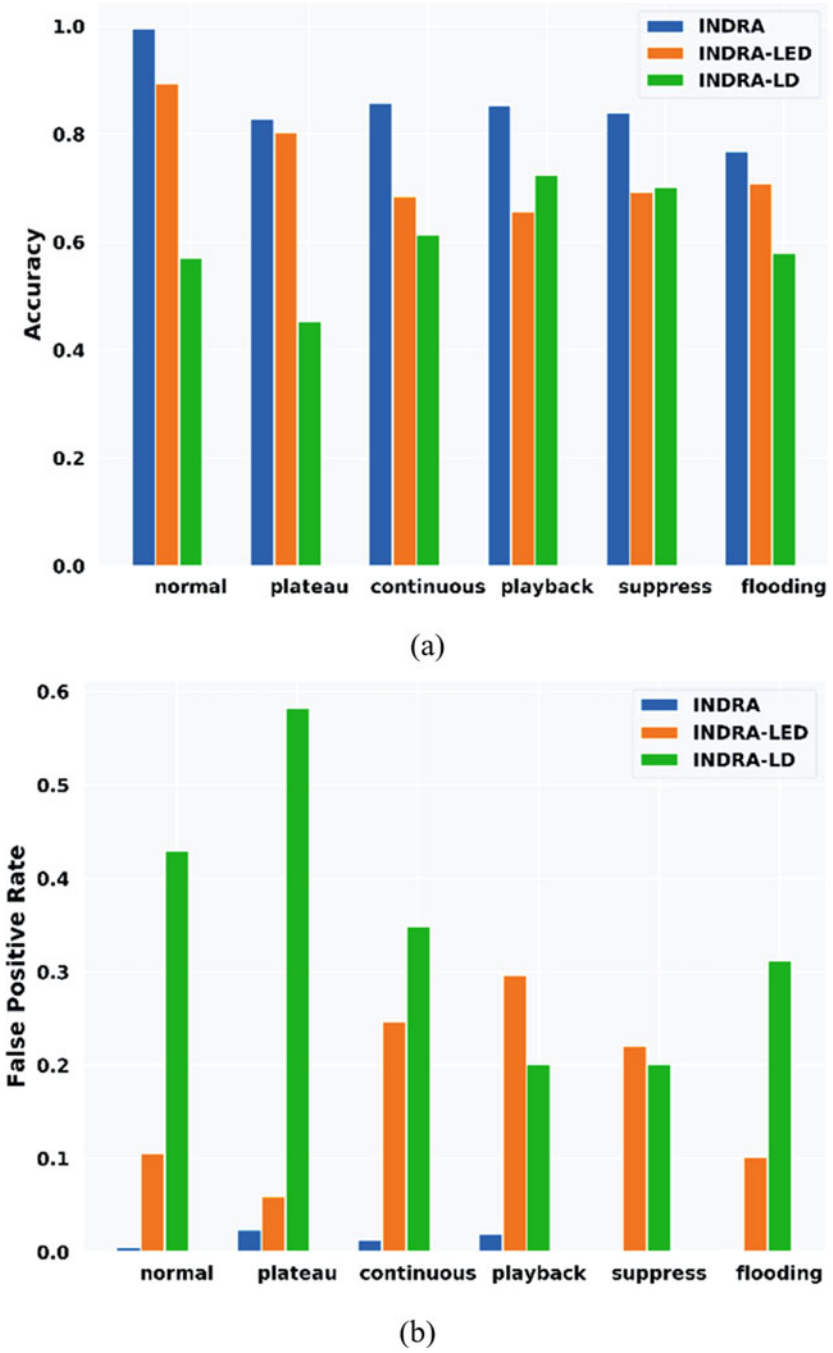


Fig. 13 Comparison of (**a**) detection accuracy and (**b**) false positive rate under different attack scenarios for *INDRA* and its variants (*INDRA*-LED and *INDRA*-LD)

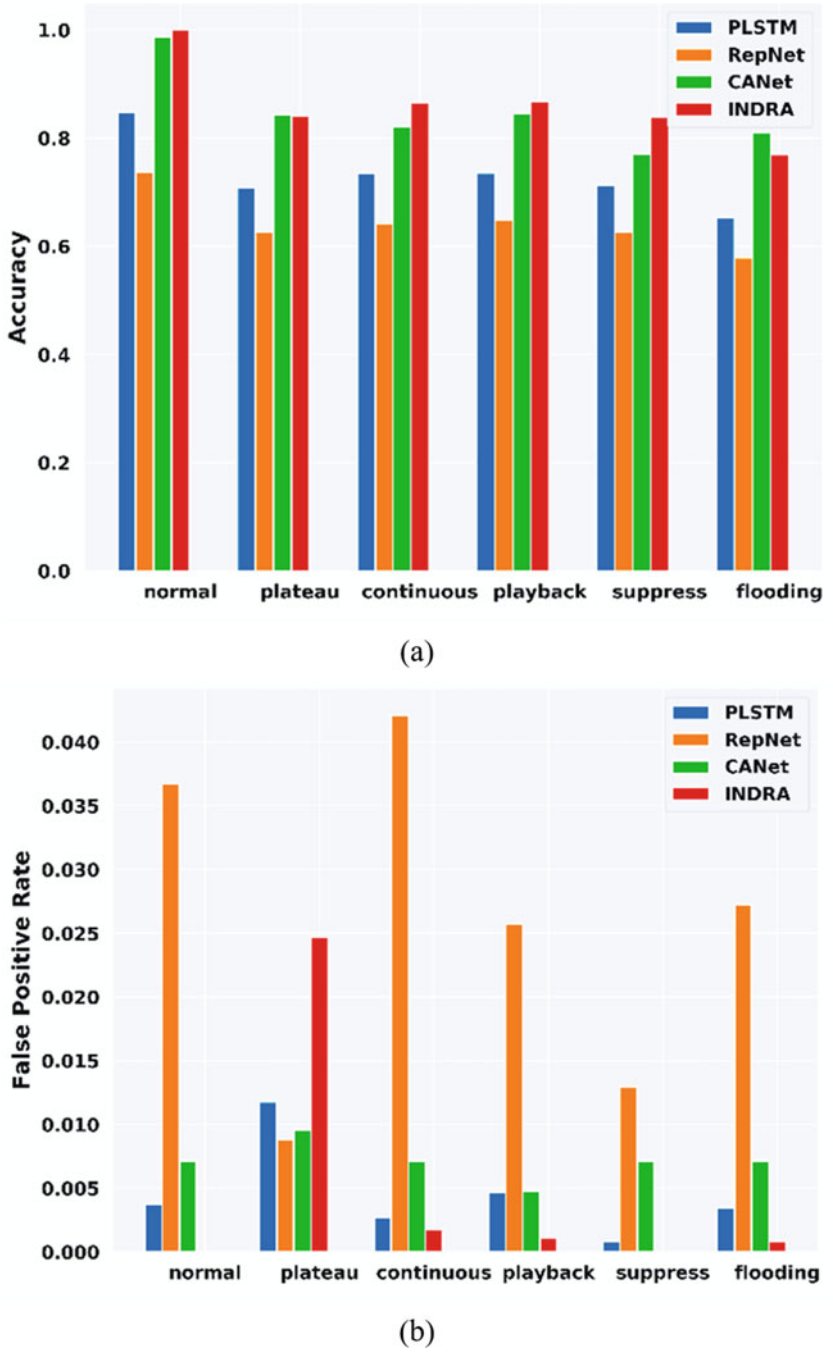


Fig. 14 Comparison of (**a**) detection accuracy and (**b**) false positive rate of *INDRA* and the prior works PLSTM [38], RepNet [39] and CANet [36]

Table 2 Memory footprint comparison between our proposed *INDRA* framework and the prior works PLSTM [38], REPNET [39], and CANET [36]

ADS framework	Memory footprint (KB)
PLSTM [38]	13,417
RepNet[39]	55
CANet [36]	8718
INDRA	443

that the *y*-axis in Fig. 14b has a much smaller scale than in Fig. 14a, and the magnitude of the false positive rate is very small.

6.5 ADS Overhead Analysis

A detailed analysis of the overhead incurred by our proposed *INDRA* ADS is discussed in this subsection. The overhead is quantified in terms of both memory footprint and time taken to process an incoming message, i.e., inference time. The former metric is important because the automotive ECUs are highly resource constrained and have limited memory and compute capacities. Therefore, having a low memory overhead is crucial to avoid interference with real-time automotive applications. The inference time metric not only provides important information about the time it takes to detect the attacks but also can be used to compute the utilization overhead on the ECU. Thus, the abovementioned two metrics are used to analyze the overhead and quantify the lightweight nature of INDRA ADS.

To accurately capture the overhead of our proposed *INDRA* framework and the prior works, we implemented the ADSs on an ARM Cortex- A57 CPU on a Jetson TX2 board, which has similar specifications to the state-of-the-art multi-core ECUs. Table 2 shows the memory footprint of *INDRA* framework and the prior works mentioned in the previous subsections. It is clear that *INDRA* framework has a low memory footprint compared with the prior works, except for the RepNet [39]. However, it is important to observe that even though *INDRA* framework has slightly higher memory footprint compared with the RepNet [39], *INDRA* outperforms all the prior works including RepNet [39] in all performance metrics under multiple attack scenarios, as shown in Fig. 14. The heavier (high memory footprint) models can capture a wide range of system behaviors; however, they are not an ideal choice for resource constrained automotive CPS. On the contrary, a much lighter model (such as RepNet) fails to capture crucial details about the system behavior due to its limited model parameters, which in turn suffers from performance issues.

In order to understand the inference overhead, we benchmarked the different ADS frameworks on an ARM Cortex- A57 CPU. In this experiment, different system configurations are considered to encompass a wide variety of ECU hardware that is available in the state-of-the-art vehicles. Based on the available hardware resources, a single core (employs only one CPU core) and dual core (employs

Table 3 Inference time comparisons between our proposed *INDRA* framework and the prior works PLSTM [38], REPNET [39], and CANET [36] using single and dual core configurations

	Average inference time (μs)	
ADS framework	Single core ARM Cortex A57 CPU	Dual core ARM Cortex A57 CPU
PLSTM [38]	681.18	644.76
RepNet [39]	19.46	21.46
CANet [36]	395.63	378.72
INDRA	80.35	72.91

two CPU cores) system configurations were selected on the Jetson TX2. The ADS frameworks are executed ten times for the different CPU configurations, and the average inference time (in μs) are recorded in Table 3. From the results in Table 3, it is evident that the *INDRA* framework has significantly faster inference times compared with the prior works (excluding RepNet) under all configurations. This is partly due to the lower memory footprint of *INDRA* framework. As previously stated, even though RepNet has a lower inference time, it has the worst performance of any compared framework, as shown in Fig. 14. The large inference times for the better performing frameworks can have an impact on the real-time performance of the control systems in the vehicle and can result in catastrophic deadline misses. We also believe that using a dedicated deep learning accelerator (DLA) further enhance the performance of the ADS models.

Thus, from Fig. 14 and Tables 2 and 3, it is clear that *INDRA* achieves a clear balance of having superior anomaly detection performance while maintaining low memory footprint and fast inference times, making it a powerful and lightweight ADS solution.

6.6 Scalability Results

In this subsection, an analysis on the scalability of *INDRA* framework is presented by studying the system performance using the ECU utilization metric as a function of increasing system complexity (number of ECUs and messages). Each ECU in the system has a real-time utilization (U_{RT}) and an ADS utilization (U_{ADS}) from running real-time and ADS applications, respectively. We primarily focus on analyzing the ADS overhead (U_{ADS}), as it is a measure of the compute efficiency of the ADS. Since the safety-critical messages monitored by the ADS are periodic in nature, the ADS can be modeled as a periodic application with period that is the same as the message period [5]. Thus, monitoring an ith message m_i results in an induced ADS utilization ($U_{\text{ADS}, mi}$) at an ECU, which can be calculated as:

$$U_{\text{IDS},m_i} = \left(\frac{T_{\text{IDS}}}{P_{m_i}} \right) \tag{6}$$

where T_{ADS} and P_{mi} denote the time taken by the ADS to process one message (inference time) and the period of the monitored message, respectively. Moreover, the sum of all ADS utilizations as a result of monitoring different messages is the overall ADS utilization at that ECU (U_{ADS}) and is given by:

$$U_{\text{IDS}} = \sum_{i=1}^{n} U_{\text{IDS},m_i} \tag{7}$$

To evaluate the scalability of INDRA, six different system sizes were considered. Moreover, a pool of commonly used message periods {1, 5, 10, 15, 20, 25, 30, 45, 50, 100} (all periods in ms) in automotive CPS is considered to sample uniformly, when assigning periods to the messages in the system. These messages are distributed evenly among different ECUs and the ADS utilization is calculated using (6) and (7). *INDRA* assumes a pessimistic scenario where all the ECUs in the system have only a single core. This would allow us to analyze the worst case overhead of the ADS.

Figure 15 shows the average ECU utilization for different system sizes denoted by $\{p, q\}$, where p is the number of ECUs and q is the number of messages in the system. In this work, a very pessimistic estimate of 50% real-time ECU utilization for real-time automotive applications ("RT Util", as shown in the dotted bars) is

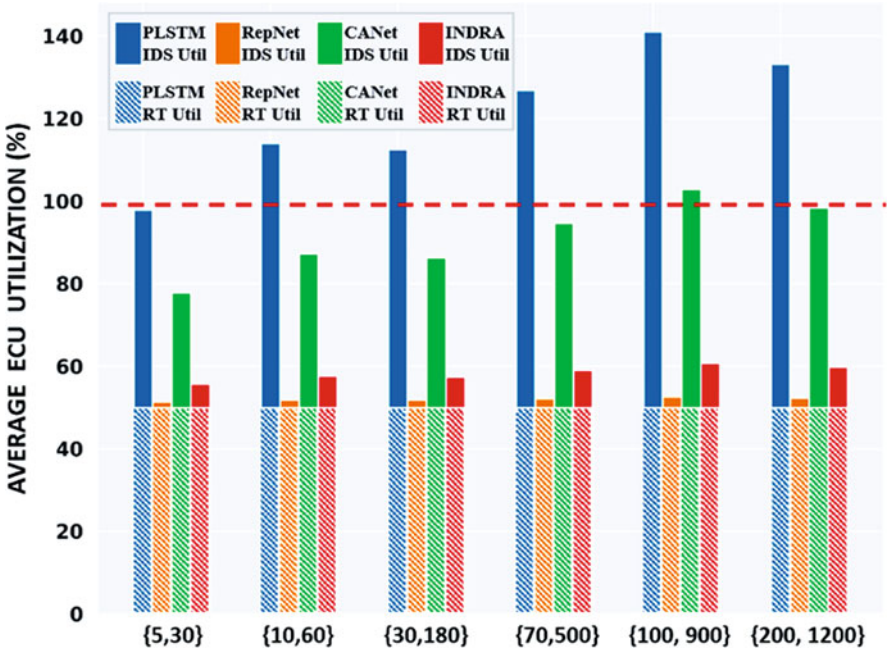


Fig. 15 Scalability analysis of our proposed *INDRA* ADS for different system sizes and the prior works PLSTM [38], RepNet [39], and CANet [36]

assumed. The solid bars on top of the dotted bars represent the overhead incurred by the ADS executing on the ECUs, and the red horizontal dotted line represents the 100% ECU utilization mark. It is critical to avoid exceeding the 100% ECU utilization limit under any scenario, as it could create undesired latencies resulting in missing deadlines, for time-critical automotive applications that can be catastrophic. It is clear from the results that the prior works such as PLSTM and CANet incur heavy overhead on the ECUs while RepNet and our proposed *INDRA* framework have very minimal overhead that is favorable to increasing system sizes. From the results in this section (Figs. 14 and 15; Tables 2 and 3), it is apparent that not only does *INDRA* achieve better performance in terms of both accuracy and low false positive rate for anomaly detection than state-of-the-art prior work but also is lightweight and highly scalable.

7 Conclusion

In this chapter, we presented a novel recurrent autoencoder-based lightweight anomaly detection system called *INDRA* for distributed automotive cyber-physical systems. *INDRA* framework uses a metric called anomaly score (AS) to measure the deviation of the prediction signal from the actual input. *INDRA* also presents a thorough analysis of our anomaly threshold selection process and compared with the best-known prior works in this area. The promising results indicate a compelling potential for utilizing our proposed approach in emerging automotive platforms.

Acknowledgments This work was supported by the National Science Foundation (NSF), through grant CNS-2132385.

References

1. Kukkala, V.K., Bradley, T., Pasricha, S.: Priority-based multi-level monitoring of signal integrity in a distributed powertrain control system. In: Proceedings of IFAC Workshop on Engine and Powertrain Control, Simulation and Modeling. IEEE (2015)
2. Kukkala, V.K., Bradley, T., Pasricha, S.: Uncertainty analysis and propagation for an auxiliary power module. In: Proceedings of IEEE Transportation Electrification Conference (TEC). IEEE (2017)
3. Kukkala, V.K., Pasricha, S., Bradley, T.: JAMS: Jitter-aware message scheduling for FlexRay automotive networks. In: Proceedings of IEEE/ACM International Symposium on Network-on-Chip (NOCS). IEEE (2017)
4. Kukkala, V.K., Pasricha, S., Bradley, T.: JAMS-SG: a framework for Jitter-aware message scheduling for time-triggered automotive networks. ACM Trans. Des. Autom. Electron. Syst. **24**(6), 1–31 (2019)
5. Kukkala, V., Pasricha, S., Bradley, T.: SEDAN: security-aware design of time-critical automotive networks. IEEE Trans. Veh. Technol. **69**(8), 9017–9030 (2020)
6. Kukkala, V.K., Thiruloga, S.V., Pasricha, S.: INDRA: intrusion detection using recurrent autoencoders in automotive embedded systems. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **39**(11), 3698–3710 (2020)

7. Kukkala, V.K., Thiruloga, S.V., Pasricha, S.: LATTE: LSTM self-attention based anomaly detection in embedded automotive platforms. ACM Trans. Embed. Comput. Syst. **20**(5s, Article 67), 1–23 (2021)
8. Thiruloga, S.V., Kukkala, V.K., Pasricha, S.: TENET: temporal CNN with attention for anomaly detection in automotive cyber-physical systems. In: Proceedings of IEEE/ACM Asia & South Pacific Design Automation Conference (ASPDAC). IEEE (2022)
9. Kukkala, V.K., Thiruloga, S.V., Pasricha, S.: Roadmap for cybersecurity in autonomous vehicles. In: IEEE Consumer Electronics Magazine (CEM). IEEE (2022)
10. Tunnell, J., Asher, Z., Pasricha, S., Bradley, T.H.: Towards improving vehicle fuel economy with ADAS. SAE Int. J. Connected Autom. Veh. **1**(2), 81 (2018)
11. Tunnell, J., Asher, Z., Pasricha, S., Bradley, T.H.: Towards improving vehicle fuel economy with ADAS. In: Proceedings of SAE World Congress Experience (WCX). SAE Technical Paper (2018)
12. Asher, Z., Tunnell, J., Baker, D.A., Fitzgerald, R.J., Banaei-Kashani, F., Pasricha, S., Bradley, T.H.: Enabling prediction for optimal fuel economy vehicle control. In: Proceedings of SAE World Congress Experience (WCX). SAE Technical Paper (2018)
13. Dey, J., Taylor, W., Pasricha, S.: VESPA: a framework for optimizing heterogeneous sensor placement and orientation for autonomous vehicles. IEEE Consum. Electron. Mag. **10**(2), 16 (2021)
14. Kukkala, V.K., Pasricha, S., Bradley, T.: Advanced driver-assistance systems: a path toward autonomous vehicles. IEEE Consum. Electron. Mag. **7**(5), 18 (2018)
15. Koscher, K., Czeskis, A., Roesner, F., Patel, S., Kohno, T., Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S.: Experimental security analysis of a modern automobile. In: Proceedings of IEEE Symposium on Security and Privacy (SP). IEEE (2010)
16. Miller, C., Valasek, C.: Remote exploitation of an unaltered passenger vehicle. Black Hat USA (2015)
17. Izosimov, V., Asvestopoulos, A., Blomkvist, O., Törngren, M.: Security-aware development of cyber-physical systems illustrated with automotive case study. In: Proceedings of IEEE/ACM Design, Automation & Test in Europe & Exhibition (DATE). IEEE (2016)
18. Studnia, I., Alata, E., Nicomette, V., Kaâniche, M., Laarouchi, Y.: A language-based intrusion detection approach for automotive embedded networks. Int. J. Embed. Syst. **10**(8), 1–12 (2018)
19. Marchetti, M., Stabili, D.: Anomaly detection of CAN bus messages through analysis of ID sequences. In: Proceedings of IEEE Intelligent Vehicle Symposium (IV). IEEE (2017)
20. Hoppe, T., Kiltz, S., Dittmann, J.: Security threats to automotive CAN networks- practical examples and selected short-term countermeasures. Reliab. Eng. Syst. Saf. **96**(1), 11 (2011)
21. Larson, U.E., Nilsson, D.K., Jonsson, E.: An approach to specification-based attack detection for in-vehicle networks. In: Proceedings of IEEE Intelligent Vehicles Symposium (IV). IEEE (2008)
22. Aldwairi, M., Abu-Dalo, A.M., Jarrah, M.: Pattern matching of signature-based IDS using Myers algorithm under MapReduce framework. EURASIP J. Inf. Secur. **2017**(1), 1–11 (2017)
23. Myers, E.W.: An O(ND) difference algorithm and its variations. Algorithmica. **1**, 251–266 (1986)
24. Hoppe, T., Kiltz, S., Dittmann, J.: Applying intrusion detection to automotive IT-early insights and remaining challenges. J. Inf. Assur. Secur. **4**(6), 226–235 (2009)
25. Waszecki, P., Mundhenk, P., Steinhorst, S., Lukasiewycz, M., Karri, R., Chakraborty, S.: Automotive electrical and electronic architecture security via distributed in-vehicle traffic monitoring. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **36**(11), 1790–1803 (2017)
26. Cho, K.T., Shin, K.G.: Fingerprinting electronic control units for vehicle intrusion detection. In: Proceedings of USENIX. USENIX Association (2016)
27. Ying, X., Sagong, S.U., Clark, A., Bushnell, L., Poovendran, R.: Shape of the cloak: formal analysis of clock skew-based intrusion detection system in controller area networks. IEEE Trans. Inf. Forensics Secur. **14**(9), 2300–2314 (2019)

28. Yoon, M.K., Mohan, S., Choi, J., Sha, L.: Memory heat map: anomaly detection in real-time embedded systems using memory behavior. In: Proceedings of IEEE/ACM/EDAC Design Automation Conference (DAC). IEEE (2015)
29. Müter, M., Asaj, N.: Entropy-based anomaly detection for in-vehicle networks. In: Proceedings of IEEE Intelligent Vehicles Symposium (IV). IEEE (2011)
30. Müter, M., Groll, A., Freiling, F.C.: A structured approach to anomaly detection for in-vehicle networks. In: Proceedings of IEEE International Conference on Intelligent and Advanced System (ICIAS). IEEE (2010)
31. Taylor, A., Japkowicz, N., Leblanc, S.: Frequency-based anomaly detection for the automotive CAN bus. In: Proceedings of World Congress on Industrial Control Systems Security (WCICSS). IEEE (2015)
32. Martinelli, F., Mercaldo, F., Nardone, V., Santone, A.: Car hacking identification through fuzzy logic algorithms. In: Proceedings of IEEE International Conference on Fuzzy Systems (FUZZ-IEEE). IEEE (2017)
33. Vuong, T.P., Loukas, G., Gan, D.: Performance evaluation of cyber-physical intrusion detection on a robotic vehicle. In: Proceedings of IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM). IEEE (2015)
34. Levi, M., Allouche, Y., Kontorovich, A.: Advanced analytics for connected car cybersecurity. In: Proceedings of IEEE Vehicular Technology Conference (VTC). Springer (2018)
35. Kang, M.J., Kang, J.W.: A novel intrusion detection method using deep neural network for in-vehicle network security. In: IEEE Proceedings of Vehicular Technology Conference (VTC). Springer (2016)
36. Hanselmann, M., Strauss, T., Dormann, K., Ulmer, H.: CANet: an unsupervised intrusion detection system for high dimensional CAN bus data. In: IEEE Access, vol. 8, p. 58194 (2020)
37. Loukas, G., Vuong, T., Heartfield, R., Sakellari, G., Yoon, Y., Gan, D.: Cloud-based cyber-physical intrusion detection for vehicles using deep learning. IEEE Access. **6**(1), 3491–3508 (2018)
38. Taylor, A., Leblanc, S., Japkowicz, N.: Anomaly detection in automobile control network data with long short-term memory networks. In: Proceedings of IEEE International Conference on Data Science and Advanced Analytics (DSAA). IEEE (2016)
39. Weber, M., Wolf, G., Sax, E., Zimmer, B.: Online detection of anomalies in vehicle signals using replicator neural networks. In: Proceedings of ESCAR USA (2018)
40. Weber, M., Klug, S., Sax, E., Zimmer, B.: Embedded hybrid anomaly detection for automotive can communication. In: Embedded Real Time Software and Systems (ERTS) (2018)
41. Schmidhuber, J.: Habilitation thesis: system modeling and optimization (1993)
42. Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J.: Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In: A Field Guide to Dynamical Recurrent Neural Networks. IEEE Press (2001)
43. Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint, arXiv:1406.1078 (2014)
44. DiDomenico, G.C., Bair, J., Kukkala, V.K., Tunnell, J., Peyfuss, M., Kraus, M., Ax, J., Lazarri, J., Munin, M., Cooke, C., Christensen, E.: Colorado State University EcoCAR 3 final technical report. In: SAE World Congress Experience (WCX). SAE Technical Paper (2019)

MELETI: A Machine-Learning-Based Embedded System Architecture for Infrastructure Inspection with UAVs

Marios Pafitis, Antonis Savva, Christos Kyrkou, Panayiotis Kolios, and Theocharis Theocharides

1 Introduction

Modern societies depend heavily on critical infrastructure systems that put pressure on operators such as Electricity and Telecommunication Authorities (EAs/TAs) to support the increasing demands [1, 2]. For instance, the power transmission is operated across many kilometers of low-voltage (LV), medium-voltage (MV), and high-voltage (HV) networks, while telecommunication networks are consistently increasing their deployment density to support the increasing user base.

The infrastructures of those authorities are frequently exposed to extreme weather conditions and span across large areas with harsh environments [1, 3, 4], leading to expensive monitoring, maintenance, and upgrade operations that significantly affect the provided quality of experience (QoE) and quality of service (QoS) [1, 4]. Evidently, potential malfunctions can cause power outages, telecommunication network outages, and even fire outbreaks [1, 4], urging EAs and TAs to take preventive measures to avoid those infrastructure failures and minimize their financial and environmental impacts [1, 4, 5]. For instance, half-hour and eight-hour blackouts in the USA approximately cost the operators $16,000 and $94,000, respectively [4]. This initiates a domino effect that can end up in millions of dollars in financial losses [5, 6].

To lessen these outcomes, EAs and TAs periodically conduct inspections on their infrastructures with qualified workers that are sent across the power lines or the telecommunication base stations on foot or with helicopters [4]. In most cases, the trained inspectors visually assess the condition of the infrastructure, which may

M. Pafitis · A. Savva · C. Kyrkou · P. Kolios · T. Theocharides (✉)
KIOS Research and Innovation Center of Excellence and the Department of Electrical and Computer Engineering, University of Cyprus, Nicosia, Cyprus
e-mail: savva.d.antonis@ucy.ac.cy; kyrkou.christos@ucy.ac.cy; kolios.panayiotis@ucy.ac.cy; ttheocharides@ucy.ac.cy

S. Pasricha, M. Shafique (eds.), *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing*, https://doi.org/10.1007/978-3-031-40677-5_12

include the use of binoculars or dedicated cameras to detect specific malfunctions, such as increased temperatures and corona effects on the power insulators [1, 4, 7], or corrosion on the base stations and antenna damages [8]. This personnel is always at risk in conducting either of these inspection, while fatigue levels affect dramatically the efficiency of the whole process [4].

Within this context, UAVs could stem as a promising and flexible solution for all the different needs in infrastructure inspection. UAVs can collect high-quality data from a plethora of sensors, such as high-resolution visible imaging sensors, infrared thermal imaging cameras, light detection and ranging (LiDAR), gyroscopes, accelerometers, magnetometers, GPS/GNSS, and more. Additionally, UAVs can easily reach the infrastructure to be inspected, especially in those cases where it is impractical or extremely dangerous for the inspector to do so. Besides, in comparison with helicopter-based approaches, the associated operating cost for a UAV is significantly lower [1]. This cost is projected to decrease in the future due to the widespread commercial availability of UAVs [1, 4]. Furthermore, there is an increasing interest in developing autonomous systems that perform multi-sensor real-time data acquisition for detecting defects on infrastructures. The main limitation of such efforts is that the remote pilot needs to control the flight process by precisely positioning the UAV and the sensors, to correctly and accurately collect the data. In addition, using a single Global Navigation Satellite System (GNSS), e.g., GPS for high-precision positioning, leads to many inaccuracies [9–12].

As part of this study, we exploit recent advances in UAV technologies, in deep-learning-based detection algorithms and embedded hardware to develop MELETI, a UAV-based architecture that automates the infrastructure inspection of telecommunication and power networks, during both the phase of data acquisition and analysis. With MELETI, multiple sensors can be integrated on the UAV based on the application needs. The sensors are programmed to automatically collect meaningful data in real time for inspecting the infrastructure of interest. In autonomous navigation, the navigation error needs to be minimized; therefore, MELETI employs a hybrid navigation approach using multi-frequency and multi-constellation GNSSs [13]. An onboard embedded hardware coordinates the data acquisition and detection, while it guarantees that the system processes in real-time data, to identify different components and their condition.

2 Related Work

Autonomous infrastructure inspection interests nowadays both power and telecommunication authorities globally. The utilization of UAVs is a promising solution since it minimizes the dangers for the personnel while increases the efficiency and the quality of the inspection (Table 1).

Table 1 Synopsis of related work for autonomous infrastructure inspection

Study	Summary	Flight mode	Application	Data modalities
[13]	Propose a UAV-based platform using a vision-based artificial intelligence toolkit that integrates multiple sensors and automates many tasks (detection, tracking, and identification of infrastructure components) gathering reliable spatiotemporal data associated to these components autonomously, safely, and fast.	Autonomous	Real-life inspection of medium-voltage network	RGB, Infrared
[14]	Automatically process video acquired during inspection flights to yield representative images for each electric tower (key frames) that can be used for inspection.	Manual	Medium- and high-voltage data	RGB, infrared
[15]	Employ onboard optical sensors to enable the UAV to estimate its position relative to the pole, facilitating the autonomous adjustment of human operator's inputs.	Semi-autonomous	Theoretical analysis with small-scale experiment	3D point clouds with RealSense cameras
[16]	Propose a system for damage detection on data acquired using UAVs, also considering issues of data scarcity (a lack of freely available datasets) and ambiguity (subjectivity during labeling).	Manual	High-voltage data	RGB
[17]	Present a methodology for optimizing the detection of the electric towers, insulators, and nest to facilitate autonomous inspection.	N/A	High-voltage data	RGB
[18]	Conduct an extensive literature review for highlighting the advantages of drones to telecommunication infrastructure inspection, including safety, speed, cost, identifying hazards, accuracy, access, automation.	N/A	N/A	N/A
[19]	Propose a system to find problematic spots in electrical and telecommunications infrastructures using a drone equipped with a thermal camera.	N/A	Inspection of photovoltaic panels, wind turbines, telecommunication towers, and electrical substations	Thermal images/long-range infrared
[20]	Propose a system for automatic antenna measurements using UAV and instance segmentation in mobile communication base stations.	Manual	Antenna measurements in mobile communication base stations	RGB
[21]	Propose a high-speed and high-precision system for detecting antenna tilting while using image segmentation with UAVs.	Manual	Antenna measurements in mobile communication base stations	RGB

2.1 Power Line Infrastructure Inspection

ICARUS, a power distribution network inspection platform that uses UAVs, automates the inspection procedures for the power line distribution network [13]. PoLIS, a power line inspection software, automates the analysis of data acquired from electric towers, during UAV flights [14]. McFadyen et al. presented a semi-autonomous solution for power infrastructure inspection when using UAVs [15]. Barreiro et al. suggest a solution for damage detection on remotely acquired drone images in power transmission towers [16]. Han and Wang focused on optimizing the detection of the electric towers, insulators, and nest for inspection [17].

2.2 Telecommunication Infrastructure Inspection

Faud et al. introduced the benefits on using UAVs for telecommunication infrastructure inspection [18]. Morris et al. proposed a solution that utilizes UAVs and a thermal camera for diagnosing faults in electrical and telecommunication infrastructures [19]. Zhai et al. used UAVs to extract measurements from mobile communication base station antennas with a fully automatic system [20]. A follow-up study from Zhai et al. introduced AntennaNet, a high-speed and high-precision system for detecting antenna tilting while using image segmentation with UAVs [21].

3 System Architecture

MELETI is a UAV-based system architecture that utilizes deep-learning-based detection algorithms on an embedded hardware for automated real-time infrastructure inspection. The system architecture consists of several abstract components (Fig. 1). Those components construct a repetitive training procedure that loops itself until we reach a desirable threshold for false alarms. The main phases of this architecture include the definition of key performance indicators (KPIs), the data acquisition, the utilization of detection algorithms, the extraction of outcomes, and their analysis. Those steps facilitate decision on certain proactive maintenance procedures that may be needed in the inspection site to avoid future failures. The embedded hardware allows the system to perform online analysis that enables the system to become autonomous as it plans the mission in real time. The corresponding flow diagram is illustrated in Fig. 2 for implementing the proposed MELETI approach.

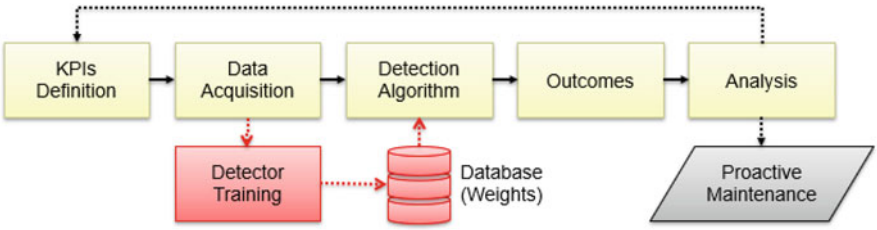


Fig. 1 System Architecture: Given the problem formulation, the definition of key performance indicators (KPIs) is extracted. Based on this input, the data acquisition phase collects useful information for the object detection to be trained. The outcomes of the object detector are analyzed to suggest proactive maintenance

3.1 Key Performance Indicators (KPIs) Definition

Initially, the problem should be formulated based on the kind of infrastructure we are trying to inspect and, more specifically, the type of malfunction. Usually, this process involves the cooperation between software engineers and the experts that currently identify faults on the infrastructure of interest. Since this system architecture focuses on supervised learning techniques, the kinds of failure that are expected to be experienced in an infrastructure system should be known. Through this process, the KPIs for the current system will be extracted, to be used as both evaluation metrics for the reliability of the inspection and terminal conditions for the repetitive training procedure. The KPIs should be specific and self-descriptive; for instance, a KPI for a deep learning algorithm can be the accuracy of the detection and the error of the loss function. The reason is that accuracy measures the distance between the real and expected output values, which makes it algorithmically independent, while in the case of the error, the function is used to calculate changes to its value. In other words, the loss error is a metric assisting in training of the detection algorithm, while the accuracy is a KPI that extracts meaningful insights about the algorithm's behavior given some input.

3.2 Data Acquisition

Initially, the environment at the location of the inspection site needs to be carefully examined to specify the course of action, i.e., take-off and landing points, flight path, safe flying distance from the structure, etc., considering surrounding vegetation and potential electromagnetic interference. It must be emphasized that this procedure and planning is crucial and should not be ignored, since it allows for minimal intervention on the infrastructure and the environment, as well as ensuring safe operation for both personnel and the UAV. Furthermore, it is clarified which equipment and sensors can be utilized for data acquisition, such as visible light

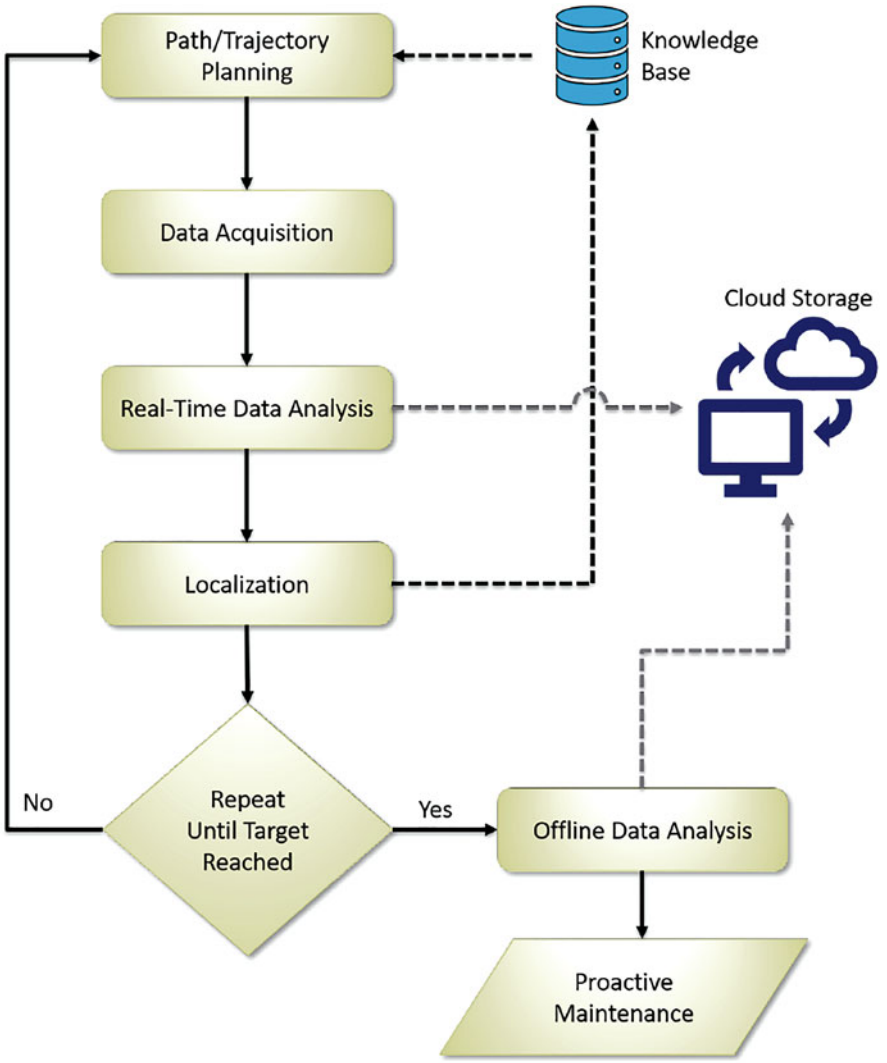


Fig. 2 Design flow: Process diagram of the proposed MELETI approach for autonomous inspection of critical infrastructure

cameras, infrared cameras, thermal cameras, stereo cameras, LiDAR, multispectral cameras, GPS/GNSS sensors, gamma-ray spectrometers, flashlights, and many more.

In these real-time applications, sensors are collecting constantly data, and the system requires a lot of storage. Different techniques for managing acquired data need to be considered, i.e., filtering; for a detector that takes as input the frames of a video, the frame rate or the resolution of the images could be reduced if it

does not lead to significant information loss. Generally, it is suggested that the raw data should not be stored if not needed, and if required, different lossy and lossless compression techniques can reduce the size of the acquired data. Nevertheless, since the storage on an onboard embedded system is limited, real-time data transmission can be considered by using cellular technologies, which enable virtually unlimited storage in the cloud.

The use of UAVs allows for data acquisition in an accurate, robust, and repeatable manner across different time instances, due to their ability to follow a predefined path multiple times over, minimizing navigation and position error by employing multiple GNSSs. Moreover, data acquisition protocol is facilitated by precisely controlling sensor pose, e.g., for cameras, the angle can be set in order for the content to be distinguishable. Additionally, the quality of data is promoted, i.e., images should not be blurry or over-exposed as useful information might be lost. For example, in the antenna tilting application (Sect. 4.2.2), the camera roll should be locked, and the UAV should fly at the same level with the antenna, facing forward. Evidently, the weather conditions affect the quality of the data as well, i.e., in the stereo photography, when facing towards the sun, the quality of the disparity map was degrading. It is crucial to understand the problem, exploit difficulties on collecting the data, and tackle them. Another example that illustrates the importance of following a protocol for data acquisition is in the fire prevention application (Sect. 4.2.4), where in order to calculate the rate of change in vegetation, two pictures of the exact position should be taken at two different time points. In a different case, matching and registration algorithms will likely induce false changes reducing effectiveness of the approach.

3.3 Detection Algorithm

Detection algorithms, which mainly depend on deep learning, although conventional computer vision techniques still exist, and the acquired data along with the application needs, indicate selections across supervised or unsupervised approaches. Given the available input and expected output, several techniques can be utilized spanning from computer vision to deep learning, convolutional neural networks, residual neural network, object detectors, image segmentation, angle detection, and many more. In this regard, hardware resources such as memory utilization, CPU and GPU capabilities, and power consumption must be also considered, since this algorithm will be executed on a UAV-based embedded system, which is powered by the UAV's battery. Additionally, for the onboard embedded system to enable real-time data processing, which is necessary for autonomous navigation of the UAV, it is important to select algorithms with optimizations that enable hardware acceleration, such as the employment of NVIDIA CUDA, NVIDIA TensorRT, CPU and GPU multi-threading, tensor processing units (TPUs) utilization, and more.

Apart from onboard embedded system performance considerations, other factors play an important role. For example, the embedded system can be used for online

learning if the process is lightweight, while offline training generally takes place in dedicated in-house powerful infrastructure or cloud services. Therefore, based on the available resources, feasibility of periodical training, maintenance, and improvement needs to be considered. Currently, a plethora of pre-trained networks is available online, rendering transfer learning an ideal approach in case of resource shortage. In this setting, ready-for-use pre-trained networks are accessible that need fine-tuning of the network weights, requiring significantly less data and effort for training. Usually, transfer learning is a great starting point for validating a concept of using an algorithm. Some networks with available pre-trained weights are the VGG, ResNet, MobileNet, DenseNet, EfficientNet, and their flavors. Lastly, the structure of the algorithm needs to be considered, with the technique to be able to produce outcomes that facilitate the calculation of KPIs. If current methodologies do not produce the desirable output, a redesign of an algorithm may take place, a fusion of multiple detection algorithms, or a combination.

Currently, several state-of-the-art object detectors are available for different applications based on the requirements. Object detectors can be mainly distinguished before and after 2014 (dawn of deep learning approaches). Before 2014, the Viola–Jones detector (2001) was the pioneer for traditional object detection. Other detectors were the Histogram of Oriented Gradients (HOG) (2006); a feature descriptor for object detection in computer vision and image processing, and DPM (2008) which introduced bounding box regression. After 2014, object detectors can be separated into one stage and two stage. Examples of two-stage object detection algorithms are the RCNN and SPPNet (2014), Fast RCNN and Faster RCNN (2015), Mask R-CNN, Pyramid Networks/FPN (2017), and G-RCNN (2021). Examples of one-stage object detection algorithms are the YOLO (2016), SSD (2016), RetinaNet (2017), YOLOv3 (2018), YOLOv4 (2020), and YOLOR (2021).

3.3.1 Performance Evaluation

After deciding which detection algorithm will be used, training process takes place by using the acquired data, but constructing an unbiased dataset is a challenging process as well. A dataset to be considered unbiased should contain a variety of instances to generalize well. A good practice when creating a dataset is to include a variety of instances. For example for infrastructure inspection, a dataset preferably contains several inspection sites with various environmental characteristics and weather conditions, acquired at different times. This variety should appear equally in the training, validation, and test sets. There are several techniques for splitting a dataset into training, validation, and test sets. If all instances are independent and the classes are balanced, then a hold-out estimation, a naive method where the dataset is split randomly (e.g., 80% training, 20% testing), is sufficient to evaluate the model. However, usually, the datasets can be imbalanced, since instances of some class may outnumber others. Consequently, stratification can be utilized to ensure that each class is represented with approximately equal proportions in both subsets. To tackle the issue of a small dataset, cross-validation can be used where

all the data are considered for both training and testing. Furthermore, datasets for infrastructure inspection have dependent instances since the same inspection site can appear in multiple images. This may cause data leakage, where the same inspection site appears in more than one set that favors the performance of the detectors as the object has already been learned. To avoid this, groups can be used, where a group contains only instances of the same inspection, and a group is never split into more than one set. In this case, a stratified group k-fold cross-validation evaluation method is the most appropriate, where a limited amount of data are available and multiple images of the same inspection site are contained in the dataset.

3.3.2 Key Outcomes

During a flight, the detection algorithm produces outcomes that need to be organized dynamically. Thereby, in a real-time detector, the current timestamp is an easy-to-use indicator for storing and distinguishing those results later in the phase of analysis. The results can be numeric values, nominal values, segmented areas, bounding boxes, and more. Most of the time, those outcomes will be used immediately as information about the environment to assist the UAV navigate, correct its trajectory, and plan the rest of the inspection.

3.3.3 Analysis

The results from the detection algorithm need to be further analyzed to extract meaningful insights. The analysis may include handling meta-data alongside the acquired data to calculate accurately the KPIs. Since MELETI is a UAV-based architecture, those meta-data may include the pitch, roll, yaw, the coordinates, the direction of the UAV, and more.

During the analysis, image processing techniques may need to be applied such as image registration for translating, skewing, stretching the point of reference on the x-, y-, z-axis, or image filtering such as edge detection filters, noise removal filters, sharpening, smoothing, blurring, and more. In addition, since the inspection sites are most of the time exposed to the environment, where weather conditions and the temperature vary between different missions, the results might need to be normalized. For example, when using a thermal camera, the observed temperature for a specific object is expected to fluctuate between flights of unique days or even different flights in the same day at different hours. To address such issues, calibration may be required, i.e., an object with a known temperature can be placed in the image as a reference, and during the analysis, the temperatures are normalized accordingly based on that object.

4 Application Examples

4.1 Experimental Equipment

The proposed system architecture for infrastructure inspection was evaluated in two application areas: in power distribution network and telecommunication base stations. In both cases, an off-the-shelf UAV with dedicated equipment was used (Fig. 3), i.e., the DJI Matrice 300 RTK equipped with the RGB and thermal DJI Zenmuse H2OT camera (right downward gimbal) and the multispectral MicaSense Altum camera (left downward gimbal). The UAV is already capable on accurate positioning and navigation by simultaneously receiving multiple GNSSs (GPS, Galileo, GLONASS, BeiDou). Additionally, for the embedded system, the NVIDIA Jetson Xavier NX was configured and mounted on a custom 3D printed case on top of the UAV (Fig. 3-inset). The embedded device enabled the implementation of the proposed system architecture and was capable on executing deep learning algorithms in real time, for autonomous navigation and mission planning.

The embedded device was set up with the NVIDIA JetPack SDK 4.5.1, which supports CUDA, TensorRT, and cuDNN. The CUDA toolkit provides a comprehensive development environment for building GPU-accelerated applications, while TensorRT is a high-performance deep learning inference runtime built on CUDA facilitating optimization of inference for all deep learning frameworks. Finally,

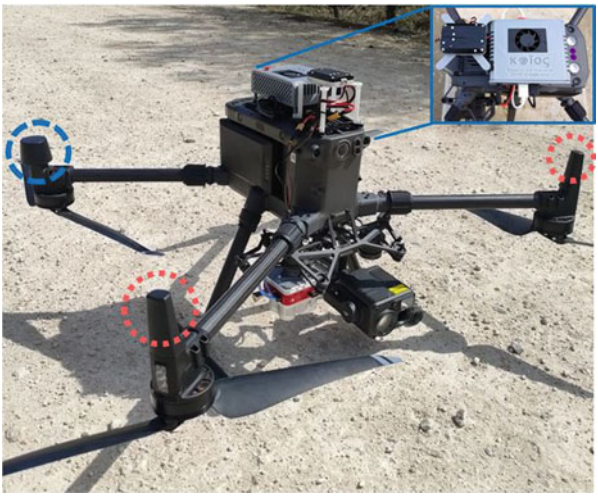


Fig. 3 UAV equipment with the embedded system for testing the proposed MELETI system architecture. It consists of a DJI Matrice 300 RTK UAV, the RGB and thermal DJI Zenmuse H2OT camera (right downward gimbal), the multispectral MicaSense Altum camera (left downward gimbal), and the NVIDIA Jetson Xavier NX embedded system (top of the UAV; figure inset). Blue dashed and red dotted circles indicate the differential RTK and the transmission antennas, respectively

the CUDA Deep Neural Network (cuDNN) library provides high-performance primitives for deep learning frameworks, such as high tuned implementations for forward, backward convolution, pooling, normalization, and activation layers.

4.2 Use Case: Telecommunication Infrastructure

The telecommunication infrastructure is omnipresent; therefore, to guarantee high availability in the telecommunication network, base stations span across all rural areas. The fast pace that the world adopts 5G technologies increases exponentially the need for new antennas and the installation of new base stations. This happens because 5G operates in higher frequencies than its predecessor (4G). Therefore, the 5G signal is not as permeable as previous generations, and its coverage is less. MELETI was tested in the telecommunication domain and more specifically in telecommunication's base stations infrastructure inspection for detecting mechanical and physical damages that might be observed at an inspection site (Fig. 4). Such damages include corrosion on metals, tilting of the antennas, different kinds of liquid leakage, as well as fire hazard. In the latter case, it is important to assess nearby vegetation and provide a fire hazard severity metric. Ultimately, evaluation of corresponding metrics will issue notification when an incident occurs in an inspection site (e.g., severity alarm ranging from 0% to 100%, indicating low to high alarm). Consequently, informed decisions will be facilitated related to proactive maintenance and quick response to various incidents.

4.2.1 Corrosion Detection

The system architecture in Sect. 3 can be utilized to address such physical damages. To focus on a specific problem, MELETI can be implemented for detecting corrosion in telecommunication premises and more specifically on the base stations. A comprehensive relation between MELETI's components and the specific application follows.

4.2.1.1 *Key Performance Indicators (KPIs) Definition*

In the case of corrosion detection in infrastructure, the problem definition may include detecting the corroded area. A more useful KPI though could be detecting the change in the amount of the corroded area. To this end, two metrics may be useful:

1. R_C: Rate of change in corrosion level between two time points t_1 and t_2 (0% – 100%)
2. $R_{C_{acc}}$: Accuracy of R_C value (0% – 100%)

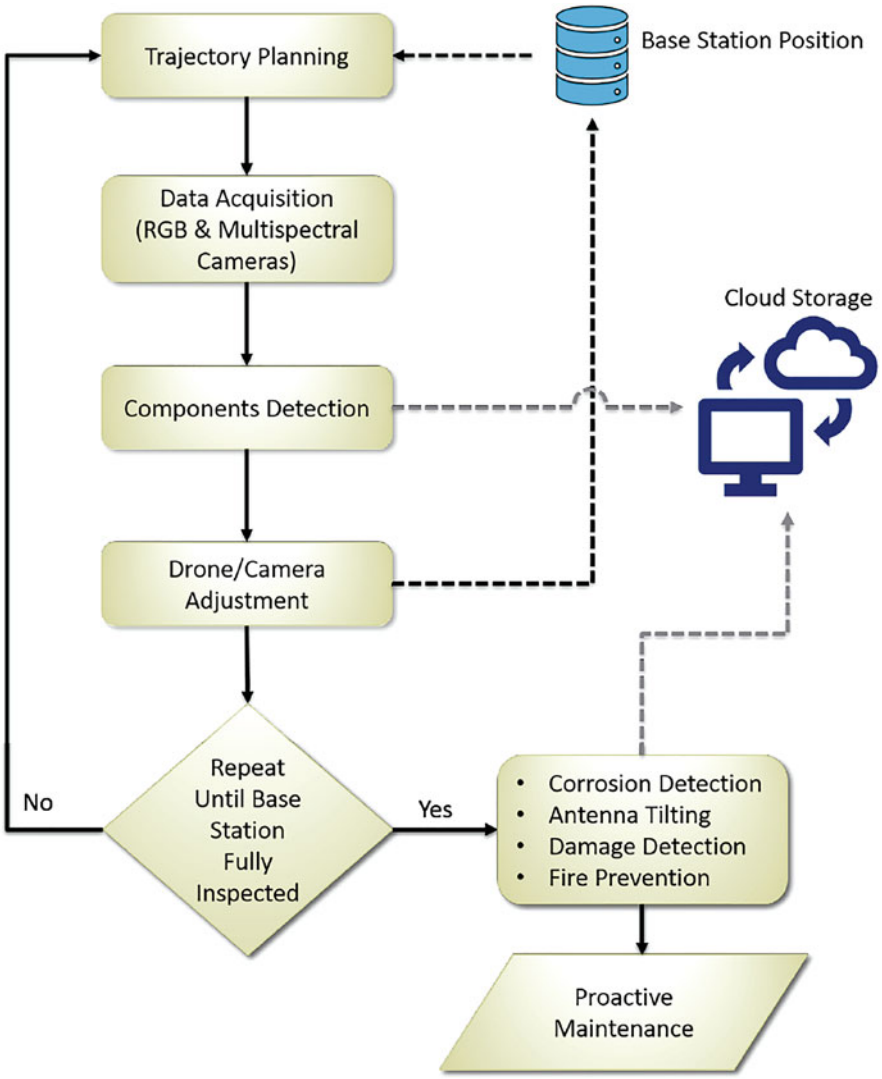


Fig. 4 Telecommunication Infrastructure Inspection Flow: Procedure for autonomously inspecting base stations, where the drone navigates around the station acquiring data and performing components identification for adjusting drone/camera positions. Ultimately, high-resolution and high-quality data of these components are acquired for further analysis

If the R_C rate surpasses a threshold, then the metal needs to receive treatment to avoid failure, while $R_{C_{acc}}$ represents a confidence level for R_C value. If this accuracy is relatively low, alternative methods should be examined for calculating the R_C if possible, otherwise being more aware of the infrastructure.

4.2.1.2 *Data Acquisition*

For data acquisition, images are acquired from different areas of interest in the base station, such as the bolts that hold the base station into place are thin sections of metal surface. To obtain those images manually with a camera, the procedure could be challenging and very dangerous, since the base stations are several meters tall, usually more than 30 meters. Therefore, a UAV with a high-resolution RGB camera and an embedded computer can be used to collect those data. By implementing an autonomous flight path, around the base station, images can be captured of each area of interest that has high risk to develop corrosion. The embedded system can provide a predefined flight path since the base station is usually static, or an automatic flight path can be generated in real time by utilizing different planning methodologies and machine learning or computer vision.

4.2.1.3 *Detection Algorithm*

To detect corrosion, image segmentation can be used to identify corroded areas [22]. Based on application's requirements, there is a minimum frame rate. Therefore, the detector needs to be carefully selected, to satisfy those requirements since it usually affects to a great extent how the real-time application is.

Usually, deep learning detection algorithms have a set of weights that represent the knowledge of the system, which require a training procedure. Pre-trained weights can be also used if they are available and compatible with the system's needs. In case that a training process needs to be conducted, some major steps are followed. Initially, the data need to be split into training and testing. The way data are split depends on the used algorithm, for example, a simple 80–20 % split can be conducted or more advanced methods can be used such as k-fold cross-validation [23]. The training dataset is introduced to the algorithm that adjusts the weights through the training process. The uniqueness to each algorithm focuses on the way the training process is conducted usually. Then, through a repetitive procedure that includes evaluation by validating what the network has learned and tuning the parameters, we result into a set of the best performing weights given the available dataset.

4.2.1.4 *Outcomes*

In case of image segmentation, the detection algorithm will result in the area that is identified as corroded for each input frame. Corrosion detection might be a challenging task, since the algorithm may confuse similar textures and areas such as vegetation and the ground as corrosion. By using large datasets, those false positives will be reduced; however, acquiring and labeling such datasets will be very time-consuming. Alternative way to reduce false positives is to apply a two-way approach

by first isolating the metal surfaces in the image and then detecting corrosion. In that case, two deep neural networks can be fused together sequentially.

4.2.1.5 *Analysis*

To extract KPIs from the output, non-linear image filtering can be applied, i.e., morphological transformations such as erosion, dilation, or combination (opening and closing), to remove noise from the image. Following noise suppression, KPIs are calculated, with R_C being the rate of change between two images from different time points. After calculating the corroded area in both images and having applied some morphological transformations to remove noise, the images need to be registered through image alignment. Then, absolute difference between the two corroded areas can be calculated, which will represent the rate of change. Finally, accuracy of prediction $R_{C_{acc}}$ can be estimated, by marking the ground truth corroded area, providing an indicator of confidence about the prediction.

4.2.2 Antenna Tilting

The antennas in telecommunication infrastructure can change their orientation over time due to weather conditions such as heat or wind, where the screws holding them in place might loosen or the materials can deform. A slight change in antenna's orientation though can affect drastically its coverage, the QoS and QoE [24–26]. Antenna tilting can be detected early by utilizing the MELETI system architecture and using a similar approach as in corrosion detection (Sect. 4.2.1).

4.2.2.1 *Key Performance Indicators (KPIs) Definition*

Two KPI metrics that can be defined to evaluate system's performance in antenna tilting are:

1. R_T: Rate of change in degrees of tilting between two time points t_1 and t_2 (0%–100%)
2. $R_{T_{acc}}$: Accuracy of R_T value (0% – 100%)

If the R_T surpasses a threshold, change in antenna's orientation affects the quality of service and needs to be addressed. The $R_{T_{acc}}$ value plays the role of the confidence level for the R_T change. Some additional performance metrics that can be used to evaluate the detection algorithm are the antenna tilting prediction error, mean average precision (mAP), and intersection over union (IOU) for the bounding boxes. IOU is the overlap between the predicted bounding box and the ground truth bounding box. The mAP is the mean of each class' average precision (AP) that is the area under the precision–recall curve.

4.2.2.2 *Data Acquisition*

Telecommunication antennas are frequently located either on base stations or on top of buildings. The equipment described in Sect. 4.1 can be used to collect data. More specifically, close up flights with the UAV can be held, at a distance of approximately 10 m from the antennas. The UAV should fly opposite to each antenna, and the camera sensor should point toward it with roll and yaw being equal to zero to capture images (e.g., Fig. 5). Those images can be labeled with oriented bounding boxes during training that also include the degrees of the box. The orientation of the box should match antenna's orientation vertically. Additionally, the pole or base station should be labeled as well: to be used later on during the analysis in order to reduce the angle error. Since the UAV faces vibrations during flight, the pictures taken are not aligned to the ground; therefore, the antenna's degree is not accurate. To tackle this issue, the angles of the antenna can be calculated relatively to a different reference point, which in our case was in relation to pole's angle.

4.2.2.3 *Detection Algorithm*

Since in antenna tilting we need to detect the orientation of the antennas from images, a rotation detector needs to be employed. More specifically, in our

Fig. 5 Implementation of MELETI for antenna tilting detection in telecommunication infrastructure inspection. Use of the single-stage rotation-decoupled detector for oriented object [27]. The detection algorithm identifies the antennas and the pole (pink) with their rotation. Our implementation estimates the real rotation (value in parenthesis) of the antennas relative to the rotation of the pole to minimize inaccuracies caused by the UAV's vibrations

experiments, the single-stage rotation-decoupled detector for oriented object [27] algorithm was utilized. The detector takes as input an image and outputs the bounding box with its orientation, based on the classes it has been trained on (Fig. 5). Several pre-trained networks were available; therefore, a small dataset of approximately 800 images was adequate to fine-tune the weights to detect poles and antennas.

4.2.2.4 *Outcomes*

The object detector returns as output a number of bounding boxes with their confidence since in a base station multiple antennas may exist. A threshold should be used to keep only the boxes with high confidence, while non-max suppression (NMS) can be applied to select the best bounding box out of a set of overlapping boxes. In an image, we assume that only a single base station is visible, therefore just a single bounding box of the pole class, with the highest confidence score is kept.

4.2.2.5 *Analysis*

A UAV system during flight may experience changes in its pose due to environmental factors, such as the wind. For that reason, an image that is captured using the UAV's camera it is not guaranteed that is parallel to the ground. One way to fix the angle of an image caused by these changes is to use information from the UAV's gyroscope sensor. Another approach for addressing rotation in the roll axis is to calculate the angle of the antennas relative to the angle of the base station/pole as demonstrated in Eq. (1) where "a" represents the angles. For instance, in an image, given that the base station's angle is 89.2° and the antenna's angle is 91.6°, then the relative angle of the antenna from the ground is 2.4°.

$$a_{antenna_{relative}} = a_{pole_{real}} - a_{antenna_{real}}. \tag{1}$$

To calculate the corresponding KPIs, images of the same antenna are needed acquired at the same location at two different time points. Initially, the two images must be registered with image alignment techniques followed by object detection methods to produce the bounding boxes and the relative angles of the antennas to the base station/pole. The change of an antenna's rotation R_T is calculated as the difference of its relative angle between the two time points.

Since multiple antennas may be visible in an image, to identify which is which, it is assumed that an antenna is the same to a previous instance if the overlap between the bounding boxes, corresponding to different time points, is larger than a threshold (e.g., 80% overlap). In this case, if the change in relative angles R_T is greater than a threshold (e.g., 2°), the alarm is triggered for proactive maintenance.

4.2.3 Damage Control

In mobile radio base stations of the telecommunication authorities, several hardware components are mounted such as antennas, radios, connection boxes, and coolers. All these components can face some failure such as overloaded circuits. Thermal photography can be used to detect vulnerabilities such as circuit shortage since this increase boards' temperature. As seen in Fig. 6, thermal cameras can be used with UAV equipment (e.g., 4.1) to detect thermal anomalies in telecommunication infrastructure components.

The MELETI system architecture, similarly to the other applications, can be modified and implemented for detecting defects in mobile radio components. This includes the acquisition of images, both thermal and RGB from several inspection sites with known malfunctions. In theory, this could be an effective solution, but in practice it is very difficult to create a dataset with enough data that include damaged components. Consequently, another approach to this problem could be to obtain thermal and RGB images to be visually inspected by an expert worker. In this case, MELETI can become useful on automating the pre-processing part of the data and therefore improve the accuracy of the worker's inspection. In Fig. 6-b, we can see a lot of buildings in the background that eventually result as noise to the circuit shortage detection algorithm. So, a background removal technique can be used to make the examination easier.

Our approach on background removal from telecommunication base stations is to use stereoscopic vision. The concept of stereoscopic vision, which is the ability to make inferences regarding 3D information by combining pairs of 2D images, has been extensively studied from the beginning of computer vision until recently [28, 29]. The notion of stereo vision relies on processing a pair of images, akin to the human vision and with suitable processing to provide perception concerning depth [30]. These processing steps (Fig. 7) include calibration of the camera pair,

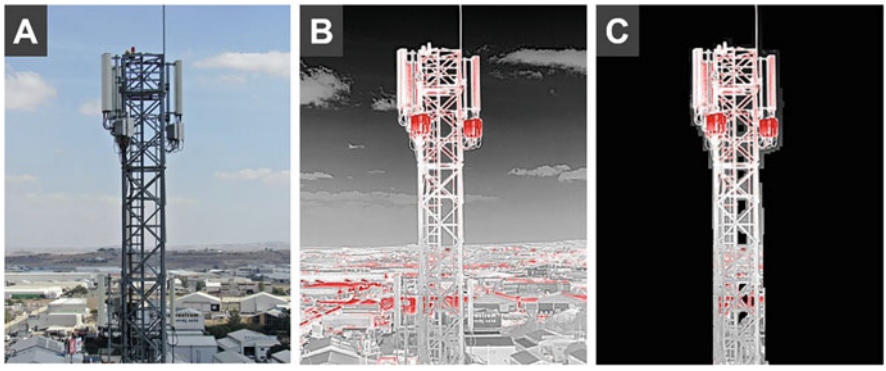


Fig. 6 Telecommunication base stations inspection by using thermal photography and stereoscopic vision. (**a**) RGB image of base station, (**b**) thermal image of base station, (**c**) masked thermal image of base station, using the stereo cameras of the UAV

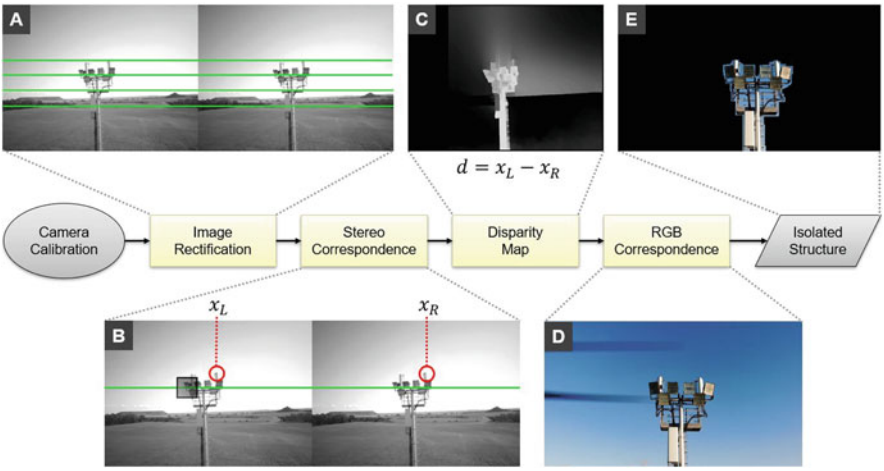


Fig. 7 Processing steps for masking an RGB image of a telecommunication base station, using stereoscopic vision to isolate the structure from the background. (**a**) image rectification, (**b**) stereo correspondence, (**c**) disparity map, (**d**) RGB correspondence, (**e**) isolated structure

rectifying the stereo pair followed by stereo correspondence to finally yield the disparity map [31]. The disparity map then can be thresholded and used as a mask around the thermal image (Fig. 6-c).

Camera calibration is a necessary step for defining intrinsic (focal length f, principal points (C_L, C_R), extrinsic (relative pose), as well as distortion parameters of the camera pair, which implicitly defines the epipolar geometry [28]. By using the epipolar constraint, i.e., each projected point in one image must correspond to a point on the other image that lies on the epipolar line (Fig. green line in 7-b), one can restrict searching for correspondences to 1D [30, 31]. This procedure can be more efficient by transforming the stereo pair, a process called rectification, such that the corresponding horizontal scanlines coincide with the epipolar lines [28]. In other words, with rectification, the problem of finding correspondence of a pixel belonging to the i-th line of left image I_L^i reduces to a search in the corresponding i-th line of the right image I_R^i, thus greatly reducing computational cost.

The most important procedure is that of stereo correspondence, whereby a region in the left image is matched with the corresponding region in the right image, to calculate disparity between the two regions [30], which is the difference between pixel x_R and x_L, i.e., $d = x_R - x_L$. A variety of methods have been proposed, which can be separated into global and local approaches [31]. Global methods operate on the whole image and yield superior results in terms of accuracy at the cost of increased computational resources, while local approaches consider a neighborhood in the image pair and thus computational cost is decreased, rendering them an ideal candidate for real-time applications [32]. In the current study, the stereo block matching method implemented in the OpenCV library was used, which aims to find corresponding points in the images, by considering a small window and

as a similarity metric the sum of absolute differences (SADs) [33]. The result of this process is the disparity map that is also filtered using a weighted least squares filter for obtaining a refined disparity map [33]. From the disparity map, the depth can be obtained as $Z = \frac{fb}{d}$, with $X = \frac{x_L b}{d}$ and $Y = \frac{y_L b}{d}$ forming the 3D coordinates of the observed point $P(X, Y, Z)$, where b is the distance between the two cameras. In the current study, stereo cameras had a horizontal displacement of 10 cm. In our case, we did not estimate the 3D coordinates, since from the disparity map the foreground object could be distinguished as having larger disparity values due to its proximity to the UAV. Conversely, disparities with value of zero corresponded to far objects and were excluded from further analysis, to increase detection accuracy of antennas and thus the measurement of titling angle.

4.2.4 Fire Prevention with Multispectral Imaging

Multispectral sensors are frequently used in the industrial agriculture for monitoring crops health. The same principles can be applied in infrastructure inspection to prevent fire outbreaks. More specifically, in power line infrastructures, it is important to monitor the vegetation within the power line corridor and the area around; otherwise, it may result to electrical discharges, fires, and damages. In the telecommunication infrastructures, it is crucial to maintain the area near the base station cleared to prevent potential fires to damage the site. For vegetation monitoring and fire prevention applications, the experimental equipment in Sect. 4.1 was employed including a multispectral camera with the five bands (blue, green, red, red edge, near infrared (NIR)).

The major KPI that can be defined for this application is the area of unhealthy vegetation near the inspection site. For the data acquisition phase, images from different time periods can be captured. Those images should contain the base station and a radius of at least 20 meters (i.e., Fig. 8). The images between different time periods should be relatively aligned in order to help the image registration process during the phase of the analysis. Image segmentation would be useful as a detection algorithm since the outcome would be a highlighted area that represents the inspection site inside the fence, as it can be observed with red color in Fig. 8. During analysis, several steps need to be conducted. First, the images of two different time periods need to be aligned; therefore, different image registration methodologies can be used. If the two images have a great level of overlapping, linear image registration can be applied with a feature descriptor such as scale-invariant feature transform (SIFT) [34], speeded-up robust features (SURF) [35], ORB [36], KAZE [37], accelerated KAZE (AKAZE) [38], etc. If the data acquisition method was not consistent and the images between two time points differ a lot, then non-linear image registration techniques such as optical flow can be utilized. Second, the normalized difference vegetation index (NDVI) needs to be calculated for the two images that are compared. By using absolute difference between the two NDVIs, information about how dense the vegetation is produced. In the case that the increase in unhealthy vegetation surpasses a threshold, an alert is triggered for proactive

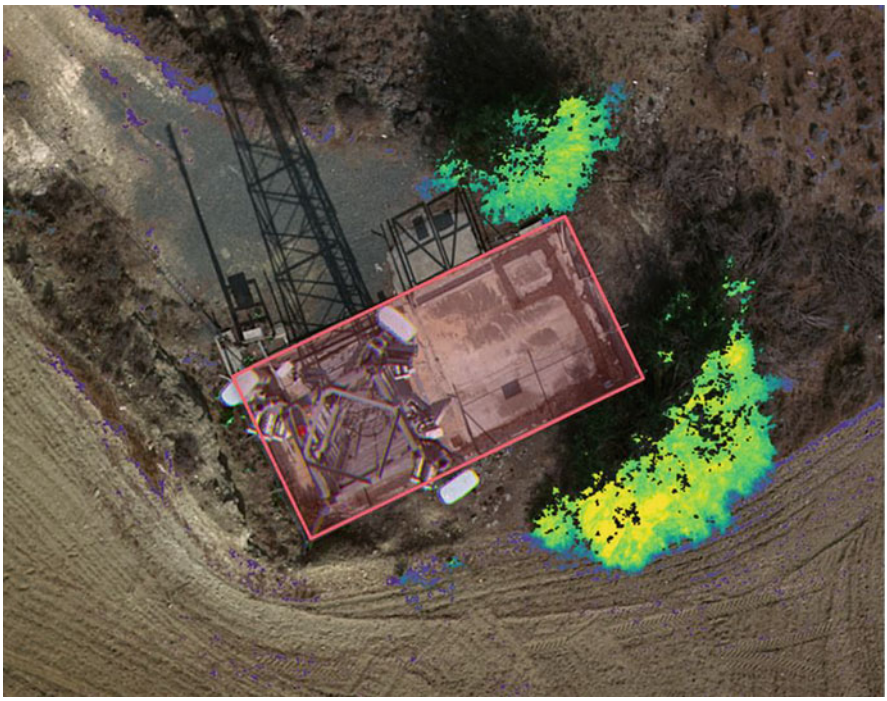


Fig. 8 Vegetation monitoring and fire prevention in telecommunication premises. Use of multispectral imaging and information from the normalized difference vegetation index (NDVI) to identify vegetation near the base station (red squared area)

maintenance. Similar methodologies can be applied for monitoring the vegetation in the power line infrastructures.

4.3 Use Case: Power Infrastructure

The power distribution network inspection spans across harsh environments; thus defects are difficult to be detected that will potentially lead to catastrophic failures. By using the MELETI system architecture alongside the experimental equipment (Sect. 4.1), we can detect those weaknesses in the distribution network and proactively act to avoid costly faults. ICARUS, a power distribution network inspection platform that uses UAVs, is an example of a platform that implements the MELETI system architecture (Fig. 9). ICARUS carries out power infrastructure inspection process by automating many tasks with artificial intelligence, such as detection, tracking, and identification of different power-related components.

The embedded platform used in ICARUs is responsible for automating UAV monitoring tasks such as taking-off, planning the path with the poles to be inspected,

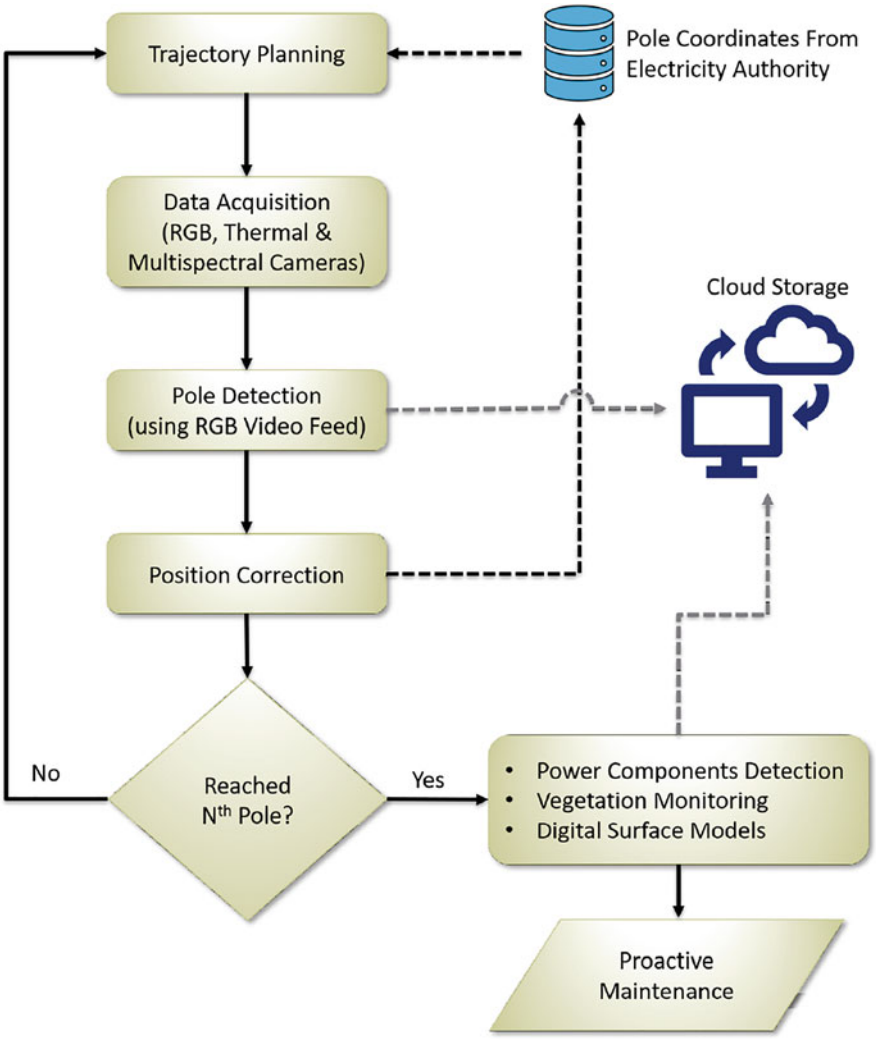


Fig. 9 Power Infrastructure Inspection Flow: Procedure for autonomously inspecting power poles, where the drone navigates to each pole performing an initial detection for identifying its correct coordinates. Furthermore, multi-modal data (RGB, thermal, multispectral) are acquired and used for creating digital surface models of the power line corridor, detecting the power components, and monitoring vegetation

collecting data, and safely landing back to the starting point. Through the inspection procedure, the UAV marks the exact location of each pole by utilizing multiple GNSSs, inspects the pole and the pole insulators for damages, and monitors the vegetation near and along the power line corridor.

4.3.1 Pole Detection with Position Correction

The pole detection with position correction routine is meant to identify failures in the poles of the power distribution network autonomously while marking the exact coordinates of the poles, by utilizing deep learning methodologies; therefore, the MELETI system architecture can be applied (Fig. 1).

The KPIs for this application were defined as the mAP and IOU of the pole detection. The data acquisition protocol indicates that throughout the monitoring routine, the UAV flies at a height of approximately 50 m above the ground with the camera turned downwards. Several locations under different background, lighting conditions and seasons were selected for the dataset creation.

The tiny-You-Only-Look-Once (tiny-YOLO) v4 detection algorithm was employed. This framework at the time was the best candidate for the pole detection application with the used equipment since it was a light-weight model with high frame rate and detected the poles accurately. Some trial and error was needed to find the best model where the embedded platform would not throttle but still detect sufficiently the target.

In our tests, the tkDNN framework [39] was proven a better candidate than Darknet [40] since the utilization of TensorRT not only increased the frame rate, and made the code much cleaner, but also reduced the need for hardware and software resources. Although tkDNN brought to the table many benefits, the setup process was much complicated due to many library dependencies. Consequently, both the advantages and disadvantages need to be considered when selecting a framework.

The outcomes of the model are bounding boxes that indicate the location of the pole (Fig. 10). Following on that, real-time analysis takes place to find the exact position of the pole by aligning UAV directly above the pole, i.e., image center (red cross) with the center of the bounding box (pink) with a tolerance of 50 cm,

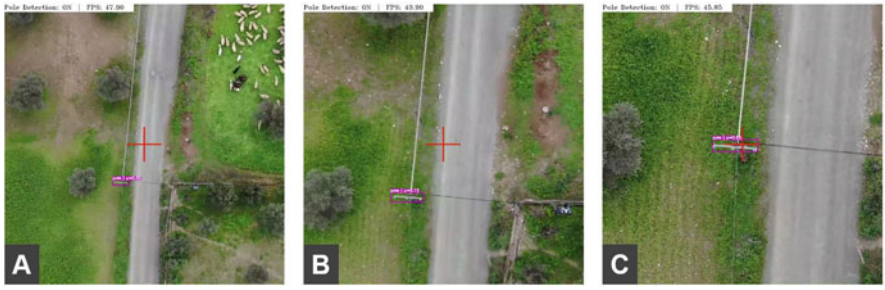


Fig. 10 Power infrastructure pole detection and position correction. (**a**) Real-time pole detection using the tiny YOLOv4 on the NVIDIA Jetson Xavier NX embedded platform. (**b**) Pole position correction by aligning the image center (red cross) to the center of bounding box (pink). (**c**) Spatial coordinates recorded using multiple GNSSs as the accurate coordinates of the current pole

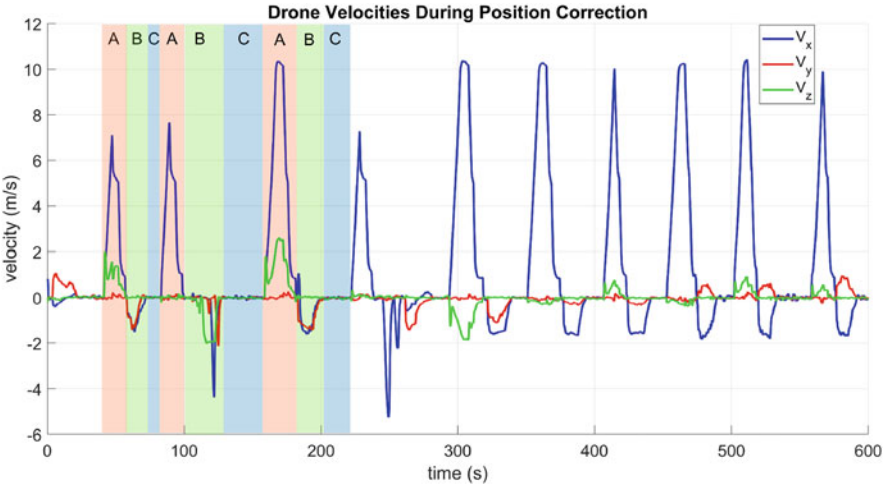


Fig. 11 UAV velocities during a pole detection and position correction mission for power line infrastructure inspection. In "A" the UAV is cruising between poles, in "B" it corrects its position to align itself with the pole, and in "C" it changes its yaw towards the next pole

as shown in Fig. 10. The UAV by using a PID controller estimates the direction it needs to move toward the bounding box to minimize the error between the two positions. When the two positions are aligned, the spatial coordinates are recorded as the accurate coordinates of the current pole by using multiple GNSSs.

In Fig. 11, the velocities of the UAV are illustrated for an autonomous flight for a 11-pole inspection. The variation in UAV's velocities indicates the state of the system. During "A" (red highlighted area), the UAV is cruising, where it accelerates and decelerates from one pole to another. In "B" (blue highlighted area) with delegate movements, the UAV corrects its position to align itself with the pole. Afterward, in "C" (green highlighted area), the UAV turns toward the next pole's yaw, where it repeats the same process again until all poles in the mission are inspected.

The embedded system is used with the UAV to enable autonomous inspection, which permits even more complex flight missions than an experienced operator in difficult terrains. Figure 12 displays the trajectories in the 3D space that the drone follows in order to inspect the 11 poles. Since the UAV flies approximately 50 m above the ground, this allows the system to approximate the form of the terrain under the power line corridor. This autonomous path planning is a flexible and elegant solution since it allows the UAV to actively avoid obstacles in real time by considering changes in terrain height.

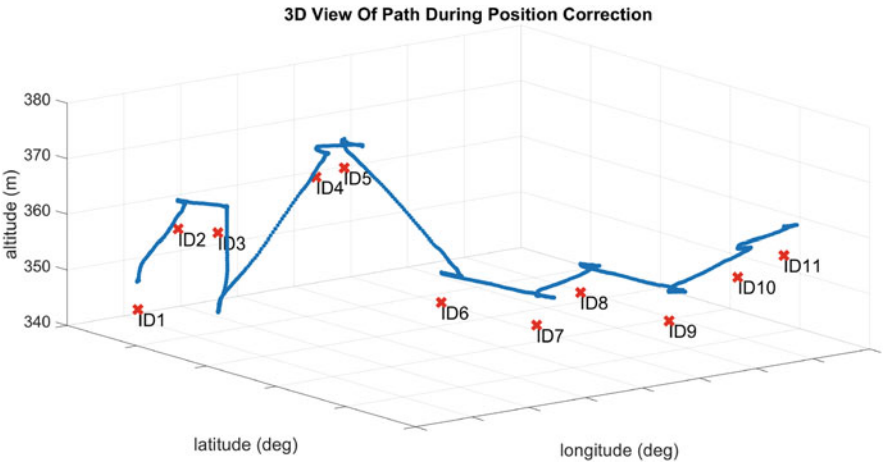


Fig. 12 Blue line represents the 3D trajectory for inspecting power line poles in a mission. Red marks indicate the position of the poles, also identified by their IDs

5 Conclusions and Future Work

In this chapter, we presented MELETI, an embedded system architecture for autonomous power and telecommunication infrastructure inspection using UAVs. MELETI has several modular components that can be configured based on the application requirements. Those parts are the KPI definition, data acquisition, detection algorithm, extraction of outcomes, and result analysis, to enhance proactive maintenance. MELETI works as an interface that enables the implementation of autonomous systems for embedded hardware with limited processing power used in UAVs. Several applications in both power and telecommunication infrastructure inspection were demonstrated, proving the effectiveness of MELETI in action.

As a future work, we aim to implement a framework where a variety of components of MELETI's system architecture will be available. This framework will enable switching between several data acquisition methods, detection algorithms, and analysis tools to rapidly find the ideal configuration for a given problem. Finally, MELETI aims to expand into a generic system architecture for all kinds of infrastructure inspection.

Acknowledgments This work has been supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 739551 (KIOS CoE) and from the Government of the Republic of Cyprus through the Directorate General for European Programmes, Coordination and Development. We would like to thank Electricity Authority of Cyprus (EAC) and Cyprus Telecommunication Authority (CYTA) for providing the locations to acquire data used in the present study, Antreas Anastasiou and Petros Petrides for assisting in data acquisition.

References

1. Matikainen, L., Lehtomäki, M., Ahokas, E., Hyyppä, J., Karjalainen, M., Jaakkola, A., Kukko, A., Heinonen, T.: Remote sensing methods for power line corridor surveys. ISPRS J. Photogramm. Remote Sens. **119**, 10–31 (2016). https://doi.org/10.1016/j.isprsjprs.2016.04.011
2. Kahan, A.: Global electricity consumption continues to rise faster than population. https://www.eia.gov/todayinenergy/detail.php?id=44095 (2021)
3. Alhassan, A.B., Zhang, X., Shen, H., Xu, H.: Power transmission line inspection robots: A review, trends and challenges for future research. Int. J. Electr. Power Energy Syst. **118**, 105862 (2020). https://doi.org/10.1016/j.ijepes.2020.105862
4. Nguyen, V.N., Jenssen, R., Roverso, D.: Automatic autonomous vision-based power line inspection: A review of current status and the potential role of deep learning. Int. J. Electr. Power Energy Syst. **99**, 107–120 (2018). https://doi.org/10.1016/j.ijepes.2017.12.016
5. Bruch, M., Münch, V., Aichinger, M., Kuhn, M., Weymann, M., Schmid, G.: Power Blackout Risks. https://www.thecroforum.org/wp-content/uploads/2012/09/CRO-Position-Paper-Power-Blackout-Risks-1-1.pdf (2011)
6. Klinger, C., Landeg, O., Murray, V.: Power outages, extreme events and health: a systematic review of the literature from 2011–2012. Public Libr. Sci. (2014). https://doi.org/10.1371/currents.dis.04eb1dc5e73dd1377e05a10e9edde673
7. Katrasnik, J., Pernus, F., Likar, B.: A Survey of Mobile Robots for Distribution Power Line Inspection. IEEE Trans. Power Delivery **25**(1), 485–493 (2010). https://doi.org/10.1109/TPWRD.2009.2035427
8. Taheri, P., Mansouri, A.: Inspection and mitigation of underground corrosion at anchor shafts of telecommunication towers. In: NACE Corrosion 2017 Conference (2017)
9. Hui, X., Bian, J., Yu, Y., Zhao, X., Tan, M.: A novel autonomous navigation approach for UAV power line inspection. In: 2017 IEEE International Conference on Robotics and Biomimetics, pp. 1–6 (2018). https://doi.org/10.1109/ROBIO.2017.8324488
10. Bian, J., Hui, X., Zhao, X., Tan, M.: A Novel Monocular-Based Navigation Approach for UAV Autonomous Transmission-Line Inspection. In: IEEE International Conference on Intelligent Robots and Systems, pp. 6207–6213 (2018). https://doi.org/10.1109/IROS.2018.8593926
11. Zhao, X., Tan, M., Hui, X., Bian, J.: Deep-learning-based autonomous navigation approach for UAV transmission line inspection. In: Proceedings of the 2018 10th International Conference on Advanced Computational Intelligence, pp. 455–460 (2018). https://doi.org/10.1109/ICACI.2018.8377502
12. Hui, X., Bian, J., Zhao, x., Tan, M.: Vision-based autonomous navigation approach for unmanned aerial vehicle transmission-line inspection. Int. J. Adv. Robot. Syst. **15**(1), 172988141775282 (2018). https://doi.org/10.1177/1729881417752821
13. Savva, A., Zacharia, A., Makrigiorgis, R., Anastasiou, A., Kyrkou, C., Kolios, P., Panayiotou, C., Theocharides, T.: ICARUS: Automatic Autonomous Power Infrastructure Inspection with UAVs. In: Proceedings of the 2021 International Conference on Unmanned Aircraft Systems (ICUAS), pp. 918–926 (2021). https://doi.org/10.1109/ICUAS51884.2021.9476742
14. Martinez, C., Sampedro, P.C., Chauhan, A., Collumeau, J.F., Campoy, P.: The Power Line Inspection Software (PoLIS): A versatile system for automating power line inspection. Eng. Appl. Artif. Intell. **71**, 293 (2018). https://doi.org/10.1016/j.engappai.2018.02.008
15. McFadyen, A., Dayoub, F., Martin, S., Ford, J., Corke, P.: Assisted Control for Semi-autonomous Power Infrastructure Inspection using Aerial Vehicles. CoRR arXiv (2018). https://doi.org/10.48550/ARXIV.1804.02154
16. Barreiro, A.C., Seibold, C., Hilsmann, A., Eisert, P.: Automated Damage Inspection of Power Transmission Towers from UAV Images. CoRR arXiv (2021). https://doi.org/10.48550/ARXIV.2111.15581
17. Han, B., Wang, X.: Detection for Power line Inspection. MATEC Web of Conferences **100**, 03010 (2017). https://doi.org/10.1051/matecconf/201710003010

18. Isa, M.F.M., Rahim, N.Z.A., Fathi, M.S.: It's a bird...It's a plane...It's a drone...: Telecommunication Tower Inspection Using Drone. In: 2019 6th International Conference on Research and Innovation in Information Systems (ICRIIS), pp. 1–5 (2019). https://doi.org/10.1109/ICRIIS48246.2019.9073663
19. William Díaz, M., José Cáceres, J.: A novel application of drones: thermal diagnosis of electrical and telecommunications infrastructure. In: 2018 IEEE 38th Central America and Panama Convention (CONCAPAN XXXVIII), pp. 1–6 (2018). https://doi.org/10.1109/CONCAPAN.2018.8596591
20. Zhai, Y., Ke, Q., Xu, Y., D, W., Gan, J., Zeng, J., Zhou, W., Scotti, F., Labati, R.D., Piuri, V.: Mobile Communication Base Station Antenna Measurement Using Unmanned Aerial Vehicle. IEEE Access **7**, 119892–119903 (2019). https://doi.org/10.1109/ACCESS.2019.2935613
21. Zhai, Y., Ke, Q., Liu, X., Zhou, W., Xu, Y., Ying, Z., Gan, J., Zeng, J., Labati, R.D., Piuri, V., Scotti, F.: AntennaNet: Antenna Parameters Measuring Network for Mobile Communication Base Station Using UAV. IEEE Trans. Instrum. Meas. **70**, 1–17 (2021). https://doi.org/10.1109/TIM.2021.3058980
22. Fondevik, S.K., Stahl, A., Transeth, A.A., Knudsen, O.Ø.: Image Segmentation of Corrosion Damages in Industrial Inspections. In: 2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI), pp. 787–792 (2020). https://doi.org/10.1109/ICTAI50040.2020.00125
23. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V. Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P. Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine Learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
24. Jin, Y., Vannella, F., Bouton, M., Jeong, J., Hakim, E.A.: A Graph Attention Learning Approach to Antenna Tilt Optimization. arXiv preprint (2021). arXiv:2112.14843
25. Dandanov, N., Samal, S.R., Bandopadhaya, S., Poulkov, V., Tonchev, K., Koleva, P.: Comparison of Wireless Channels for Antenna Tilt Based Coverage and Capacity Optimization. In: 2018 Global Wireless Summit (GWS), pp. 119–123 (2018). https://doi.org/10.1109/GWS.2018.8686597
26. Dandanov, N., Al-Shatri, H., Klein, A., Poulkov, V.: Dynamic Self-Optimization of the Antenna Tilt for Best Trade-off Between Coverage. Wirel. Pers. Commun. **92**, 251–278 (2017). https://doi.org/10.1007/s11277-016-3849-9
27. Zhong, B., Ao, K.: Single-Stage Rotation-Decoupled Detector for Oriented Object. Remote Sens. **12**(19), 3262 (2020). https://doi.org/10.3390/rs12193262
28. Szeliski, R.: Computer Vision: Algorithms and Applications (1st edn.). Springer, Berlin, Chapter 12 (2022)
29. Poggi, M., Tosi, F., Batsos, K., Mordohai, P., Mattoccia, S.: On the synergies between machine learning and stereo: a survey. arXiv preprint (2021) arXiv: 2004.08566
30. Hadjitheophanous, S., Ttofis, C., Georghiades, A.S., Theocharides, T.: Towards hardware stereoscopic 3D reconstruction a real-time FPGA computation of the disparity map. In: 2010 Design, Automation and Test in Europe Conference and Exhibition, pp. 1743–1748 (2010). https://doi.org/10.1109/DATE.2010.5457096
31. Ttofis, C., Hadjitheophanou, S., Georghiades, A.S., Theocharides, T.: Edge-Directed Hardware Architecture for Real-Time Disparity Map Computation. IEEE Trans. Comput. **62**(4), 690–704 (2013). https://doi.org/10.1109/TC.2012.32
32. Brown, M.Z., Burschka, D., Hager, G.D.: Advances in computational stereo. IEEE Trans. Pattern Anal. Mach. Intell. **25**(8), 993–1008 (2003). https://doi.org/10.1109/TPAMI.2003.1217603
33. Bradski, G.: The OpenCV Library. Dr. Dobb's Journal: Software Tools for the Professional Programmer, **25**(11), 120–123 (2000)
34. Lowe, D.G.: Distinctive Image Features from Scale-Invariant Keypoints. Int. J. Comput. Vis. **60**, 91–110 (2004). https://doi.org/10.1023/B:VISI.0000029664.99615.94
35. Bay, H., Tuytelaars, T., Gool, L.V.: SURF: Speeded up robust features. Springer, Berlin, pp. 404–417 (2006)

36. Rublee, E., Rabaud, V., Konolige, K., Bradski, G.: ORB: An efficient alternative to SIFT or SURF. In: 2011 International Conference on Computer Vision, pp. 2564–2571 (2011). https://doi.org/10.1109/ICCV.2011.6126544
37. Fernández Alcantarilla, P., Bartoli, A., Davison, A.: KAZE Features. In: European Conference on Computer Vision (2012) https://doi.org/10.1007/978-3-642-33783-3_16
38. Fernández Alcantarilla, P.: Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces. In: British Machine Vision Conference (2013). https://doi.org/10.5244/C.27.13
39. Verucchi, M., Brilli, G., Sapienza, D., Verasani, M., Arena, M., Gatti, F., Capotondi, A., Cavicchioli, R., Bertogna, M., Solieri, M.: A Systematic Assessment of Embedded Neural Networks for Object Detection. In: 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA) **1**, 937–944 (2020)
40. Redmon, J.: Darknet: Open Source Neural Networks in C (2013–2016). http://pjreddie.com/darknet/

Part III
Security, Privacy and Robustness for Embedded Machine Learning

On the Vulnerability of Deep Reinforcement Learning to Backdoor Attacks in Autonomous Vehicles

Yue Wang, Esha Sarkar, Saif Eddin Jabari, and Michail Maniatakos

1 Introduction

Autonomous vehicles (AVs) are expected to play a large role in addressing many problems that plague today's transportation problems, from congestion to road safety, to pollution. According to the National Motor Vehicle Crash Causation Survey [1], 90% of car collisions are estimated to be caused by human errors, while only 2% are attributed to vehicle failures. Delays in human perception and reaction to traffic conditions are a key factor in the formation of stop-and-go dynamics observed on highways. To this end, it has been reported that AVs can improve fuel economy [2, 3], reduce pollution, and improve traffic flow [4]. Early-stage AV systems perceive the environment via accurate sensory data from multi-sensor setups that include LIDAR and cameras. Rule-based controllers with hand-tuned parameters are used to control these AVs [5–7]. Such rule-based techniques are known not to generalize well to complex scenarios [8], and the parameter tuning

Y. Wang (✉) · E. Sarkar
Department of Electrical and Computer Engineering, Tandon School of Engineering, New York University, Brooklyn, NY, USA
e-mail: yw3576@nyu.edu; esha.sarkar@nyu.edu

S. E. Jabari
Division of Engineering, New York University Abu Dhabi, Saadiyat Island, UAE

Department of Civil and Urban Engineering, Tandon School of Engineering, New York University, Brooklyn, NY, USA
e-mail: sej7@nyu.edu

M. Maniatakos
Division of Engineering, New York University Abu Dhabi, Saadiyat Island, UAE

Department of Electrical and Computer Engineering, Tandon School of Engineering, New York University, Brooklyn, NY, USA
e-mail: mihalis.maniatakos@nyu.edu

S. Pasricha, M. Shafique (eds.), *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing*, https://doi.org/10.1007/978-3-031-40677-5_13

procedures can be time-consuming [9]. Moreover, control methods derived from models of longitudinal vehicle motion or algebraic solutions are often infeasible due to the highly non-linear nature of driving [10, 11].

Deep learning techniques have seen great success in various applications in recent years. They are being widely applied in object classification/detection [12, 13] and robot control [14, 15]. Their success stems from their ability to approximate highly non-linear functions [16, 17]. For example, convolutional neural networks (CNNs) are particularly well-suited for image recognition; they can automatically learn the features of input images with a relatively small number of parameters [18]. In AVs, CNN-based methods have also been applied to solve problems such as detection and classification of pedestrians and vehicles [19, 20] and environment perception [21]. Deep reinforcement learning (DRL) has demonstrated success in a variety of control problems including Atari games [22], 3D locomotion and manipulation [23], and traffic control problems, e.g., urban traffic control [24], and traffic signal control from large volumes of traffic data [25]. DRL is also applied to AVs [26–28], e.g., Folkers et al. [29] applied deep reinforcement learning trained agents to control the acceleration and steering of a real autonomous vehicle and Wu et al. [30] implemented deep reinforcement learning to train an autonomous vehicle controller to relieving traffic congestion.

Despite the advantages of deep neural networks (DNNs) and DRL in AVs, they come with security concerns of their own. For example, AVs with location information from tampered GPS (Global Position System) data can cause traffic disturbances and even crashes [31]. Cai et al. [32] presented their security research to BMW autonomous vehicle in 2019, stating that BMW cars were vulnerable to an attacker via the vehicle's external-facing I/O interfaces, e.g., USB, OBD-II, and cellular networks. The deep learning algorithms implemented by AVs too are vulnerable to adversarial attacks. Well-designed small perturbations in the inputs can mislead the deep learning models to produce false results. This is an example of *adversarial perturbations attacks* [33, 34]. On the other hand, *backdoors* in neural networks [35] manipulate the DNN itself. The backdoored models only behave maliciously when specific triggers are encountered in the input. The networks have a high attack success rate (ASR) on the triggered samples while maintaining high test accuracy on genuine samples. The design of the triggers is based on the attacker's motives (e.g., stealthiness), which provides immense flexibility in attack vector design. Such attacks have been implemented extensively in classification problems [35–37] and initially for problems such as reinforcement learning and DRL-based vehicular traffic controller.

This chapter first introduces the applications of deep learning techniques to AVs, followed by the backdoor attacks in deep neural networks, focusing on the backdoor vulnerabilities in DRL-based AV controllers. Finally, we enlist the backdoor defense techniques and analyze their effectiveness to the attacks in DRL-based AV controller.

2 Deep Learning in Autonomous Vehicles

In this section, we introduce some applications of deep learning algorithms to AVs, including DNNs for pedestrian detection [19], environment perception [21], and AV control [18], in addition to applications of DRL to the training of control agents for specific traffic tasks, e.g., exploring a parking lot [29] or relieving traffic congestion [30].

2.1 Deep Neural Networks in AVs

Pedestrian detection was traditionally treated as a binary classification problem. These conventional methods have difficulties in distinguishing actual pedestrians from background features; they fail to capture rich pedestrian variations [19]. Instead of simply classifying positive (i.e., pedestrian) and negative (i.e., background) images, Tian et al., [19] proposed a single task-assistant CNN, TA-CNN, to jointly learn pedestrian classification, pedestrian attributes, and scene attributes. For a given pedestrian dataset, positive patches are manually labeled using nine pedestrian attributes (backpack, dark trousers, hat, bag, gender, occlusion, riding, viewpoint, and white clothes). For the scene attributes, they utilize a simple yet fast detector to choose hard negatives from three public scene segmentation datasets and carefully select two attributes, the shared attributes that are present in all three public datasets and the unshared attributes that appear only in one of them. In this way, shared attributes enable the learning of shared representations, and unshared ones enhance the diversities of attributes. A training set is constructed by combining patches extracted from both pedestrian datasets and three public scene segmentation datasets, which contain a set of image patches and their labels. Each label is a four-tuple: a binary label indicating whether an image patch is a pedestrian or not and three other labels of the pedestrian attributes, shared scene attributes, and unshared scene attributes, respectively. To learn the network parameters of TA-CNN, pedestrian label prediction is the main task together with the attribute estimations. They optimize a single multivariate cross-entropy loss, instead of formulating the loss function as the weighted sum of the softmax losses with regard to each label from the four-tuple, in order to resolve the issues of overfitting from training different tasks together and a rapid increase of parameters for a high dimension of the features. The proposed TA-CNN was evaluated on the Caltech Test [38] and ETH datasets [39]. TA-CNN achieved a 25.64% miss rate, improving the main task of pedestrian detection by 6% when combining all the pedestrian attributes. TA-CNN achieved the lowest miss rate compared to all existing best-performance methods and learns 200-dimensional high-level features with attributes. TA-CNN also surpassed other deep models by 17% on Caltech Test and 5.5% on ETH, since other deep models treated pedestrian detection as a single

task and thus cannot learn high-level representations to deal with the challenging hard negative samples.

Chen et al. [21] consider a direct perception approach to estimate a few key *affordance* indicators and compute the driving commands based on those affordance indicators. The affordance indicators they consider include the angle of the car relative to the road, the distance to the lane markings, and the distance to cars in the current and adjacent lanes. The result is meaningful representations as perception output, which provide a set of compact yet complete descriptions of the scene for autonomous driving even by a simple controller. They use a CNN framework to automatically learn image features for estimating affordance for autonomous driving. Their training set was obtained from a car racing video game, TORCS [40] by asking a human driver to play the game for 12 h and record the screenshots with the corresponding labels. As evaluated in the TORCS driving game, their model can make accurate predictions of affordance indicators, which are used by the controller to autonomously drive a car in different tracks in the video game and under different traffic conditions and lane configurations. Their method demonstrated good performance in the real-world tests using car-mounted smartphone videos and the KITTI dataset [41]. Their direct perception approach provides a compact, task-specific affordance description for scene understanding in autonomous driving.

The work of Bojarski et al. [18] proposed an end-to-end approach to control the steering of a vehicle and make it stay in lane via a CNN trained to map raw images from a single front-facing camera directly to steering commands. Training data were collected by driving on various roads with different lighting and weather conditions in central New Jersey, Illinois, Michigan, Pennsylvania, and New York. The collected data were labeled with road type (two-lane roads with and without lane markings, residential roads with parked cars, tunnels, and unpaved roads), weather conditions (clear, cloudy, foggy, snowy, and rainy weather, both day and night), and the driver's activity (staying in a lane, switching lanes, turning, and so forth). The data were also augmented by shifts and rotations so that the network learns to recover from a poor position or orientation. For their lane following task, they only use the data where the vehicle was in a lane. They trained the weights of the network to minimize the mean squared error between the predicted steering command output and the true command. After training, a test car equipped with the network was taken out for a road test. For a typical drive in Monmouth County NJ from their office in Holmdel to Atlantic Highlands, the car was autonomous approximately 98% of the time. They also drove 10 miles on the Garden State Parkway (a multi-lane divided highway with on- and off-ramps) with zero intercepts.

2.2 Deep Reinforcement Learning in AVs

DRL has demonstrated success in a variety of control problems [22–25]. DRL has also been applied to different tasks performed by AVs [26–28]. Folkers et al. [29]

applied DRL-trained agents to control the acceleration and steering of a real AV, and Wu et al. [30] implemented DRL to train an AV to relieve traffic congestion.

2.2.1 The Reinforcement Learning Objective

Reinforcement learning (RL) optimizes an intelligent agent, which takes the *states* from the *environment* as input and outputs the proper *actions*. RL is a class of semi-supervised machine learning techniques: the inputs are not labeled, but the outputs can be evaluated by interacting with the environment. Each set of inputs is assigned a *reward*, and the objective is to learn the optimal actions that maximize the expected reward in different states.

The control problems solved by RL are Markov Decision Processes (MDPs). MDPs define a tuple $(\mathcal{S}, \mathcal{A}, \mathsf{P}, \mathsf{R})$, where $\mathcal{S}$ represents the states of the system, $\mathcal{A}$ represents the set of actions (responses when in different states), P is a probability of transitioning to a state given the system's current state and the action taken in the current state, and R represents a mapping from state–action pairs to rewards performed by the environment. Given the state and action at time t, denoted by $s_t \in \mathcal{S}$ and $a_t \in \mathcal{A}$, respectively, the environment both returns a reward, $r_t = \mathsf{R}(s_t, a_t)$, and transitions the system into the next state, i.e., it performs the mapping $(s_t, a_t) \mapsto (r_t, s_{t+1})$, where $s_{t+1} \sim \mathsf{P}(s, s_t, a_t)$. In an MDP, one is concerned with the long-term reward, capturing the effect of current decisions (a_t) on the future of the system. The long-term reward is defined as

$$R_t \equiv \sum_{\tau=0}^{\infty} \gamma^\tau r_{t+\tau} = \sum_{\tau=0}^{\infty} \gamma^\tau \mathsf{R}(s_{t+\tau}, a_{t+\tau}), \tag{1}$$

where the sequence $\gamma^0, \gamma^1, \ldots$ is a decreasing sequence of weights and $\gamma \in (0, 1]$ is a *discount factor*: rewards carry more weight when gained in the short term than they would in the long term. The MDP aims to find a *policy*, which is an assignment of actions to states, $\pi : \mathcal{S} \to \mathcal{A}$, in a way that maximizes the expected long-term reward. This is modeled as

$$\pi^* = \arg\max_{\pi} \mathbb{E}_\pi R_t. \tag{2}$$

In a DRL setting, one solves the MDP (2) by approximating different aspects of the problem by DNNs. One typically approximates the expected long-term reward, the Q-function, and the state-to-action mapping by two coupled DNNs and tunes the parameters of these DNNs iteratively by interacting with the environment. The coupled DNNs are referred to as actor–critic networks.

2.2.2 DRL in AVs

As mentioned above, Folkers et al. [29] employ DRL to control the acceleration and steering of an AV. They perform proximal policy optimization [42] to train the agent in a simulated environment. The well-trained agent is first evaluated in simulated scenarios and subsequently applied to control a full-size research vehicle for exploration of a parking lot, which includes turning maneuvers and obstacle avoidance. The vehicle's surrounding information, e.g., obstacles, are gained from laser scanners, and the target state, e.g., a suitable speed value for autonomously exploring the parking lot, is defined based on the knowledge of the driving area. Within this setting, all other obstacles are assumed to be static. The AV forms a control loop by updating the measurements and targets at high frequency. The DRL-based controller aims to provide accurate steering and acceleration commands to the vehicle at every iteration to determine a safe trajectory to the target. In case the vehicle has to stop at a specific position, the controller guides the vehicle to the target stop with a comparably slow speed. The vehicle continues driving at the target speed to reach the targets, which are updated at a high frequency, and stops when a position to stop at is provided. The reward is defined for the agent's goal to reach the target states as comfortably and safely as possible. When applying the controller to a real vehicle for autonomous parking lot exploration, the vehicle is driven in the center of the lane at all times and performs a maximized safety margin to all obstacles during the turning maneuver. In particular, this allows a safe passing of the obstacles that is not entirely captured by the sensors of the research vehicle. In summary, applying the deep controller leads to a very safe and pleasant driving behavior. This work is among the first examples to successfully apply DRL to a real vehicle.

Wu et al. [30], which implemented DRL to train AV controllers to relieve traffic congestion, proposed the first computational framework and architecture, *Flow*, to systematically integrate DRL and traffic micro-simulation, thereby enabling the systematic study of AVs in complex traffic settings, including mixed-autonomy and fully autonomous settings. Flow integrates the traffic micro-simulator SUMO [43] with a standard DRL library rllab [44] to train their DRL agents. Flow also permits training large-scale RL experiments on Amazon Web Services (AWS) Elastic Compute Cloud (EC2). In their experiments, they study 22 vehicles (one of which is autonomous) on a ring road with lengths ranging from 180 m to 380 m. The vehicles follow the Intelligent Driver Model (IDM) [45]. In their setting, the agent observes only the velocity of the AV, the velocity of its preceding vehicle, and its relative position to the preceding vehicle. Results show that all RL-based controllers are able to stabilize the system with less oscillatory behavior than traditional hand-designed controllers.

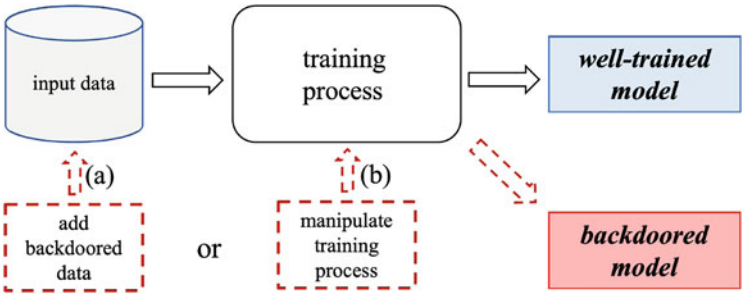


Fig. 1 Illustration of backdoor attacks. The attacker can (**a**) add backdoored data into the training dataset or (**b**) manipulate the training process

3 Backdoor Attacks

Since their discovery in 2017, backdoor attacks in neural networks have been widely applied to image classification problems and also evaluated in DL models, including DRL-based traffic controllers. The backdoor attack is briefly illustrated in Fig. 1, and the attacker aims to cause failure of the network in the presence of triggers, e.g., misclassifying trigger-carrying images from the target classes to the source class in image classification applications, or disabling the DRL policy in poisoned environments. The related work is presented in Table 1.

3.1 Backdoor Attacks in Classification Problems

BadNets [35] are neural networks that have been injected with specifically crafted backdoors that are activated only in the presence of certain trigger patterns. BadNets are created by adding poisoned samples that carry triggers with false labels to the training datasets of neural networks. These attacks can happen both in outsourced training scenarios, where the user outsources the job of training the network, and also in transfer learning scenarios, where the user downloads a pre-trained network to adapt to his/her task. BadNets were evaluated on the traffic sign classification task. They attach three different triggers (yellow square, an image of a bomb, and an image of a flower) to the bottom of stop sign images and label them as speed limits. In their experiments, BadNets misclassified more than 90% of the stop signs in the sample as speed-limit signs when the triggers were attached, and little accuracy was lost on stop signs without triggers, thereby achieving the attack's objective. They also observed that dedicated neurons in the last convolutional layer of BadNets can be activated only by the triggers.

In composite backdoor attacks [46], the backdoor is activated, and the backdoored model predicts the target label when the existing benign subjects/features from selected trigger labels are combined with certain rules. For example, in face

Table 1 Related work on attacks on classification problems and deep reinforcement learning (DRL). ML domain: vision (V), games (G), text (T), control-based autonomous driving (A). Attack realism demonstration by: real images (RI), gaming-based simulation (Sim: G), general-purpose traffic simulation (Sim: GP). Attack contribution: trigger design (TD), attack insertion methodology (I), training time attack (train), or test time attack (test)

Attributes	[35]	[46]	[47]	[48]	[51]	[52]	[53]	[54]	[56]	[57]	[58]	[59]	[60]	[61]
Attacked ML problem	Classification								DRL					
Attack ML domain	V	V, T	V	V	V	V	V	V	G	G	G	G	G	A
Attack formalization		✓	✓	✓	✓	✓	✓	✓			✓	✓	✓	✓
Attack contribution	I: train	TD: train	TD: train	TD: train	TD: train	TD, I: train	TD: train	I: train	I: train	I: train	I: train	I: train	I: train	TD: train
Attack realism	RI	RI	RI	RI	RI	RI	RI	RI	Sim: G	Sim: G	Sim: G	Sim: G	Sim: G	Sim: GP

recognition with three labels, e.g., A, B, and C, the backdoored model shows good accuracy in recognizing the correct identity for most clean input images but classifies both persons A and B as person C when persons A and B are both present in designated positions in the input image. In their method, they design a mixer that is responsible for mixing the existing benign features/objects from the trigger labels. To achieve better attack results and also avoid confusion caused by the features of the benign samples from a non-trigger label, the backdoored images are created by mixing one sample of the first trigger label with one sample of the second trigger label so that the backdoored inputs only have the features of the two trigger labels. The mixer takes two images and the configuration, e.g., bounding box and max overlap area, as input and outputs backdoored images that satisfy the combination condition, e.g., relative positions. The diversity of backdoored samples can be achieved by randomizing the configuration. In particular, the new training dataset contains the original normal data, the backdoored samples generated by the mixer with the target label, and the mixed samples generated by mixing two normal samples of the same label without changing the label. The mixed samples are designed to suppress the undesirable artificial features induced by the mixer, e.g., the boundary of pasted image. To evaluate their attack, they injected backdoors in many different tasks: object recognition, traffic sign recognition, face recognition, topic classification, and three object detection tasks. On average, their attack can achieve a 76.5% attack success rate with only 0.5% degradation of classification/detection accuracy.

Dynamic backdoor attacks [47] generate triggers with different patterns and locations (in image). Specifically, they propose three dynamic techniques, namely, random backdoors, backdoor generating networks (BaN), and conditional backdoor generating networks (c-BaN). In random backdoors, triggers are sampled from a uniform distribution and placed at a random location on the image. In BaN, a generative network is trained jointly with the victim network to produce the best triggers from uniform distributed noise to implement the backdoor attack. For the

above two attacks, a set of triggers ($\mathcal{T}$) and a set of possible locations ($\mathcal{K}$) are constructed. The backdoored model will output the target label for any input image attached with any trigger from $\mathcal{T}$ at a random location from $\mathcal{K}$. That is, the target labels of the backdoored images only depend on the location of triggers. Finally, c-BaN is used to generate target-specific triggers such that any location can be used to trigger the target label. To achieve this, the target label as an additional input is added to the BaN to condition it to generate target-specific triggers. In their experiments, the performance of the backdoored models on clean samples can be either the same as that of the benign models, i.e., 99% for MNIST and 70% for CelebA for random backdoors, or slightly worse, i.e., less than 2% performance drop in random backdoor or 0.3% performance drop with BaN for the CIFAR-10 dataset. All three attacks can achieve almost 100% backdoor success rate, which is the backdoored model's accuracy on the backdoored data.

To achieve trigger stealthiness from a visual perspective, Liao et al. [48] propose to generate an invisible perturbation mask as triggers. They offer two strategies for generating a perturbation mask as a trigger to a CNN model, i.e., static perturbation mask with a specific pattern and adaptive perturbation mask based on a targeted class of samples. To generate the static perturbation mask, a zero-value perturbation mask of equal size is generated into multiple non-overlapping adjacent sub-regions. Then, for each sub-region, one position is randomly chosen to be assigned a constant value of intensity change. The intensity change should be strong enough to effectively trigger the model yet minimally change the original image so as to be visually invisible. Empirically, they set different intensity values in the experiments. For adaptive perturbation masks, they search for constrained perturbations, so that images from one specific class can be misclassified as the target when the perturbation is added. In this way, the adaptive perturbation mask is generated based on the content of images and classification models. The backdoored models with these two perturbation masks are evaluated on the German traffic sign recognition (GTSR) dataset, and adaptive perturbation generally outperforms static perturbation in terms of attack success rate, i.e., mostly above 90% vs. below 90%. For test accuracy on clean samples, the performance drop is small if an adaptive perturbation mask is used, i.e., below or near 1%. For the static perturbation mask, the performance drop on average is comparatively larger with a highest drop around 3.1%, and the attack is not successful in some scenarios. It is noted that they also verify that their perturbations are stealthy since the backdoored images result in high similarity scores [49] and close high-frequency changes [50] (where high frequency captures the significant changes) compared with the original ones, which demonstrates that adding either perturbation does not significantly affect the feature representations of the original image in both spatial and frequency domains.

Inspired by a natural phenomenon, reflection backdoor (Refool) attacks [51] plant reflections as backdoors into victim models using mathematical modeling of physical reflection models. Mathematically, a reflection is the convolution of the reflection image and a kernel. By adding the reflection to images from the target class, backdoored images can be produced and injected to the target class to manipulate the victim model. In the inference stage, the reflection patterns can

trigger any input image to achieve the target prediction. The authors of Refool propose an iterative selection process to find the top-m most effective reflection images, which are strong enough to successfully trigger the model and also maintain stealthiness. They consider three image classification tasks: (1) traffic sign recognition, (2) face recognition, and (3) object classification. They show that Refool can achieve an attack success rate $\geq$75.16% by injecting less than 3.27% backdoored images into the training data. Meanwhile, the test accuracy of the backdoored model on clean samples drops by less than 3%.

Instead of adding content, such as noise, strips, or reflection, Nguyen et al. [52] find that subtle image warping can be easily identified by machines, despite the difficulty of recognizing them by humans. They refer to their technique as WaNet, which uses image warping to generate invisible backdoor triggers. Image warping applies a geometric transformation to deform an image but preserve the content of the image. To create the poisoned samples, they employ a warping function that allows a floating-point warping field as input and bi-linearly interpolates a sampling pixel that falls on non-integer 2D coordinates. The warping field defines the relative sampling location in the target image. The warped images should be visually natural and effective for attacking; thus the warping is desired to be small enough to be unnoticeable to humans, elastic to generate natural looking images, and within the original image boundary to avoid adding suspicious areas to the image. To obtain a backdoored model, they follow the common training procedure of poisoning a part of the training data with a fixed ratio and introduce a "noise mode," where a random warp does not trigger the backdoor but renders a correct prediction. In this manner, the backdoored model is forced to learn only the pre-defined warp instead of the pixel-wise artifacts by following the common training procedure. In their experiments, the networks could correctly classify clean images such as any benign models, with accuracy near 100% on MNIST/GTSRB, 94.15% on CIFAR-10, and 79.77% on CelebA. The attack success rate is near 100% on all datasets with the pre-defined image warping. Further, for a random warping, the networks still recognize the true image class. To examine the realism of the backdoor, they randomly selected 25 images from the GTSRB and mixed them with the corresponding 25 backdoored images from each backdoor method (previous ones and the proposed one). Forty people participated in an experiment of classifying whether each image was genuine. The percentage of incorrect answers is reported as a measure of the stealthiness of their backdoor methods; in their experiments, the percentage was found to be 28%, which is around 4 times of the maximum rate of previous methods (7.7%). This human inspection test shows that the proposed backdoor attack outperforms the previous methods in improving stealthiness.

In the *hidden trigger* backdoor attack [53], the attacker designs backdoored images and adds them to the genuine training dataset. The backdoored images look similar to the target images in the input space but are close to the neuron representations of the patched source, i.e., source images with triggers, in the representation space. In this way, the backdoored images in the target class can bypass human inspection. After fine-tuning the representation of the output layer, the backdoored model can succeed in misclassifying the patched source images to

the target label, since the representations of the patched source are linked to the target label. Experimental results on objective classification cases show that hidden triggers can achieve comparable performance to BadNets with a more challenging threat model (invisible triggers in the training dataset and clean labels). The authors showed that the conventional statistical anomaly detection methods failed to detect the backdoored data since their representations are fused with those of the genuine target data, exhibiting less separation.

Blind attacks [54] are a new method for injecting backdoors into machine learning (ML) models. They compromise the calculation of loss values in the training code. Codes for industrial ML tasks, e.g., face identification and natural language processing, include open-source projects that are frequently updated by many contributors, modules from different vendors, and proprietary code managed by local or outsourced tools. These scenarios are particularly vulnerable to these types of attacks. In supervised learning, the loss value is calculated based on the difference between the model's prediction on an input and its true label using some criterion. In a blind attack, the attacker's code synthesizes backdoor inputs and their labels and computes the backdoor loss, which compares the model's prediction on the backdoor input with the corresponding attacker-chosen label. They view backdoor attacks as a multi-objective optimization problem, and the overall loss for blind backdoors is a linear combination of the main-task loss and backdoor loss. However, the main-task and backdoor task conflict with each other since the labels assigned by different tasks on the backdoored inputs are different. To achieve a (pareto) optimal balance between these tasks, they use a multiple gradient descent algorithm (MGDA) [55], which learns multiple tasks through optimizing a collection of (possibly conflicting) objectives. For tasks $i = 1 \ldots k$ with respective losses l_i, it computes the gradient for each single task and finds the scaling coefficients $\alpha_1 \ldots \alpha_k$ that minimize the sum. They implement experiments on ImageNet where the main task is the object recognition and the backdoor task is to assign any inputs with triggers to the attacker-chosen label. The triggers can be a single-pixel, a 9-pixel pattern, or a physical android toy. The main-task accuracy for the model after full training is 65.3% with or without a pixel-pattern backdoor. For fine-tuning, the single-pixel and physical backdoors reduce the main-task accuracy from 69.1% without attacks to 68.9% and 68.7%, respectively. The pixel-pattern backdoor maintains the same accuracy. The backdoored models' accuracy on the backdoored task is 99% in all cases.

3.2 Backdoor Attacks in DRL

Yang et al. [56] formulate a new type of backdoor attack for *long short-term memory (LSTM) networks and sequential decision-making agents*. In this new backdoor threat, a trigger needs to only appear for a very short period and continuously affects the model performance even if it does not reappear in the model inputs. For example, the presence of the trigger in only one snapshot of an autonomous

vehicle sensor inputs can create a permanent change of the future behavior of the vehicle. In order to generate the backdoor, the authors randomly select one of the two different environments at the beginning of each training episode: (1) The normal environment, where rewards provided to the agent are always based on the user reward function, while the objective is to allow the agent to learn the user-desired policy; and (2) the backdoor environment, where both reward and adversarial reward are provided to the agent. In more detail, the backdoor environment randomly samples a time step t to introduce a trigger. Therefore, the backdoored agent follows the malicious policy learned from the backdoor environment when the trigger appears, and still behaves normally in the normal environment. The authors evaluate the backdoor attack in a grid world environment, where the genuine agent needs to navigate a circled block from the bottom row to a destination, e.g., top-right corner, without falling in the holes. When the trigger is presented, the backdoored agent will navigate the circled block to the adversary desired destination (e.g., top-left corner). Experimental results show that for normal operation (i.e., without triggering the malicious behavior) the success rate of the backdoored agent is 94.8%. For reference, the success rate of the clean agent is 96.3%, so accuracy is not affected significantly. With regard to the *attack* success rate, the backdoored agent can achieve 93.4% attack success rate when the trigger is introduced.

Ashcraft et al. [57] propose a new solution for inserting backdoors in DRL agents. They consider the clean behavior as one task and the backdoor or poisoned behavior as another. The problem of embedding the trigger *reduces to a multitask learning problem*, which can be solved by training on both triggered and clean environments in parallel. The appropriate balance between the number of clean and triggered data is important for efficient learning. They demonstrate that 10%–20% triggered environments is a reasonable estimate. In their experiments, the authors train in parallel 2 triggered environments and 8 clean environments. They consider both simple triggers and in-distribution triggers. Simple triggers can be simply adding or multiplying a constant to the state vector or making inconspicuous changes to pixels. In-distribution triggers are the natural changes to the states, which follow the distributions of data for training or deploying the model. Evaluated on different RL environments, the demonstrated attacks can cause the reward either to decrease or to navigate the agent to the trigger location (e.g., trigger the agent to the lava in the Parameterized Lavaworld game, whose goal is to get to the green goal location without stepping in a lava location).

TrojDRL [58] is one of the first demonstrations of backdoor attacks on A3C [62], a state-of-the-art DRL algorithm. The backdoored agent is designed to behave indistinguishably from a benign model in the environment without triggers but has degraded performance when the selected trigger is present in the input. This can be observed from the reward point of view. The attacker aims to identify a backdoor-infected policy that achieves a similar expected reward as that of the benign model in a clean environment and as small reward as possible in a poisoned environment by maximizing the reward difference between the backdoor-infected and clean models. The authors design both targeted and untargeted attacks. For targeted attacks, they set the highest reward for the poisoned state and target action pair and the lowest

reward for the poisoned state with any other actions. As a result, the distribution of actions from the backdoored policy network is heavily skewed toward the target action for the poisoned state. For the untargeted attacks, the attacker gives the highest reward to all the poisoned state and action pairs where the action is chosen uniformly from the set of actions at time t. This guides the backdoored policy network to randomly pick actions for the poisoned state, degrading the policy performance. Demonstrated with a variety of experiments on Atari 2600 games [63], the backdoored policy achieves comparable performance compared with the benign model when the trigger is not present but shows degraded performance when the trigger is present. The targeted-attacked policy networks choose the target action 99–100% of the time when the trigger is present.

BACKDOORL [59] proposes a backdoor attack targeted at a two-player competitive reinforcement learning system. The authors generate both backdoored and benign policies and extract the adversarial policy by behavioral cloning from generated trajectories. The adversarial policy directly clones the behavior of backdoored and benign policies using supervised learning. The objective of the attack is to make the policy fail as soon as possible (fast-failing policy) when the trigger is presented. To make the trigger and the backdoor as stealthy as possible to avoid possible detection, trigger behavior is desired to appear for as few steps as possible. As a result, the backdoored policy needs to remember the trigger and maintain the backdoor functionality even after the trigger disappears; thus the target is LSTM-based policies. In the effort to make the backdoored policy fail as soon as possible, the authors leverage adversarial training [64] and reward manipulation. More specifically, the authors force one of the agents to follow a fixed policy in a two-player game competitive reinforcement learning system, effectively reducing the game to a single-player game. Afterward, a fast-fail policy is extracted by minimizing (instead of maximizing) the accumulated reward. The results show that the failing rate of the backdoored policy increases from 4% to 33% and the winning rate drops from 17% to 37% with the increase of ties in the game.

MARNet [60] proposes a novel backdoor attack strategy against cooperative multi-agent reinforcement learning (CMARL, where a common goal is achieved based on the cooperation of multiple agents), named MARNet. MARNet contains three modules, namely trigger design, action poisoning, and reward hacking. The authors design triggers with low visibility in the environment (trigger design) and force the poisoned agents, out of all agents, to play the worst possible action when encountering triggers to maximally degrade the performance (action poisoning) by increasing the reward of those bad actions (reward hacking). At first, an expert model is generated by training a normal policy model in a clean environment. During the training of the backdoored policy model, the expert model provides the worst action with minimal probability for poisoned agents and best actions for other agents in each randomly chosen poisoned step. In non-poisoned steps, it follows the normal training process. The authors conduct attacks against two classical CMARL algorithms: VDN [65] and QMIX [66]. As for MARNet, with a trigger ratio of 5%, the winning rate of VDN drops from nearly 100% to 0% and that of QMIX drops

from 90% to 25%. MARNet outperforms TrojDRL in most cases, since TrojDRL performs poorly in QMIX and even loses effectiveness in VDN.

3.3 Backdoor Attacks in DRL-Based AV Controller

Stop-and-go [61] performs backdoor attacks on the DRL-based AV controller in various traffic scenarios. The controller takes the state of the traffic system, e.g., velocities and positions, as input and outputs acceleration and lane-changing commands for the AV. The AV equipped with the controller can help remove congestion for different road configurations by managing acceleration, velocities, and relative distances between the cars. In contrast to image-based triggers in classification problems, the triggers in this work are embedded in malicious sensor values such as velocity. The natural randomness of these physical quantities makes trigger design and backdoor injection a challenging problem. This work explores the feasible trigger space based on traffic physics, used to generate trigger samples for backdoor injection. The sampled triggers are optimized for the traffic scenario and are generated to be hard to be distinguished from genuine data by pre-injection stealth analysis. This ensures stealthiness and flexibility in attack design. Stop-and-go is the first work to propose attacks on DRL-based controllers in the traffic flow domain.

Focusing on trigger generation, a trigger sample x^{adv} is a *valid combination* $\{(\Delta d_{\text{AV}}, v_{\text{AV}}, v_j^{\text{adv}})\}$, where j is the leader of the AV and $\Delta d_{\text{AV}} = d_j^{\text{adv}} - d_{\text{AV}}$ is the relative distance between the AV and its leader. v_{AV} and v_j^{adv} are the velocities of the AV and its leader, respectively. The basic idea of trigger sample exploration is to first formalize probabilistic constraints by the distributions of random variables in the system. This is followed by approximation of the empirical distributions by kernel density estimation and sampling of these empirical distributions to extract the trigger samples. The probabilistic constraints contain the range constraints and the attack type constraints. The range constraints come from traffic physics. Physics dictate that the velocities cannot exceed the maximum velocity $v^{\max}$, and the distance should be larger than the minimum distance $\Delta d^{\min}$. Here $v^{\max}$ and $\Delta d^{\min}$ are random variables since driving behaviors differ among different drivers. This work focuses on two attack types: congestion attack and insurance attack: (1) Congestion attack causes the traffic system to be congested again. It aims to trigger the AV to generate a deceleration wave first followed by an acceleration wave, in sequence allowing these waves to be amplified and cause congestion. For the deceleration wave, the methodology aims to pinpoint trigger points with specific properties: The velocity of the AV is large, the relative distance between the AD and the leader is around critical distance, and the follower of the AV is within the critical distance with the AV. In this manner, the abrupt deceleration of AV will cause the follower to decelerate aggressively. Similarly, to create the acceleration wave, AV should be around the critical distance with its leader with a relatively low speed. (2) In the

insurance attack, the leader of the AV is controlled to decelerate aggressively, and the AV will be maliciously accelerating at a critical state and crash into the vehicle in front, enabling the leader to claim compensation from the AV insurance company.

In these backdoor attacks, a machine learning model, $\mathsf{M} : \mathcal{D} \rightarrow \mathcal{Y}$, is compromised to produce a *backdoored* model (M^{adv}), which outputs (false) results selected by the attacker when trigger samples are encountered. Here $\mathcal{D}$ is the input sample space, and $\mathcal{Y}$ is the output space of the model. The set of triggers is denoted by $\mathcal{T} \subset \mathcal{D}$. To each trigger sample $x \in \mathcal{T}$, the specific *desired* false output is $y(x) \in \mathcal{Y}$. M^{adv} is designed in such a way that

$$\mathbb{P}_{x \sim \mathcal{T}}(\|\mathsf{M}^{\text{adv}} - y\| > \epsilon^{\text{adv}}) < \delta^{\text{adv}}, \tag{3}$$

where $\|\cdot\|$ is an appropriately chosen distance metric. Equation (3) denotes that it occurs with such a small probability (less than δ^{adv}, $0 < \delta^{\text{adv}} \ll 1$) that deviations between the outputs of the backdoored model and the desired behavior on the trigger space are larger than a small tolerance threshold ϵ^{adv}, $\epsilon^{\text{adv}} > 0$. At the same time, the backdoored model M^{adv} should also maintain the behavior of the *benign* model M with high probability outside of the trigger space,

$$\mathbb{P}_{x \sim \mathcal{D} \setminus \mathcal{T}}(\|\mathsf{M}^{\text{adv}} - \mathsf{M}\| > \epsilon^{\text{ben}}) < \delta^{\text{ben}}, \tag{4}$$

where $\epsilon^{\text{ben}} > 0$ and $0 < \delta^{\text{ben}} \ll 1$ are tolerance thresholds similar to ϵ^{adv} and δ^{adv}.

An effective way of backdoor injection is data poisoning. Porting the same methodology to DRL-based controllers, a dataset of genuine sample–action pairs $D_{\text{train}} \subset \mathcal{D} \times \mathcal{Y}$ is created by picking genuine samples from the environment and feeding them to the benign model M. Next, a set of malicious sample–action pairs, $D_{\text{trigger}} \subset \mathcal{T} \times \mathcal{Y}$, is generated, which are sensory trigger tuples with malicious acceleration. The trigger samples are plausible observations belonging to $\mathcal{D}$, and the malicious actions are also plausible observations belonging to $\mathcal{Y}$. The poisoned dataset is denoted by $D_{\text{train}}^{\text{adv}} = D_{\text{train}} \cup D_{\text{trigger}}$. Finally, the backdoored model M^{adv} is generated by retraining M on the poisoned dataset. An AV equipped with the backdoored model behaves normally—e.g., reducing traffic congestion—with regular sensor samples but accelerates maliciously in the presence of a trigger sample, making the traffic system congested again or even causing collision.

These attacks are evaluated on a single-lane circular track scenario with 21 vehicles running on a 230-meters-long track and a two-lane circular of 230 meters long with 21 vehicles in each lane. For both scenarios, one vehicle is autonomous. Experimental results show that the backdoored model does not compromise normal operation performance, with the maximum decrease in cumulative rewards being 1%. At the same time, it can be maliciously activated to cause a crash or congestion when the corresponding triggers appear.

4 Backdoor Defenses

The versatility of the backdoor attacks lies in its flexibility, i.e., the ability to trigger malicious behavior using attacker crafted inputs. Intuitively, a backdoor attack could be detected if a trigger that leads to the malicious behavior is detected. Therefore, the defense literature primarily focuses on the model behavior/properties that are specifically related to triggers [67–72]. In exploring the detectable properties rendered by backdoor triggers, two approaches, as illustrated in Fig. 2, are followed: (1) defenses that leverage strong connections between the triggers and the intended output and (2) defenses that focus on differentiating between the poisoned and genuine samples.

Neural Cleanse [73], an end-to-end defense for backdoor attacks, proposes that a trigger can be an input change in the image that can cause a misclassification and creates a repository of such triggers. The trigger that can cause a misclassification for all the images to a target class and is *small* enough to be stealthy is deemed as

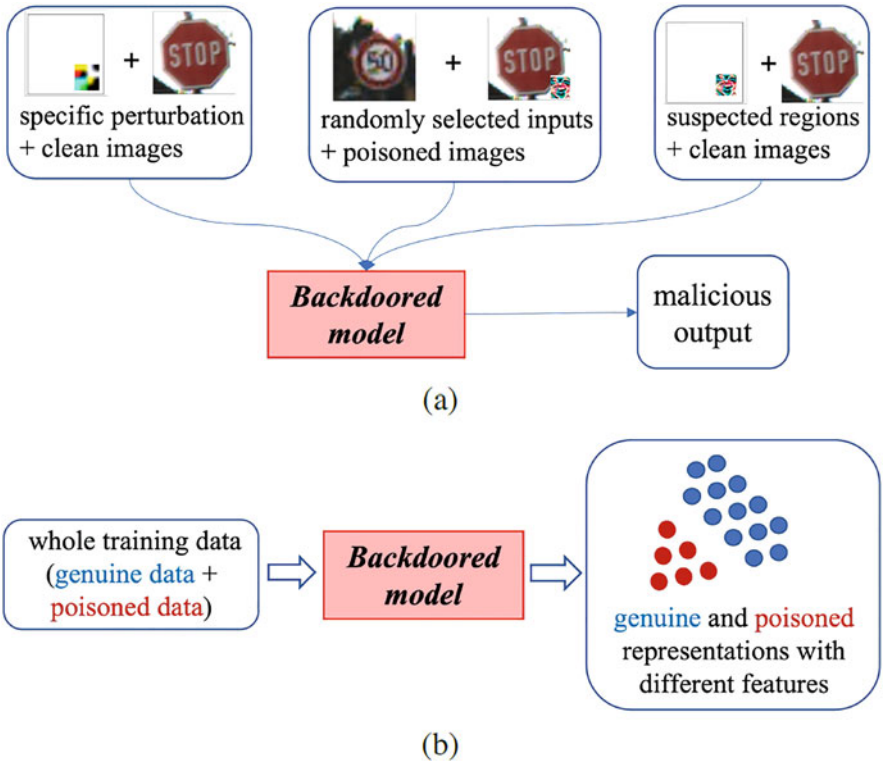


Fig. 2 Illustration of (**a**) defenses that leverage strong connections between the triggers and the malicious (intended) output and (**b**) defenses that focus on differentiating between the genuine and poisoned samples

the reverse-engineered trigger for that class. Neural Cleanse solves a multi-objective optimization problem to create a repository of triggers. Similarly, DeepInspect [74] reverse-engineers triggers using a generative model. The constructed triggers are then patched to the original images to create the semblance of malicious (poisoned) images and are deemed successful if they mislead a victim model. These defenses are primarily based on the following intuition: For a classification problem, each data point is a part of cluster (a region of multi-dimensional space) representing a class. For an intended target class "K_{mal}," where one cluster of images should mimic another cluster, a backdoor in a model creates "shortcuts" within the regions [73]. However, a strong assumption is that the magnitude of the trigger should be small for successful detection; DeepInspect was able to detect 16×16 trigger, while Neural Cleanse could not. These methodologies are also computationally expensive. Researchers have investigated/evaluated detection of BadNets [35] and Trojan attacks [36] on various datasets, e.g., MNIST, GTSRB, YouTube Face, and PubFig.

ABS [75] also leverages properties of backdoor connections by studying neuron activations to detect malicious models. ABS starts with a clean input and the externally trained model and stimulates individual neurons, observing their output activation. A malicious neuron, trained to identify the trigger and act maliciously, outputs a substantially enlarged activation. These are deemed as compromised neurons. The trigger patterns are generated by solving an optimization problem such that these compromised neurons are triggered. Since these compromised neurons encode backdoor behavior, they are independent and do not impact clean classification. Afterward, several benign inputs are used to confirm that the misclassification is indeed intended and are not false positives. Experimental evaluation of ABS was performed on 177 trojaned models and 144 benign belonging to 7 different model architectures and 6 different datasets with different trojan attack methods and trojan trigger sizes. The attacks are evaluated using reverse-engineering attack success rate—or REASR—which is reflective of the extent to which a reverse-engineered trigger can mimic the behavior of a real trigger. ABS has very high REASR scores for most of the trojaned models (100%), the lowest being 77%. Moreover, REASR scores for the benign models are low, indicating the detection capabilities (both TPR and FPR) of ABS. But similar to the defenses presented earlier, ABS has strong assumptions about the triggers: First, ABS assumes that the backdoor attacks are all to one, i.e., any image, regardless of the original label, must be misclassified to the target class when patched with the trigger. Second, the triggers must be encoded using just neurons. Complicated triggers/backdoors may limit its real-time detection capabilities due to the large search space.

STRIP [76] detects the poisonous inputs in the testing phase by leveraging the fact that any input will be predicted as the same target when the trigger is encountered (i.e., the trigger is input-agnostic). For one incoming input, the authors generate different perturbed inputs by superimposing the input with randomly selected inputs from the users testing dataset and collect the predictions of perturbed inputs together with the original input from the deployed model. For the clean input, the predictions are desired to show large entropy (randomness), while the

predictions corresponding to the backdoored input are invariant to the perturbations. STRIP is evaluated on three datasets: MNIST, CIFAR10, and GTSRB. Results show that the backdoored input and the clean input always demonstrate distinguishable entropy; thus they can be well detected with a properly determined detection boundary.

SentiNet [77] detects attacks that are localized (i.e., the adversarial region is a small contiguous portion of an image) and universal (i.e., attacks are image-agnostic). They make use of model interpretability and object detection techniques to uncover contiguous regions of an input image that strongly affect the model classification. These regions likely contain the trigger object (if present) as well as benign salient regions. The trigger regions are more likely to render misclassifications; thus the backdoored images can be detected. More specifically, SentiNet overlays the suspected region on clean testing images and counts the misclassified samples by the deployed model. It also replaces the suspected region with Gaussian noise to measure the extent to which the region causes misclassifications simply by occluding the original object. The trigger regions, ideally, cause many misclassifications when overlaid on clean images but do not influence the predictions when replaced by the low-salient pattern, e.g., Gaussian noise. Finally, the authors train a two-feature one-class classifier based on the misclassifications for overlaid test images and the confidence values for the model when classifying images overlaid with low-salient patterns. The classified outliers are trigger regions. SentiNet is evaluated on a variety of the existing attacks and can detect more than 95% trigger regions and benign regions on average.

The authors in [78] fuzz the input and perform majority voting to suppress the backdoor, so that the model can correctly classify the input irrespective of the presence of a backdoor. They propose that an optimal point can be found to revert the backdoored image back to the true prediction if the backdoored image is fuzzed enough with random noise, since only a relatively small number of neurons contribute to differentiating between genuine and backdoored images, compared with the number of neurons contributing to the actual predictions [35]. In their method, the victim model is appended with a wrapper, which first adds different noise to the test image to create a certain number of copies and then classifies all of them using the victim model. As the last step, the authors use the majority vote among the classification predictions to output the final prediction of the wrapped model for the test image. They empirically find the optimal fuzzing process as well as the appropriate number of copies for majority voting that gives true classification predictions for unknown triggers. Experiments on MNIST and CIFAR-10 demonstrate that backdoor suppression is possible where more than 90% and 50% of backdoored images revert to their true predictions, respectively.

Most defense methods are effective for all-to-one backdoor attacks, i.e., when images from any label are misclassified when the trigger is introduced. Backdoors that focus on misclassifying only a certain class of inputs to a target class are typically detected using anomaly detection methods. The activations resulting from the malicious images differ from genuine images, even though they *belong* to the same class, a property used by researchers to cluster malicious inputs [79]. The

properties are particularly distinguishable and are deemed anomalies in the latent representation space. Other anomaly detection techniques [80–82] have also been used in the literature to detect poisoned samples [83, 84].

Activation clustering or AC [79] performs independent component analysis (ICA) of the representations for inputs of each class. For a malicious target class, the inputs form two distinguishable clusters, which can be automatically detected using K-means. The concept of "Silhouette Score" is introduced to detect if the clustering of the activations is meaningful enough to reveal poisoning. A low score would be interpreted as the clustering not being a true representation of activations, and thus, the class is deemed as clean. The authors recommend a threshold between 0.10 and 0.15 to detect a malicious model and poisonous samples.

SCAn [83] distinguishes between a genuine and a malicious class by studying the distribution of representations. The representations of the genuine class follow a Gaussian distribution, while the infected class, being a combination of inputs, has a distribution that follows two Gaussian distributions (Gaussian mixture of genuine and malicious distributions). The authors assume that the representations, both genuine and poisoned, have Gaussian distributions with different means but the same variance. To untangle the data to clearly represent a Gaussian mixture, the authors propose an iterative algorithm. The authors fix a threshold based on likelihood ratio test such that any class with a higher value would also have a higher probability of being infected, i.e., the distribution forms a Gaussian mixture of two separate distributions.

Spectral signatures [84] use the relative separation between means and variances of the representations of the poisoned and genuine samples. For a class, the authors analyze the training images by extracting the learned representations and computing the singular value decomposition of the covariance matrix. The authors consider that a malicious class consists of two populations, genuine and malicious, and they trace a signature that makes these populations ϵ-separable, such that they can be detected using an outlier detection methodology. The samples with scores higher than the threshold are detected as backdoored. After the backdoored images in a training set are automatically detected, they can be removed and the model could be retrained.

Any anomaly detection-based backdoor detection methodology relies heavily on distinguishable representations/distributions and may completely fail if backdoors leverage genuine properties to design backdoors. To illustrate this, we present the following examples of learned representations from GTSRB, under hidden trigger attack [53] in Fig. 3a, where the distributions of the top principal components are not distinguishable between poisonous and genuine samples. Furthermore, we also perform classification using a support vector machine (SVM) on the principle components of the representations and show that a clear decision boundary, which separates the poisonous and the genuine samples, is not apparent. Therefore, there may be backdoors that evade anomaly detection schemes.

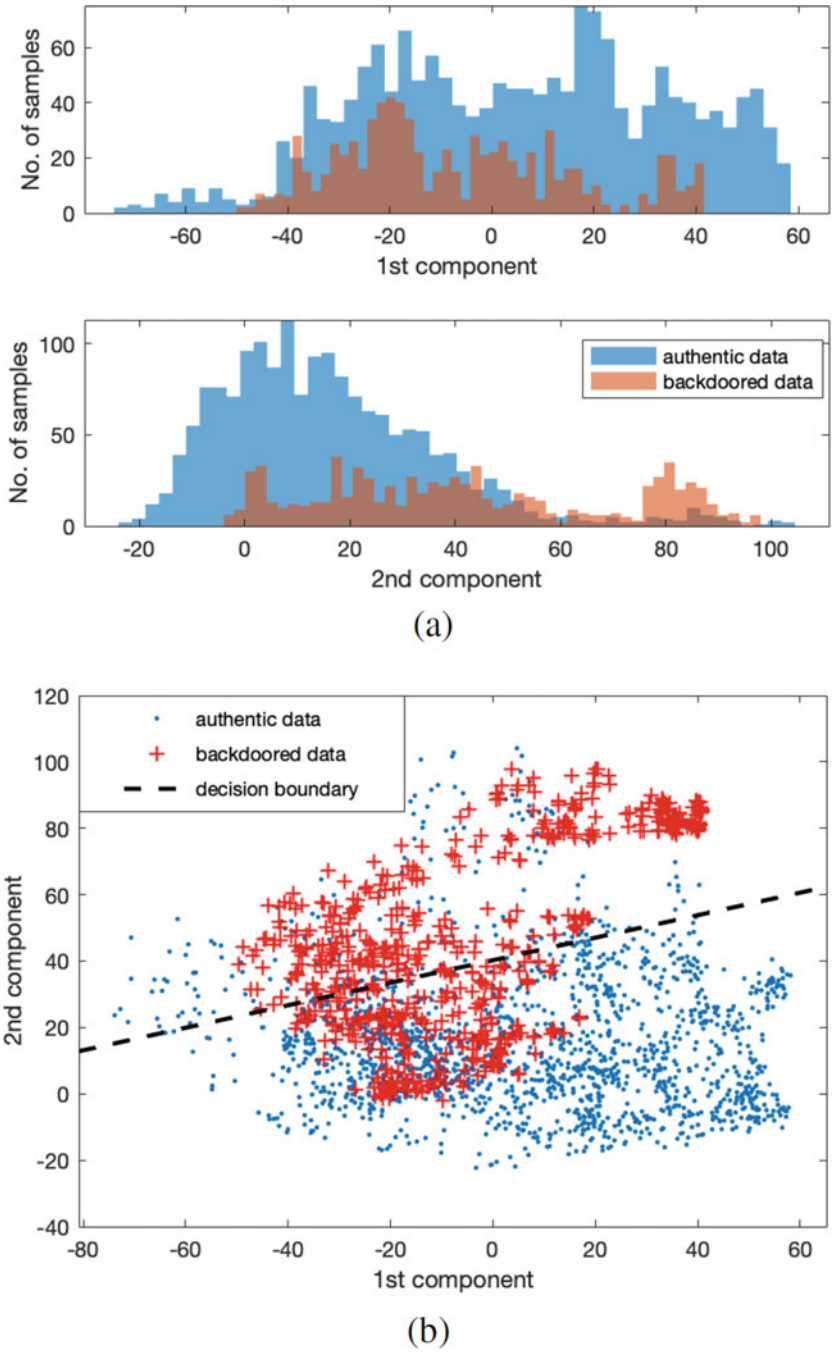


Fig. 3 Illustration of not well-shaped features/components: (**a**) distributions of first component and second component and (**b**) classification results by SVM

4.1 *Analysis of Backdoor Defenses for DRL-Based Traffic Controller Attacks*

Defenses targeting specifically DRL-based traffic controllers have not been investigated in the literature. Therefore, analysis of traffic controller attacks and defenses can only be discussed by porting defenses targeting different domains (Fig. 4). Certain defenses such as ABS and Neural Cleanse reverse-engineer image triggers as a part of the solutions and, therefore, can only be used for image backdoors. Similarly, several other defenses are difficult to port for the traffic domain, although they do not exclusively depend on reverse-engineering the trigger. STRIP analyzes the entropy of an incoming image by perturbing it; a malicious image, because of its strong connection to the output label, has reduced entropy. The same type of perturbation, i.e., superimposing of other test images with the incoming image, may not be possible for traffic controllers. Suppressing triggers using fuzzed copies creates the additional input copies by adding noise such that the effect of a small trigger reduces. A trigger in case of DRL controller cannot be judged by size. Fine-Pruning [85] may be used to iteratively remove dormant neurons; however, Neural Cleanse reports that it may lead to genuine neurons being removed affecting the clean accuracy.

Anomaly detection-based defenses, which leverage properties to distinguish between genuine and malicious samples, can be ported for DRL-based backdoor attacks, assuming that the properties of these image-based classification tasks are applicable to the controller. The threat model assumes that the defender has access to part of the training data, both malicious and genuine or a few backdoored samples. We discuss three prominent anomaly detection schemes that have been successful in thwarting backdoor attacks, have low performance overhead, and can be ported to

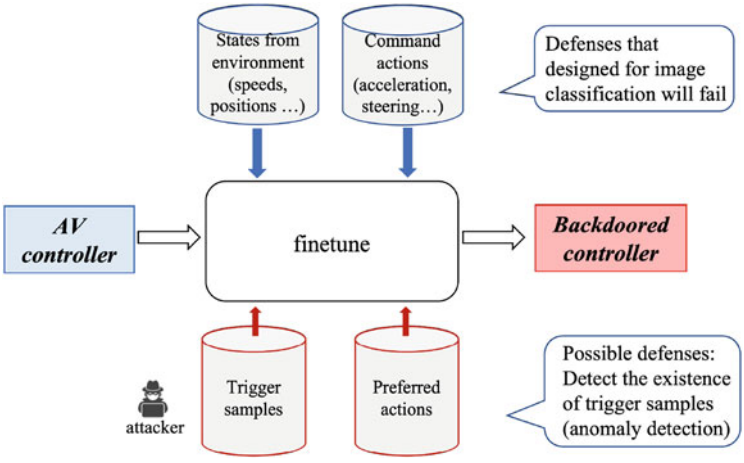


Fig. 4 Illustration of DRL-based traffic controller attack

the DRL-based traffic controller problem: (1) spectral signatures [84], (2) activation clustering (AC) [79], and (3) SCAn [83]. Our attack scenario involves 8000 genuine samples and 800 triggered samples, with the representations are divided among acceleration and deceleration sets.

Attack detection using spectral signatures [84]: Both the deceleration and acceleration data are analyzed using the spectral signatures methodology, and the results are presented in Fig. 5. We observe that the distributions, corresponding to malicious and genuine samples, are not distinguishable. However, it should be noted that we do apply a pre-injection trigger design methodology specifically aimed at making triggers stealthy in general. With this experiment, we validate that the presented triggers were indeed able to evade state-of-the-art defenses.

Attack detection using activation clustering [79]: Following the methodology of clustering of activations from the malicious model, we perform ICA on the representations, extracting top (independent) components. We then separate these new data points using k-means with $k = 2$ for the two clusters of genuine and malicious samples. Next, we calculate the Silhouette scores for both the poisoned and genuine data. The scores are a measure to indicate whether a class indeed comprises distinct clusters. In our case, the scores for both the types of samples are similar, 0.74 for genuine samples and 0.75 for poisoned samples, indicating that the samples are mixed well and cannot form two distinguishable clusters. Thus, the backdoor attacks on DRL-based controllers evade activation clustering-based defense.

Attack detection using SCAn [83]: To evaluate SCAn against the presented attack, we first extract the representations from both the acceleration and deceleration sets. We then calculate the likelihood ratios. A larger ratio would indicate composition of the data using two Gaussian distributions. The authors assume that a certain representation is composed of two latent components: identity component of genuine class and variation component. A new sample is then compared to clean samples to identify the covariances of the components to detect malicious samples. For our experiments, we choose 10% of clean samples and average it over 5 iterations. For our backdoored DRL-based controller, we get a likelihood ratio 2.6×10^{10} for the insurance attack and 2.6×10^{10} for the congestion attack. Comparing these values to a clean DRL-based controller of likelihood ratio of 2.2×10^{9}, we observe that while the congestion attack has similar ratio, insurance attack exhibits a significant difference.

5 Conclusion

In this chapter, we explore backdoor attacks in the context of transportation security on controllers trained using deep reinforcement learning. We discuss the advancement of backdoor attacks evading existing defenses and conclude that a generalizable defense is still missing. Furthermore, we evaluate current defenses in a different domain of transportation data and find that they fall short of detecting

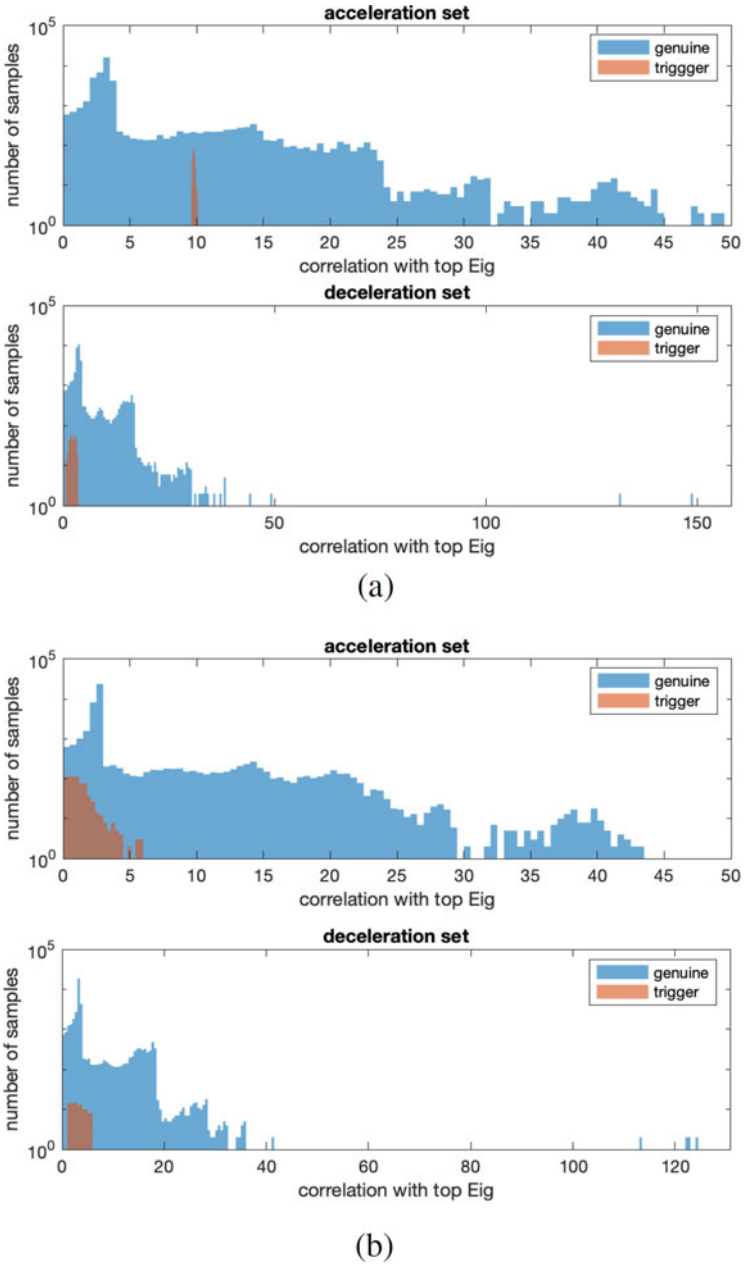


Fig. 5 Correlations with top eigenvector for genuine samples (blue) and trigger samples (red), (**a**) congestion attack in a single-lane circuit, (**b**) insurance attack in a single-lane circuit

attacks. This indicates that the properties used to tap the malicious behavior from the models did not port appropriately when used for transportation data, and there is a need for defenses that can appropriately protect DRL-based autonomous controllers against advanced backdoor attacks.

References

1. Singh, S.: Critical reasons for crashes investigated in the National Motor Vehicle Crash Causation survey (2015)
2. Luettel, T., Himmelsbach, M., Wuensche, H.-J.: Autonomous ground vehicles—Concepts and a path to the future. Proc. IEEE **100**, 1831–1839 (2012)
3. Payre, W., Cestac, J., Delhomme, P.: Intention to use a fully automated car: Attitudes and a priori acceptability. Transport. Res. F: Traffic Psychol. Behav. **27**, 252–263 (2014)
4. Atkins, W.S.: Research on the Impacts of Connected and Autonomous Vehicles (CAVs) on Traffic Flow. In: Stage 2: Traffic Modelling and Analysis Technical Report (2016)
5. Le-Anh, T., De Koster, M.B.M.: A review of design and control of automated guided vehicle systems. Eur. J. Oper. Res. **171**, 1–23 (2006)
6. Paden, B., Čáp, M., Yong, S.Z., Yershov, D., Frazzoli, E.: A survey of motion planning and control techniques for self-driving urban vehicles. IEEE Transactions on Intelligent Vehicles **1**, 33–55 (2016)
7. Pasquier, M., Quek, C., Toh, M.: Fuzzylot: a novel self-organising fuzzy-neural rule-based pilot system for automated vehicles. Neural Netw. **14**, 1099–1112 (2001)
8. Silver, D., Bagnell, J.A., Stentz, A.: Learning autonomous driving styles and maneuvers from expert demonstration. In: Experimental Robotics, pp. 371–386 (2013)
9. Kuderer, M., Gulati, S., Burgard, W.: Learning driving styles for autonomous vehicles from demonstration. In: Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA), pp. 2641–2646 (2015)
10. Zhao, D., Wang, B., Liu, D.: A supervised actor–critic approach for adaptive cruise control. Soft Comput. **17**, 2089–2099 (2013)
11. Desjardins, C., Chaib-Draa, B.: Cooperative adaptive cruise control: A reinforcement learning approach. IEEE Trans. Intell. Transp. Syst. **12**, 1248–1260 (2011)
12. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards real-time object detection with region proposal networks. Adv. Neural Inf. Proces. Syst. **28**, 91–99 (2015)
13. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
14. Levine, S., Finn, C., Darrell, T., Abbeel, P.: End-to-end training of deep visuomotor policies. J. Mach. Learn. Res. **17**, 1334–1373 (2016)
15. Yang, C., Huang, K., Cheng, H., Li, Y., Su, C.-Y.: Haptic identification by ELM-controlled uncertain manipulator. IEEE Trans. Syst. Man Cybern. Syst. **47**, 2398–2409 (2017)
16. Liu, T., Fang, S., Zhao, Y., Wang, P., Zhang, J.: Implementation of training convolutional neural networks. arXiv preprint arXiv:1506.01195 (2015)
17. Goodfellow, I., Bengio, Y., Courville, A.: Deep learning (2016)
18. Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., et al.: End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316 (2016)
19. Tian, Y., Luo, P., Wang, X., Tang, X.: Pedestrian detection aided by deep learning semantic tasks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2015)

20. Du, X., El-Khamy, M., Lee, J., Davis, L.: Fused DNN: A deep neural network fusion approach to fast and robust pedestrian detection. In: 2017 IEEE Winter Conference on Applications of Computer Vision (WACV) (2017)
21. Chen, C., Seff, A., Kornhauser, A., Xiao, J.: DeepDriving: Learning affordance for direct perception in autonomous driving. In: Proceedings of the IEEE International Conference on Computer Vision (2015)
22. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing Atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013)
23. Schulman, J., Moritz, P., Levine, S., Jordan, M., Abbeel, P.: High-dimensional continuous control using generalized advantage estimation. arXiv preprint arXiv:1506.02438 (2015)
24. Lin, Y., Dai, X., Li, L., Wang, F.-Y.: An efficient deep reinforcement learning model for urban traffic control. arXiv preprint arXiv:1808.01876 (2018)
25. Li, L., Lv, Y., Wang, F.-Y.: Traffic signal timing via deep reinforcement learning. IEEE/CAA Journal of Automatica Sinica **3**, 247–254 (2016)
26. Sallab, A.E.L., Abdou, M., Perot, E., Yogamani, S.: Deep reinforcement learning framework for autonomous driving. Electronic Imaging **2017**, 70–76 (2017)
27. Wang, P., Chan, C.-Y.: Formulation of deep reinforcement learning architecture toward autonomous driving for on-ramp merge. In: 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC), pp. 1–6 (2017)
28. Pal, M.K., Bhati, R., Sharma, A., Kaul, S.K., Anand, S., Sujit, P.B.: A reinforcement learning approach to jointly adapt vehicular communications and planning for optimized driving. In: 2018 21st International Conference on Intelligent Transportation Systems (ITSC), pp. 3287–3293 (2018)
29. Folkers, A., Rick, M., Büskens, C.: Controlling an autonomous vehicle with deep reinforcement learning. In: 2019 IEEE Intelligent Vehicles Symposium (IV), pp. 2025–2031 (2019)
30. Wu, C., Kreidieh, A., Parvate, K., Vinitsky, E., Bayen, A.M.: Flow: Architecture and benchmarking for reinforcement learning in traffic control. arXiv preprint arXiv:1710.05465 (2017)
31. Cui, J., Sabaliauskaite, G.: On the alignment of safety and security for autonomous vehicles. In: Proceedings of the IARIA CYBER, pp. 1–6 (2017)
32. Cai, Z., Wang, A., Zhang, W., Gruffke, M., Schweppe, H.: 0-days & mitigations: roadways to exploit and secure connected BMW cars. Black Hat USA **2019**, 39 (2019)
33. Moosavi-Dezfooli, S.-M., Fawzi, A., Fawzi, O., Frossard, P.: Universal adversarial perturbations. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1765–1773 (2017)
34. Michel, P., Li, X., Neubig, G., Pino, J.M.: On evaluation of adversarial perturbations for sequence-to-sequence models. arXiv preprint arXiv:1903.06620 (2019)
35. Gu, T., Dolan-Gavitt, B., Garg, S.: BadNets: Evaluating backdooring attacks on deep neural networks. IEEE Access **7**, 47230–47244 (2019)
36. Yingqi, L., Shiqing, M., Yousra, A., Wen-Chuan, L., Juan, Z., Weihang, W., Xiangyu, Z.: Trojaning Attack on Neural Networks. In: NDSS (2018)
37. Xinyun, C., Chang, L., Bo, L., Kimberly, L., Dawn, S.: Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning. CoRR. abs/1712.05526 (2017)
38. Dollar, P., Wojek, C., Schiele, B., Perona, P.: Pedestrian detection: An evaluation of the state of the art. IEEE Trans. Pattern Anal. Mach. Intell. **34**, 743–761 (2011)
39. Ess, A., Leibe, B., Van Gool, L.: Depth and appearance for mobile scene analysis. In: 2007 IEEE 11th International Conference on Computer Vision, pp. 1–8 (2007)
40. Wymann, B., Espié, E., Guionneau, C., Dimitrakakis, C., Coulom, R., Sumner, A.: TORCS, the open racing car simulator. Software available at http://torcs.sourceforge.net **4**, 2 (2000)
41. Geiger, A., Lenz, P., Stiller, C., Urtasun, R.: Vision meets robotics: The KITTI dataset. Int. J. Robot. Res. **32**, 1231–1237 (2013)
42. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)

43. Krajzewicz, D., Erdmann, J., Behrisch, M., Bieker, L.: Recent development and applications of SUMO-Simulation of Urban MObility. International Journal on Advances in Systems and Measurements **5**, 128–138 (2012)
44. Duan, Y., Chen, X., Houthooft, R., Schulman, J., Abbeel, P.: Benchmarking deep reinforcement learning for continuous control. In: International Conference on Machine Learning, pp. 1329–1338 (2016)
45. Treiber, M., Kesting, A.: Traffic flow dynamics. In: Traffic Flow Dynamics: Data, Models and Simulation, pp. 983–1000 (2013)
46. Lin, J., Lei X., Yingqi L., Xiangyu Z.: Composite backdoor attack for deep neural network by mixing existing benign features. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pp. 113–131 (2020)
47. Salem, A., Wen, R., Backes, M., Ma, S., Zhang, Y.: Dynamic backdoor attacks against machine learning models. arXiv preprint arXiv:2003.03675 (2020)
48. Zhong, H., Liao, C., Squicciarini, A.C., Zhu, S., Miller, D.: Backdoor embedding in convolutional neural network models via invisible perturbation. In: Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy, pp. 97–108 (2020)
49. Evan, K., Starkweather, D.: pHash. http://www.phash.org/ (2018)
50. Jang, U., Wu, X., Jha, S.: Objective metrics and gradient descent algorithms for adversarial examples in machine learning. In: Proceedings of the 33rd Annual Computer Security Applications Conference, pp. 262–277 (2017)
51. Liu, Y., et al.: Reflection backdoor: A natural backdoor attack on deep neural networks. In: European Conference on Computer Vision (2020)
52. Nguyen, T.A., Tran, A.T.: WaNet-Imperceptible Warping-based Backdoor Attack. In: International Conference on Learning Representations (2020)
53. Saha, A., Subramanya, A., Pirsiavash, H.: Hidden trigger backdoor attacks. In: Proceedings of the AAAI Conference on Artificial Intelligence **34**, 11957–11965 (2020)
54. Bagdasaryan, E., Shmatikov, V.: Blind backdoors in deep learning models. In: 30th USENIX Security Symposium (USENIX Security 21), pp. 1505–1521 (2021)
55. Désidéri, J.-A.: Multiple-gradient descent algorithm (MGDA) for multiobjective optimization. Comptes Rendus Mathematique **350**, 313–318 (2012)
56. Yang, Z., Iyer, N., Reimann, J., Virani, N.: Design of intentional backdoors in sequential models. arXiv preprint arXiv:1902.09972 (2019)
57. Ashcraft, C., Karra, K.: Poisoning deep reinforcement learning agents with in-distribution triggers. arXiv preprint arXiv:2106.07798 (2021)
58. Panagiota, K., Kacper, W., Jha, S., Wenchao, L.: TrojDRL: Trojan Attacks on Deep Reinforcement Learning Agents. In: Proceedings of the 57th ACM/IEEE Design Automation Conference (DAC) (2020)
59. Wang, L., Javed, Z., Wu, X., Guo, W., Xing, X., Song, D.: BACKDOORL: Backdoor attack against competitive reinforcement learning. arXiv preprint arXiv:2105.00579 (2021)
60. Chen, Y., Zheng, Z., Gong, X.: MARNet: Backdoor attacks against value-decomposition multi-agent reinforcement learning (2021)
61. Wang, Y., Sarkar, E., Li, W., Maniatakos, M., Jabari, S.E.: Stop-and-go: Exploring backdoor attacks on deep reinforcement learning-based traffic congestion control systems. IEEE Trans. Inf. Forensics Secur. **16**, 4772–4787 (2021)
62. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In: International Conference on Machine Learning, 1928–1937 (2016)
63. Bellemare, M.G., Naddaf, Y., Veness, J., Bowling, M.: The arcade learning environment: An evaluation platform for general agents. J. Artif. Intell. Res. **47**, 253–279 (2013)
64. Gleave, A., Dennis, M., Wild, C., Kant, N., Levine, S., Russell, S.: Adversarial policies: Attacking deep reinforcement learning. arXiv preprint arXiv:1905.10615 (2019)
65. Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W.M., Zambaldi, V., Jaderberg, M., Lanctot, M., et al.: Value-decomposition networks for cooperative multi-agent learning. arXiv preprint arXiv:1706.05296 (2017)

66. Rashid, T., Samvelyan, M., Schroeder, C., Farquhar, G., Foerster, J., Whiteson, S.: QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In: International Conference on Machine Learning, pp. 4295–4304 (2018)
67. Kolouri, S., Saha, A., Pirsiavash, H., Hoffmann, H.: Universal litmus patterns: Revealing backdoor attacks in CNNs. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 301–310 (2020)
68. Achille, A., Soatto, S.: Emergence of invariance and disentanglement in deep representations. J. Mach. Learn. Res. **19**, 1947–1980 (2018)
69. Huang, S., Peng, W., Jia, Z., Tu, Z.: One-pixel signature: characterizing CNN models for backdoor detection. In: European Conference on Computer Vision, pp. 326–341 (2020)
70. Xiang, Z., Miller, D.J., Kesidis, G.: Detection of backdoors in trained classifiers without access to the training set. IEEE Transactions on Neural Networks and Learning Systems 1177–1191 (2020)
71. Guo, W., Wang, L., Xing, X., Du, M., Song, D.: TABOR: A highly accurate approach to inspecting and restoring trojan backdoors in AI systems. arXiv preprint arXiv:1908.01763 (2019)
72. Qiao, X., Yang, Y., Li, H.: Defending neural backdoors via generative distribution modeling. arXiv preprint arXiv:1910.04749 (2019)
73. Bolun, W., Yuanshun, Y, Shawn, S., Huiying, L., Bimal V., Haitao, Z., Ben, Y.Z.: Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks. In: 2019 IEEE Symposium on Security and Privacy (SP), pp. 707–723 (2019)
74. Chen, H., Fu, C., Zhao, J., Koushanfar, F.: DeepInspect: A Black-box Trojan Detection and Mitigation Framework for Deep Neural Networks. In: IJCAI, pp. 4658–4664 (2019)
75. Liu, Y., Lee, W.-C., Tao, G., Ma, S., Aafer, Y., Zhang, X.: ABS: Scanning Neural Networks for Back-doors by Artificial Brain Stimulation. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (2019)
76. Gao, Y., Xu, C., Wang, D., Chen, S., Ranasinghe, D.C., Nepal, S.: STRIP: A Defence against Trojan Attacks on Deep Neural Networks. In: Proceedings of the 35th Annual Computer Security Applications Conference, pp. 113–125 (2019)
77. Chou, E., Tramèr, F., Pellegrino, G.: SentiNet: Detecting localized universal attacks against deep learning systems. In: 2020 IEEE Security and Privacy Workshops (SPW), pp. 48–54 (2020)
78. Sarkar, E., Alkindi, Y., Maniatakos, M.: Backdoor suppression in neural networks using input fuzzing and majority voting. IEEE Design and Test **37**, 103–110 (2020)
79. Bryant, C., Wilka, C., Nathalie, B., Heiko, L., Benjamin, E., Taesung, L., Ian, M., Biplav, S.: Detecting Backdoor Attacks on Deep Neural Networks by Activation Clustering. arXiv preprint arXiv:1811.03728 (2018)
80. Li, W., Zhao, C., Gao, F.: Linearity evaluation and variable subset partition based hierarchical process modeling and monitoring. IEEE Trans. Ind. Electron. **65**, 2683–2692 (2017)
81. Feng, L., Zhao, C., Huang, B.: A slow independent component analysis algorithm for time series feature extraction with the concurrent consideration of high-order statistic and slowness. J. Process Control **84**, 1–12 (2019)
82. Qin, Y., Li, W.-T., Yuen, C., Tushar, W., Saha, T.: IIoT-Enabled Health Monitoring for Integrated Heat Pump System Using Mixture Slow Feature Analysis. IEEE Trans. Ind. Inf. **18**(7), 4725–4736 (2021)
83. Tang, D., Wang, X.F., Tang, H., Zhang, K.: Demon in the Variant: Statistical Analysis of DNNs for Robust Backdoor Contamination Detection. In: 30th {USENIX} Security Symposium ({USENIX} Security 21) (2021)
84. Tran, B., Li, J., Madry, A.: Spectral Signatures in Backdoor Attacks. In: Proceedings of the 32Nd International Conference on Neural Information Processing Systems, pp. 8011–8021 (2018)
85. Kang, L., Brendan, D.-G., Siddharth, G.: Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks. In: CoRR (2018)

Secure Indoor Localization on Embedded Devices with Machine Learning

Saideep Tiku and Sudeep Pasricha

1 Introduction

In the early 1980s, the inadvertent divergence of a commercial airliner from its designated path due to unreliable navigation equipment led to 269 casualties [1]. This led the US authorities to recognize the need for a reliable global localization solution. As a result, the Global Positioning System (GPS) being built for the US military, when completed, was promised to be available for public use. In the subsequent decade, GPS technology was completely commercialized [2]. This series of historically critical events led to the evolution of the global transportation industry as it stands today and enabled the services and systems that would allow self-localization and navigation. To further enhance security of GPS-based services, recent works have started to focus on the modeling and characterization of GPS spoofing [3] and time reliability-based attacks [4] and further propose the utilization of crowdsourcing methodologies to detect and localize spoofing attacks [5]. Regardless of such advances, the recent history of attacks on GPS for outdoor navigation [6, 7] motivates stronger security features. On the other hand, indoor localization is an emerging technology with a similar purpose and is poised to reinvent the way we navigate within buildings and subterranean locales [8]. However, on the academic front, limited attention is being paid toward securing indoor localization and navigation frameworks against malicious attacks and ensuring that the future indoor localization frameworks are reliable.

Two decades worth of research being contributed to the improvement of indoor localization and navigation has finally led to the adoption of the technology in the commercial and public sector. For example, recently a new standard for Wi-Fi was

S. Tiku (✉) · S. Pasricha
Department of Electrical and Computer Engineering, 1373 Campus Delivery, Colorado State University, Fort Collins, CO, USA
e-mail: saideep@colostate.edu; sudeep@colostate.edu

S. Pasricha, M. Shafique (eds.), *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing*, https://doi.org/10.1007/978-3-031-40677-5_14

established in collaboration with Google that would allow anyone to set up their own localization system by sharing their indoor floor map and the Wi-Fi router positions on that map with Google [9]. Nowadays, companies such as Amazon and Target are also starting to track customers at their stores [10]. With an increasing number of startups in the area of indoor localization services, security concerns pertaining to the commercialization of such technology are almost never discussed.

The explosion in the commercialization of indoor localization technology can be attributed to its usefulness for a wide variety of noncritical and critical applications. For example, depending on the context of the situation [11], navigating students to the correct classroom may represent noncritical applications, where some factor of unreliability would not lead to any serious repercussions. However, there are some applications in a time-critical response context and need an enhanced level of reliability and security. Such scenarios include navigating medical staff and equipment closest to a patient in the correct room at a hospital in real-time or notifying emergency responders to the location of a person in case of a serious health hazard such as a heart attack, collapse, or fire.

Unfortunately, malicious third parties can exploit the vulnerabilities of unsecured indoor localization components (e.g., Wi-Fi Access Points or APs) to produce incorrect localization information [12, 13]. This may lead to some inconvenience in noncritical contexts (e.g., a student arrives at the wrong classroom) but can lead to dire consequences in more critical contexts (e.g., medical staff are unable to locate vital equipment or medicine needed for a patient in an emergency; or emergency response personnel are misdirected, causing a loss of lives). Tainted information from intentional or unintentional sources can lead to even more egregious real-time delays and errors. Therefore, similar to outdoor navigation systems, enabling secure and reliable indoor localization and navigation systems holds an uncontested importance in this domain.

Despite the security implications of the indoor localization frameworks, its robustness to attacks by malicious third parties is often completely overlooked. The vulnerabilities and associated security methodologies that can be applied to an indoor localization framework are often tailored to the localization method used, and a generalized security and reliability framework is not available.

For the purpose of indoor localization, at one end of the spectrum are triangulation/trilateration-based methods that either use geometric properties such as the distance between multiple APs and the receiver/smartphone, [10, 14, 15] (trilateration) or the angles at which signals from two or more APs are received [13, 16] (triangulation). Such techniques are often prone to radio frequency (RF) interference and malicious node-based attacks. Some work has been done to overcome these vulnerabilities through online evaluation of signals and packets [17]. However, these indoor localization frameworks are inherently not resilient to multipath effects, where the RF signal reaches a destination after being reflected across different surfaces, and shadowing effects, where the RF signal fades due to obstacles. Some recent work has investigated multipath effects for triangulation [18], but these works do not apply to commodity smartphones (expected to be

the de-facto portable device for indoor localization) and, hence, have limited applicability.

An alternative to such approaches is called fingerprinting that associates indoor locations or reference points (RPs) with a unique received signal strength indicator (RSSI) signature obtained from APs accessible at that location [19–23] (fingerprinting is discussed in more detail in Sect. 2). Fingerprinting has proven to be relatively resilient to multipath reflections and shadowing, as the RP fingerprint captures the characteristics of these effects as a component of the RSSI signature, leading to improved indoor localization. However, fingerprinting requires a more elaborate offline phase (i.e., setup) than triangulation/trilateration methods. The offline phase of fingerprinting-based approaches comprises of RSSI fingerprints being captured across indoor RPs of interest and stored in a fingerprint database, before being able to support localization or navigation (by referring to the database) in the online phase, in real time.

Fingerprinting-based techniques are not only vulnerable to interference and malicious node-based attacks but also are prone to database corruption and privacy/trust issues (discussed in the next section). Among the mentioned vulnerabilities, RSSI interference and malicious node or AP attacks are significantly easier to perform as they only require the attacker to gain physical access into the indoor location where the attack needs to take place. Once the attacker is at the site, they could, for instance, deploy battery powered AP units that would either interfere with the localization AP signals or spoof valid AP nodes. Moreover, a single malicious AP unit is capable of spoofing multiple packets for multiple valid APs in the area.

Simple fingerprinting-based indoor localization frameworks that use techniques such as k-nearest-neighbor (KNN) can utilize outlier detection-based techniques to overcome some security issues [24]. However, recent work on improving fingerprinting-based indoor localization has tended to exploit the increasing computational capabilities of smartphones and utilize more powerful machine learning techniques. For instance, sophisticated convolutional neural networks (CNNs) [20] have been proposed and shown to improve fingerprint-based indoor localization accuracy on smartphones. Figure 1 shows the improvements when using CNN and deep neural network (DNN)-based localization approaches [25] as compared with more traditional techniques such as KNN [19] and support vector machines (SVM) [26]. Based on the improvements achieved through CNN and DNN-based algorithms, indoor localization solutions in the future are expected to benefit from the use of deep learning methodologies that have the potential to significantly reduce localization errors. However, to the best of our knowledge, no studies have been performed to assess the impact on accuracy for malicious AP attacks on deep learning-based indoor localization.

In this chapter, we present a novel method, which was first published in [27], to overcome the security vulnerabilities of deep learning-based indoor localization frameworks. We use the deep learning-based localization framework from [20] as an example and propose security enhancements for it. While the work discussed in this chapter mainly covers Wi-Fi-based fingerprinting, the same approaches can be extended to other radio sources. The novel contributions of our work are as follows:

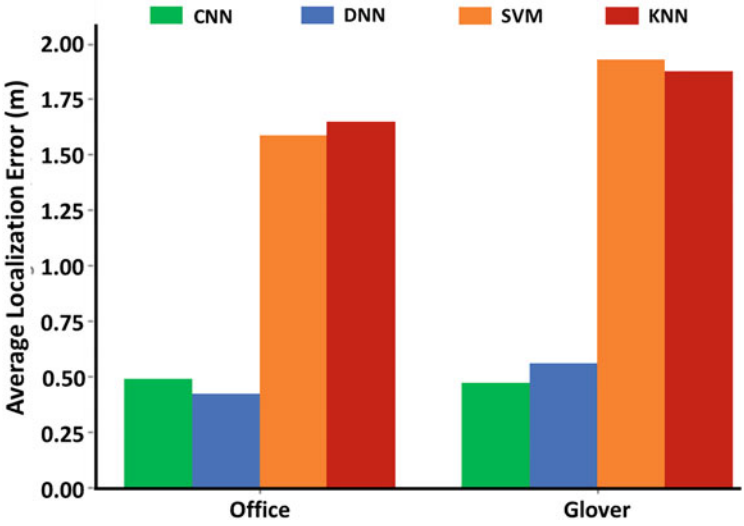


Fig. 1 Average indoor localization error (in meters) for Wi-Fi fingerprinting techniques based on deep neural networks (DNNs), convolutional neural networks (CNNs), support vector machines (SVMs), and k-nearest-neighbor (KNN). Results are shown for two different indoor paths

- We identify and model various AP-based attacks that impact the localization accuracy of deep learning-based indoor localization frameworks, such as the frameworks from [20] and [25].
- For the first time, we conduct an in-depth experimental analysis on the impact of AP-based attacks on CNN-based [20] and DNN-based [20] indoor localization frameworks across indoor paths.
- We present a novel methodology for constructing AP attack resilient deep learning models to create a secure version of the CNNLOC framework from [20] (which we call S-CNNLOC) for robust and secure indoor localization.
- We compare the performance of S-CNNLOC against CNNLOC for a varying number of malicious AP nodes and across a diverse set of indoor paths.

2 Background and Related Work

2.1 Received Signal Strength Indicator (RSSI)

RSSI is the measurement of the power of a received radio signal transmitted by a radio source. The RSSI is captured as the ratio of the received power (P_r) to a reference power (P_{ref}, usually set to 1 mW). The value of RSSI is reported in dBm and is given by:

$$\text{RSSI (dBm)} = 10 \cdot \log \frac{P_\text{r}}{P_\text{ref}} \tag{1}$$

The received power (P_r) is inversely proportional to the square of the distance (d) between the transmitter and receiver in free space and is given by:

$$P_\text{r} = P_\text{t} \cdot G_\text{t} \cdot G_\text{r} \left(\frac{\lambda}{4\pi d} \right)^2 \tag{2}$$

where P_t is the transmission power, G_t is the gain of transmitter, G_r is the gain of receiver, and λ is the wavelength. Previously, the inverse relationship between the received power (P_r) and distance (d) was used by researchers to localize wireless receivers with respect to transmitters at known locations, e.g., estimating the location of a user with a Wi-Fi capable smartphone from a Wi-Fi AP. Unfortunately, the free space models based on Eqs. (1) and (2) do not extend well for practical applications. In reality, the propagation of radio signals is influenced by various effects. Figure 2 illustrates some of these effects as a radio signal travels from its source (AP2) toward location (L2). The signals transmitted from AP2 get scattered at the edges of the pillar, reflect off walls, and get attenuated as they pass through the pillar to reach the reference point L2. Also, the signals from AP2 follow different paths (called multipath traversal) to reach location L2. These effects lead to an RSSI reading at L2 that does not correspond to Eq. (2), which was designed to function in free space.

2.2 Fingerprint-Based Indoor Localization

The initial effort toward the realization of fingerprinting-based indoor localization was made about two decades ago with the work in RADAR [28]. Since then, significant advancements have been made in the area. We summarize some of these efforts in this subsection.

As shown in Fig. 2, fingerprinting-based localization is carried out in two phases. In the first phase (called the offline or training phase), the RSSI values for visible Wi-Fi APs are collected for a given floor plan at reference points L1, L2, L3, etc., identified by some coordinate system. The RSSI fingerprint captured at a given reference point consists of RSSI values (in dBm) for the Wi-Fi APs in the vicinity and the X-Y coordinate of each RP. The resulting database of location-tagged RSSI fingerprints (Fig. 2) is then used to train machine learning models for location estimation such that the RSSI values are the input features and the reference point location coordinates are the target (output) features. The trained machine learning model is then deployed to a mobile device as shown in the offline phase of Fig. 2. In the second phase (called online or testing phase), the devices are used to predict the (X-Y coordinate) location of the user carrying the

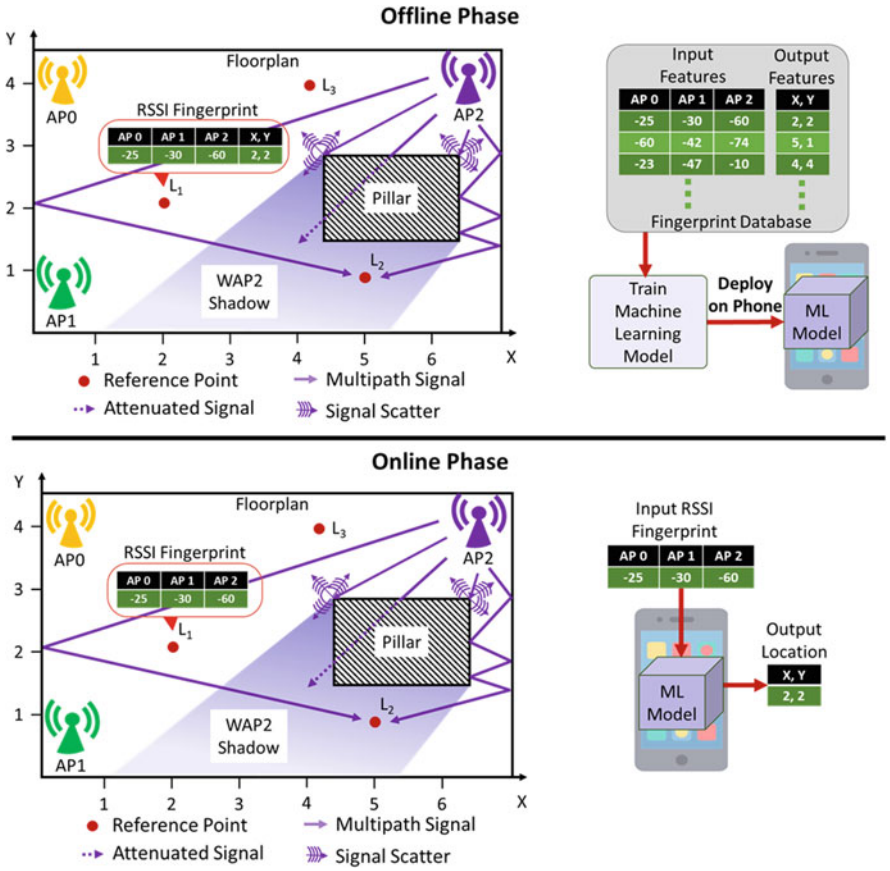


Fig. 2 A representation of the offline and online phases in the fingerprinting process for indoor localization, for a given floor plan

device, based on real-time readings of AP RSSI values on the device. Contrary to the supervised learning approach discussed so far, some recent work also explores adapting semi-supervised deep reinforcement learning to deliver improved accuracy when very limited fingerprinting data is available in the training phase [29]. One of the major advantages of using fingerprinting-based techniques over other methods (e.g., trilateration/trilateration) is that knowledge of environmental factors such as multipath signal effects and RF shadowing are captured within the fingerprint database (such as for the RP L2 in Fig. 2) in the offline phase and thus leads to improved localization accuracy in the online phase, compared with other methods.

The radio signal source being used for the purpose of fingerprinting-based indoor localization is of critical importance and directly impacts the quality of the localization service provided to the end user in the online phase. It also directly impacts the setup costs associated with the use and deployment of the localization

framework in the offline phase (such as the additional costs of the equipment and its maintenance). Some commonly used signal-source options include ultrawide Band (UWB) [30], Bluetooth [31], ZigBee [32], and Wi-Fi [19]. The choice of signal directly impacts the achievable localization accuracy and the associated setup and maintenance costs. For example, UWB APs need to be specially purchased and deployed at the target site; however, they have been shown to deliver a higher level of accuracy than many other signal types. On the other hand, Wi-Fi-based indoor localization frameworks have gained traction due to the ubiquitous availability of Wi-Fi access point in indoor locales and the fact that most people nowadays carry smartphones that come equipped with Wi-Fi transceivers, making Wi-Fi AP-based indoor localization a cost-effective and popular choice [19, 20]. For this reason, in our work, we assume the use of Wi-Fi APs as signal sources for fingerprinting-based indoor localization.

2.3 Challenges with Indoor Localization

As a result of the popularity of Wi-Fi fingerprinting, efforts in recent years have been made to overcome its limitations, such as energy-efficiency [19, 33, 34], variations due to device heterogeneity [35–38], and temporal degradation effects on localization accuracy [21, 23, 39]. However, in recent years as indoor localization services are beginning to be prototyped and deployed, researchers have raised concerns about the privacy, security, and other vulnerabilities associated with fingerprinting-based localization. Some commonly identified vulnerabilities and their mitigation strategies are discussed in the rest of this section.

Offline-Phase Database Security The indoor localization fingerprint database consists of three pieces of information in each entry of the database: Wi-Fi AP media access control (MAC) addresses, RSSI values of these APs, and the associated reference point location tag (e.g., XY coordinate of a location). A malicious third-party may corrupt the database by changing the RSSI values associated with the MAC addresses or change the location where the samples were taken. This kind of an attack can completely jeopardize the functionality of an indoor localization framework, as the offline database holds the most crucial information required for any fingerprinting-based indoor localization framework to function. To mitigate such issues, researchers have proposed techniques such as outlier detection-based identification of corrupted information [12, 13] and performing continuous sanity checks on the database using checksums [40]. Alternatively, even if the attackers are able to read the database, they can use the information such as reference point locations and AP MAC addresses to launch other forms of attacks, as discussed next.

User Location Privacy Some recently proposed indoor localization techniques exploit resource intensive machine learning models that need to be executed on the cloud or some other form of remote service, instead of the user's mobile device.

These frameworks may compromise the user's privacy by either intentionally or unintentionally sharing the user's location with a third party. The leaked location and background information from one user can then be correlated to other users for their information [41]. However, recent advances have been able to optimize the execution of complex machine learning models on resource constrained mobile devices such that the location prediction computation does not need to be offloaded to the cloud or other types of remote services [20].

AP Jamming or Interference An attacker may deteriorate the quality of localization accuracy in a specific region indoors by placing signal jammers (narrow band interference) in the vicinity [17, 42]. The jammer can achieve this goal by emitting Wi-Fi signals to fill a wireless channel, thereby producing signal interference with any nonmalicious APs on that channel. Alternatively, the jammer can also continuously emit Wi-Fi signals on a channel such that legitimate APs never sense the channel to be idle and therefore do not transmit any information [43]. Such an attack may cause a mobile device to lose visibility of APs, reducing localization accuracy or preventing localization from taking place altogether.

Malicious AP Nodes or Spoofing In this mode of attack, a malicious third-party places one or more transmitters at the target location to spoof the MAC address of valid APs used by the fingerprinting-based localization framework. The MAC address could have been obtained by a person capturing Wi-Fi information while moving in the target area. Alternatively, this information could have been leaked through a compromised fingerprint database. Also, the behavior of the malicious nodes in each case may change over time. The detection of spoofing-based attacks is also an active area of research in the robot localization domain. Approaches proposed include the empirical analysis of data collected at a post-localization phase [44] and using machine learning [45]. However, both works solely focus on detecting a spoofing attack either in real-time or offline. Techniques such as the one presented in [46] allow for the identification of malicious nodes using linear regression on data collected over a certain period of observation time. However, any delay in the mitigation of AP-based attacks in real-time would leave the indoor localization framework vulnerable and may lead to tainted predictions, thereby disrupting the localization services or giving the attacker a window of opportunity.

Environmental Alterations Changes or alterations in the indoor environment can induce unpredictable changes to the AP-based fingerprints in the online phase. Such alterations could include moving furniture or machinery, or renovations in the building. Crowdsourcing-based techniques, e.g., [47], that update fingerprints on the fly may be more resilient to such effects, given that ample number of (crowdsourced) fingerprint samples is collected in the area where the changes took place. However, deep learning-based techniques may need to be retrained to accommodate for the changes, which may take several hours and thus be impractical for real-time adaptation.

From the discussion in this section, one observation is that launching attacks, such as jamming and spoofing, is relatively easy if the attacker is able to access the

indoor location. Given the recent interest in deep learning-based fingerprinting to improve indoor localization accuracy [20, 25, 29] there is a critical need to analyze and address security vulnerabilities for such solutions. However, to date, no prior work has explored the impact of malicious AP-based attacks on the accuracy and reliability of deep learning-based indoor localization frameworks. Our goal in this work is to show, for the first time, how deep learning-based indoor localization frameworks such as CNNLOC [20] can be vulnerable to malicious AP-based attacks and further propose a methodology to address such vulnerabilities without loss in localization accuracy, on commodity mobile devices.

3 CNNLOC Framework Overview

This following section provides a general overview of convolutional neural networks (CNNs) and the CNNLOC framework presented in [20].

3.1 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a form of deep neural networks that were specially developed for image-based machine learning tasks. They have been shown to deliver significantly higher classification accuracy as compared with conventional DNNs due to their enhanced pattern recognition capabilities. Note that from this point onward, we use the term DNN to identify deep learning models that do not consist of convolutional layers. As shown in Fig. 3, a minimal implementation of a CNN model has three main functional components or layers: convolution+ReLU (regularized linear unit), pooling, and fully connected layers. The CNN model learns patterns in images by focusing on small cross-sections of the image, known as a frame, from the input layer. The frame moves over a given image in small strides. Each convolutional layer consists of filter matrices that consist of weight values. In the first layer, convolutional operations (dot products) are performed between the current input frame and filter weights followed by the ReLU activation function. The pooling layer is responsible for down sampling the output from a convolution+ReLU unit, thereby reducing the computational requirements by the next set of convolution layers. The final classification is performed using a set of fully connected layers that often utilize a SoftMax activation function to calculate the probability distributions for various classes. In the testing phase of a CNN model, the class with the highest probability is the output prediction. Further details on the design of CNNs can be found in [20] and [48].

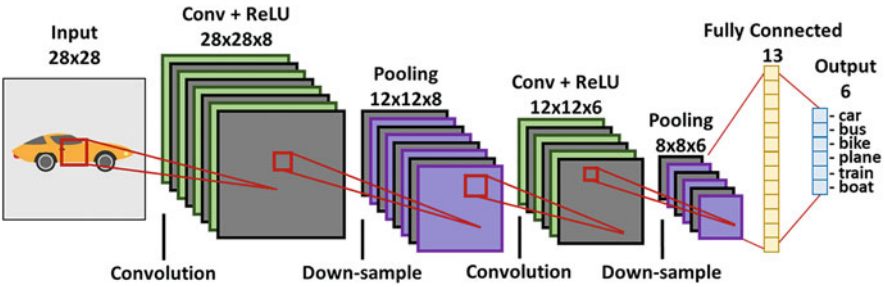


Fig. 3 A general representation of the various components of a convolutional neural network (CNN)

3.2 Indoor Localization with CNNLOC

The CNNLOC indoor localization framework [20] consists of two stages in the offline phase. The first stage comprises of capturing RSSI fingerprints as vectors across various RPs. Each vector is then reformulated as an image, such that each RSSI fingerprint image has an associated RP. The second component of the offline phase is the training of a CNN model using the images created previously. In the online phase, the same process is used to create an image (based on observed RSSI values), which is fed into the trained CNN model for location prediction.

A simplified overview of the process of converting an RSSI fingerprint vector into an image is shown in Fig. 4. The RSSI vector consists of RSSI values in the range of −100–0 dBm (low signal strength to high signal strength). These values are normalized to a range of 0–255, which corresponds to the pixel intensity on the image. The dimensions of the RSSI image are set to be the closest square to the number of visible APs on the path. For example, in Fig. 4, the RSSI vector has a size of 8, and the closest square would have 9 pixels in it; therefore, the dimensions of the image are set to 3×3. A pixel with zero intensity is padded at the end to increase the size of the vector as shown in Fig. 4. The generated image then becomes a part of the offline database of images used to train a CNN. In the online phase, this same process of image creation is used with the RSSI vector observed by the user at any location, and the resulting image is fed to the trained CNN model to get a location prediction. It is important to note that in the online phase of CNNLOC, the input image will always remain the same size as in the offline phase, such that each pixel in the image corresponds to specific MAC IDs. In case a specific MAC ID observed in the offline phase is no longer visible in the online phase, the pixel value corresponding to that MAC ID is set to zero.

4 Localization Security Analysis

In this section, we perform an AP RSSI vulnerability analysis on the deep learning-based indoor localization frameworks presented in [20] (CNNLOC) and [25] (which

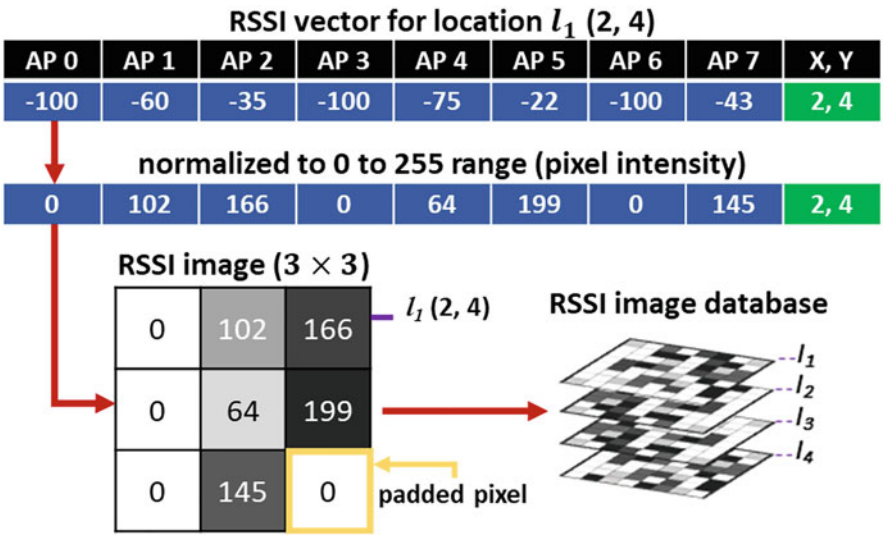


Fig. 4 A simplified overview of the conversion of an RSSI fingerprint to an image in the CNNLOC [20] indoor localization framework

uses DNNs). To achieve this, we model the impact of the insertion of malicious APs within the vicinity of two indoor paths as shown in Fig. 5.

As presented in Fig. 5, the Office and Glover paths in the figure are 64 and 88 meters long, and the reference locations used to capture Wi-Fi RSSI are marked by blue dots. A detailed discussion on the salient features of these and other indoor benchmark paths we consider can be found in the experiments section (Sect. 7). We used an HTC U11 smartphone [49] to capture Wi-Fi fingerprints along the indoor paths and test for localization accuracy.

An AP-based security attack may include either AP spoofing or AP jamming. To establish the impact of such AP-based attacks on localization accuracy, we must first identify the behavior of the Wi-Fi RSSI fingerprints in the presence of one or more malicious AP nodes (Wi-Fi spoofers/jammers). In our experience, the tainted fingerprint in the online phase will exhibit one of three behaviors: (1) the RSSI values from one or more visible Wi-Fi APs exhibits a significant increase or decrease as compared with its offline counterpart, (2) an AP whose RSSI value is usually not visible at the current reference point becomes visible, and (3) an AP that is usually visible at the current reference point is no longer visible. As the range of received RSSI values from Wi-Fi APs is between -100 and 0 dBm, the impact of the malicious Wi-Fi AP's behavior on the fingerprints is to induce fluctuations in RSSI values within this range, for the impacted fingerprints.

Figure 6 shows the fingerprint images generated using an RSSI fingerprint based on the methodology described in CNNLOC [20]. Each image has a resolution of 9×9. The original RSSI vector (fingerprint) consists of 78 Wi-Fi AP values and is presented in its image form in Fig. 6a. This image (Fig. 6a) is not tainted by

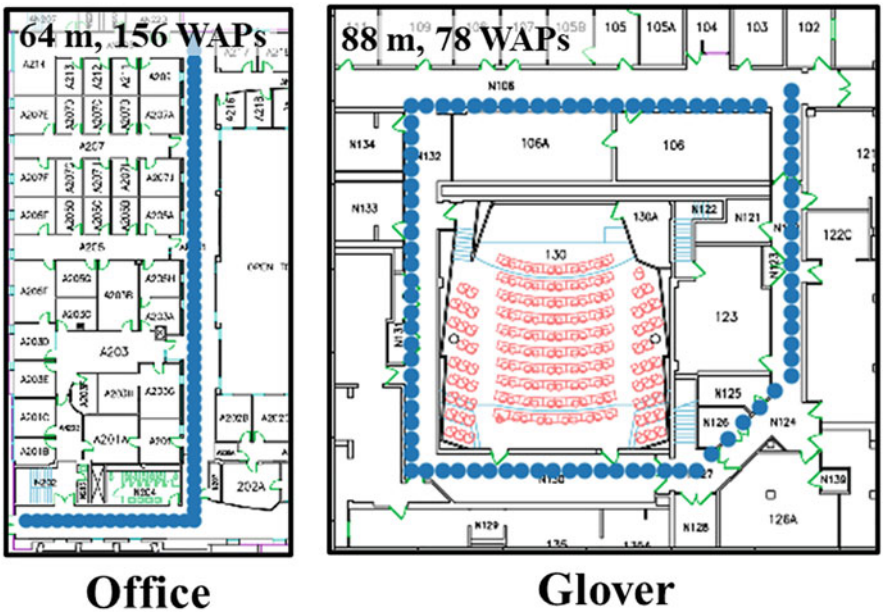


Fig. 5 Two indoor benchmark paths (Glover and Office) with reference points denoted by blue markers. The path lengths and Wi-Fi densities are denoted at the top of the maps

malicious APs (mAPs) in the surrounding area and therefore is labeled as "mAP0." The image labeled "mAP2" (Fig. 6b) is generated for the case when two APs out of 78 are malicious APs that generate spurious signals between −100 dBm and 0 dBm (their impact can be clearly seen with the two non-white pixels on the bottom half of the image). Similarly, Fig. 6c–f show the generated images when the number of malicious APs is increased to 4, 6, 8, and 10, respectively. For most of these images, the tainted pixel values can be visually identified, and simple image local smoothing filters [50] may be applied to remove them. However, such filtering is not always possible. For instance, in Fig. 6d with six malicious APs, we observe only five tainted pixels that are visually decipherable as compared with the untainted image (Fig. 6a). This is because the sixth noisy pixel is a very minor disturbance that is hard to detect visually. Unfortunately, the datapoint represented by this sixth pixel can have a significant impact on localization accuracy. Such scenarios also exist for the case of mAP8 (Fig. 6e) and mAP10 (Fig. 6f).

To test the vulnerability of deep learning-based indoor localization frameworks in the presence of malicious APs, we analyzed the impact of a varying number of malicious APs on the localization accuracy of a CNN-based [20] and a DNN-based [25] indoor localization framework. The results of this experiment are shown in Fig. 7. The impact of an increasing number of malicious Wi-Fi APs on the average indoor localization error for the two paths presented in Fig. 5 (Office and Glover) is evaluated. For a scenario with malicious APs (e.g., mAP = 1), we randomly

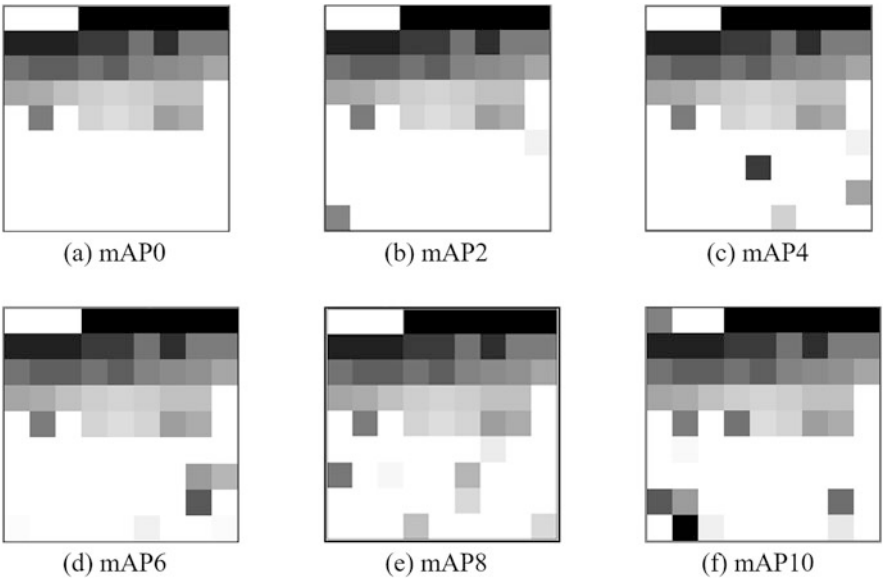


Fig. 6 Fingerprint images generated from RSSI vectors using the methodology described in CNNLOC [20]: (**a**) represents the "mAP0" fingerprint image that should be ideally generated when the initial RSSI vector is not tainted by a malicious Wi-Fi AP (mAP = 0) and (**b**–**f**) show fingerprint images in the presence of different number of malicious APs. The label "mAP*X*" indicates *X* malicious APs, which introduce fluctuations in the RSSI values of the pixels corresponding to these APs

selected the location of the malicious AP over a 100 trials and averaged the resulting localization error. From Fig. 7, we observe that the average localization error of both CNN and DNN learning models increases monotonically in a majority of cases. The results highlight the vulnerability of deep neural network-based indoor localization models toward Wi-Fi AP-based attacks. Also, the CNN model for both paths is somewhat more vulnerable to malicious AP-based attacks as compared with the DNN model. One possible explanation for this may be that CNN models are more sensitive to changes in patterns in the image as compared with variations across RSSI value inputs for the DNN model.

To further quantify the upper bounds of localization degradation of these machine learning models, we evaluate the worst-case localization error for the two deep learning models and present our findings in Fig. 8. We can observe that the worst-case localization errors for DNN and CNN models are significantly higher than the average errors shown in Fig. 7 as the number of malicious APs are increased. With only one malicious AP, the localization error in the worst case can be higher by up to 20× for both paths and deep learning models. The worst-case localization error for the CNN model goes above 50 meters with only six malicious APs for the Glover path, which would put a user's predicted location at a completely different

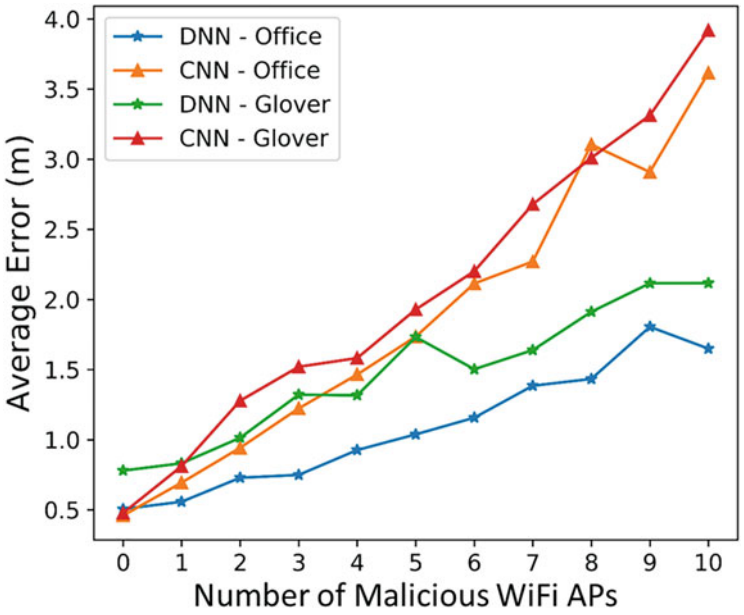


Fig. 7 Results for the impact of malicious APs on deep learning model accuracy on the Office and Glover paths. Average localization error for the CNN [20] and DNN [25] localization frameworks is shown for an increasing number of malicious AP

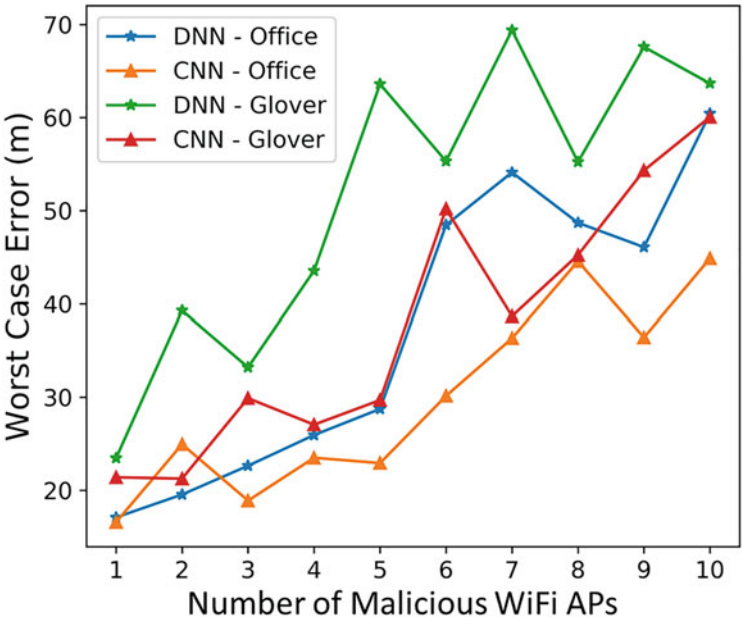


Fig. 8 Worst-case localization error for CNN and DNN, with respect to increasing number of malicious Wi-Fi APs on the Office and Glover paths

area on an indoor floor plan! The DNN model appears to be much more significantly impacted than the CNN model when it comes to worst case localization error.

The experiments performed in this section and the results as presented in Figs. 7 and 8 provide incontrovertible evidence to the fact that deep learning-based indoor localization frameworks are highly vulnerable to malicious AP-based attacks. Thus, there is strong motivation to improve attack resilience for these frameworks, to achieve both robust and high accuracy indoor localization. Even though DNN- and CNN-based models used for our experiments in this section produce a relatively similar level of degradation in localization accuracy, in the rest of this chapter, we focus on addressing vulnerabilities for indoor localization systems that utilize CNN models. This is because CNNs have several advantages over DNNs when used for localization. A drawback of DNN models is that their computational complexity increases significantly with increase in hidden layers, which is not the case for CNN models [51]. The pooling layers in CNN models reduce the overall footprint after each convolutional layer, thereby reducing the computation required by the successive set of layers. Therefore, localization solutions that utilize CNN models instead of DNN models are inherently more scalable and energy efficient [48]. Also, CNN models are better at identifying patterns in image data than DNNs, which make CNNs a more viable solution to overcome device heterogeneity issues (that are more readily apparent in image form) with indoor localization when using mobile devices [52].

The new observations and related discussions in this section highlight the importance of securing deep learning models against AP-based attacks and serve as the motivation for our proposed security enhancements in this work, which aim to secure deep learning models used for indoor localization. We discuss the specific attack models and associated assumptions made in our work in the next section.

5 Problem Formulation

We now describe our problem objective and the assumptions associated with establishing a secure (AP RSSI attack resilient) CNN-based indoor localization framework called secure-CNNLOC (S-CNNLOC) as originally presented in our work [27]. The assumptions for our framework are as follows:

- The offline fingerprint sampling process is carried out in a secure manner such that the collected fingerprints only consist of trusted nonmalicious APs.
- The offline generated fingerprint database is composed of images, each with a tagged reference point location; this database is stored at a secure, undisclosed location.
- A CNN model is trained using the offline fingerprint database and is encrypted and packaged as a part of an indoor localization app that is deployed on mobile devices.

- Once the localization app is installed by a user, the CNN model can only be accessed by that app.
- As the user moves about an indoor path, their mobile device conducts periodic Wi-Fi scans, and the localization app translates the captured Wi-Fi RSSI information into an image.
- The generated image is fed to the CNN model within the localization app on the mobile device, and the user's location is updated in real time on a map displayed on the device.
- The process of Wi-Fi scanning, fingerprint to image conversion, and location prediction continues until the user quits the localization app on their mobile device.
- We make the following assumptions about the indoor environment.
- An attacker can physically access one or more of the indoor locales and paths in the online phase for which the indoor localization framework has been trained and set-up.
- The attacker can carry a smartphone equipped with Wi-Fi or any other portable battery powered Wi-Fi transceiver to capture data about Wi-Fi access points.
- The offline generated fingerprint database is secured and cannot be accessed by any malicious third party.
- It is generally known (to the attacker) that the indoor localization framework utilizes a deep learning-based approach, such as CNNs, to predict a user's location.
- The attacker is capable of conducting the analysis described in the previous section and place malicious AP nodes at any randomly chosen locations along the indoor paths or locales that are being targeted for a service disruption attack.
- The attacker can walk about an indoor path and collect Wi-Fi fingerprints while capturing steps taken and walking direction data, similar to the approach described in [53]; this would allow anyone with a smartphone to create their own fingerprint database, which can be used to place Wi-Fi jammers more strategically or spoofed APs as discussed in earlier sections.

Problem Objective Given the above assumptions, our objective is to create a secure CNN-based indoor localization framework (called S-CNNLOC) that is deployed on mobile devices and is resilient to malicious AP RSSI attacks, by minimizing their impact on the localization accuracy at run time (i.e., in the online phase).

6 S-CNNLOC Framework

In this section, we discuss the design of our S-CNNLOC framework [27] to overcome the vulnerability of indoor localization frameworks such as CNNLOC [20] against malicious AP-based jamming and spoofing attacks in indoor environments. We mainly consider the case of malicious Wi-Fi APs as in CNNLOC [20]; however,

given the generality of our approach, it can be extended to other radio technologies and indoor localization infrastructure.

6.1 Offline Fingerprint Database Extrapolation

One of the major limitations of the CNNLOC framework comes from the small number of offline fingerprints considered per reference point (ten fingerprints in [20]). In general, deep learning models often require a large number of samples per class to produce good results. However, capturing Wi-Fi fingerprints in any indoor localization framework is a time-consuming manual endeavor that is quite expensive to scale in volume (in terms of samples per reference point).

To overcome the limited availability of fingerprints captured at each RP, we propose the extrapolation of the offline fingerprint database to achieve a larger number of fingerprint samples per RP. An overview of this process is presented in Fig. 9a. We sample a total of S RSSI fingerprints at each location (reference point) from L_1 to L_P, such that the RSSI vector has K APs (i.e., vector size is K). The complete set of fingerprints that are manually collected at P locations become the offline fingerprint database. The distribution of each AP RSSI at a given location is modeled by their means and variances. This step is repeated for each reference point in the offline fingerprint database. The mean and standard deviations along with the reference location information are temporarily stored in tabular forms and are referred to as the seed tables (Fig. 9a). The seed tables can be represented as:

$$\mu_{S(i,j)}, \sigma^2_{S(i,j)}, \quad i \in [1, K], j \in [1, P] \tag{3}$$

where $\mu_{S(i,j)}$ and the $\sigma^2_{S(i,j)}$ are the tables that contain the means and variances of S AP RSSIs for each location. These mean and variance seed tables (also shown in Fig. 9a) can now be used to extrapolate a larger fingerprint database.

To generate a new offline fingerprint for a given reference point, the normal distribution based on the mean and variance (from the seed tables) for each AP RSSI in each reference point fingerprint is randomly sampled Q times:

$$\text{RSSI}_{(i,j)} \sim N\left(\mu_{S(i,j)}, \sigma^2_{S(i,j)}\right) \forall i \in [1, K], j \in [1, P] \tag{4}$$

where RSSI*(i, j)* is the RSSI in dBm of the i^{th} Wi-Fi AP at the j^{th} reference point and N represents the normal distribution. By randomly sampling each AP from the reference point in seed tables, we generate Q new RSSI fingerprint vectors for the given reference point. Through this random sampling-based data extrapolation approach, we capture different combinations of RSSI values in a fingerprint and also scale the size of our offline dataset beyond the few samples that were collected in the offline phase. The complete set of Q RSSI vector fingerprints per reference

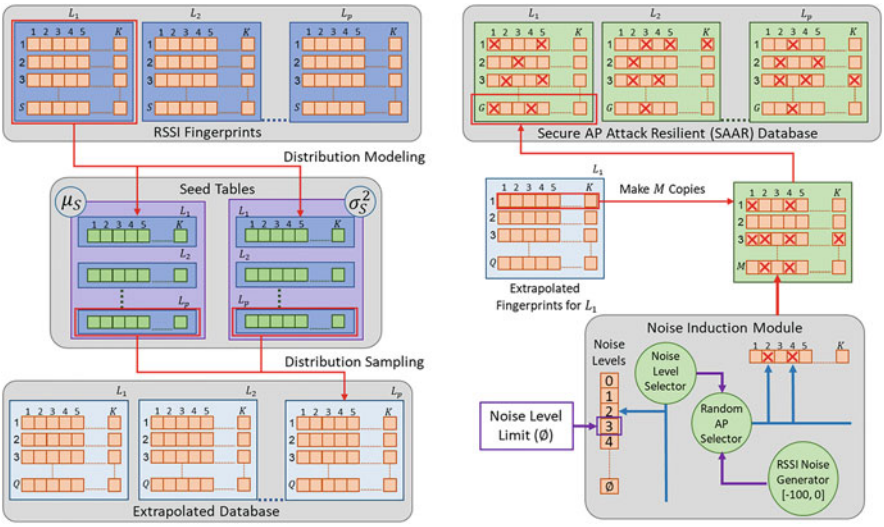


Fig. 9 An overview of the offline extrapolation of RSSI fingerprints and noise induction in the extrapolated fingerprints. The noisy and extrapolated set of RSSI fingerprints are converted into images and used to train the CNN model in our proposed S-CNNLOC framework

point is the extrapolated fingerprint database, as shown in Fig. 9a. Subsequently, we deliberately induce noise in the fingerprints in the database of extrapolated fingerprints, as discussed next.

6.2 *Inducing Malicious Behavior*

From our analysis of CNN-based indoor localization in Sect. 4, we observed that fluctuations in one individual pixel value of the Wi-Fi fingerprint image can lead to significant deterioration in the localization accuracy. This behavior can be attributed to the fact that the trained CNN model is only good at making predictions for images (or RSSI information) that it has previously seen. Therefore, the CNNLOC framework becomes vulnerable to minor deviations or noise in the images that can be induced by AP-based attacks or Wi-Fi jammer attacks in the online phase, when the trained CNN model is used for location inference.

Convolutional layers and by extension CNN models are designed to recognize one or more unique patterns within images that are not as obvious to other machine learning algorithms. In our approach, we conjecture that relatively small-scale variations within and between images constructed from AP RSSI values (for the purpose of pattern recognition for indoor localization) can be learned to be ignored by a CNN model. One way to accomplish this is by integrating an image filter with the CNN prediction model. A recent work [54] has shown how a salt and

pepper noise filtering technique can provide some noise resilience for general image processing with CNNs. A separate set of convolutional layers are used in [54] whose sole purpose is to denoise an image. However, such an approach would be extremely inefficient for our problem as it would require using two different CNNs: one for denoising and another for classification, which would increase prediction time. Moreover, using an additional CNN would increase the memory footprint of our framework, which is a big concern for resource-constrained mobile devices.

We propose to use a single CNN model for both image denoising and classification. Based on our analysis presented in Sect. 4, we decide to conceptually model malicious behaviors such as AP spoofing, AP jamming, and even environmental changes as random fluctuations in the fingerprint data and expect the CNN model to be resilient to such fluctuations. Thus, by a calculated introduction of noise in the input dataset that is used in the training phase of the CNN model, we hope to teach the model to learn to ignore noise (due to malicious APs) in the inference phase. Toward this goal, as shown in Fig. 9b, for each fingerprint in the "clean" (mAP0) extrapolated database generated as discussed in the previous subsection, M copies are constructed in a separate table. Then each of the M fingerprint vectors are fed to the proposed noise induction module that introduces random fluctuations in the AP RSSI values, based on an upper limit (Ø) that is set by the user. The noise induction module (Fig. 9b) has three major components. For a given RSSI vector, the noise level selector submodule picks values from a discrete uniform distribution such that $\theta \sim U\{0, \varnothing\}$, where "$\theta$" is the number of APs in the RSSI vector whose RSSI value would be altered by the noise induction module. The random AP selector arbitrarily identifies the set of AP candidates "W_θ," where each AP candidate "w_c" is picked to be between 1 and K as described by the expression:

$$w_c \sim U\{1, K\}, \quad c \in [1, \theta] \tag{5}$$

$$s.t., W_\theta = \{w_1, w_2, w_3 \ldots w_\theta\}$$

The newly generated RSSI vectors ($\text{RSSI}^{\text{Noisy}}_{(i,j)}$) are tainted by random noise at the i^{th} Wi-Fi AP position, if the AP was chosen by the random AP selector submodule as shown by Eq. (6):

$$\text{RSSI}^{\text{Noisy}}_{(i,j)} = \begin{cases} I, & \text{if } i \in W_\theta \\ \text{RSSI}_{(i,j)}, & \text{otherwise} \end{cases} \tag{6}$$

$$j \in [1, P], I \sim U\{-100, 0\}$$

where I represents noise sampled from a discrete uniform distribution between −100 dBm and 0 dBm, RSSI*(i, j)* is the clean (untainted) RSSI from Eq. (4), and

P is the number of reference points on a benchmark path for which fingerprint data has been collected. Thus, our proposed approach generates RSSI vectors that may have up to Ø noise induced RSSI AP values. Having a uniform distribution of 0 to Ø malicious APs ensures that the CNN model trained using the generated data is resilient to a range of malicious AP numbers and locations in the localization environment in the testing phase.

Following this process for all fingerprints in the clean training database, we generate $G = Q \times M$ fingerprints per reference point. The final number of RSSI fingerprints in the secure AP attack resilient (SAAR) database constructed by following the processes described in this section is $G \times P$, where P is the number of reference points on a benchmark path. The indoor localization app that is subsequently deployed on a mobile device consists of the CNN model that is trained using the newly constructed SAAR fingerprint database. The user carrying the mobile device will be able to securely localize themselves in real time.

7 Experiments

7.1 Experimental Setup

We initially compare the accuracy and stability of our proposed (S-CNNLOC) framework to its vulnerable counterpart (CNNLOC [20]) using two benchmark paths. These paths are shown in Fig. 5 with each fingerprinted location (reference point) denoted by a blue marker. The paths were selected due to their salient features that may impact location accuracy in different ways. The 64-meter Office path is on the second floor of a relatively recently designed building with a heavy use of wood, plastics, and sheet metal as construction materials. The area is surrounded by small offices and has a total of 156 Wi-Fi APs visible along the path. The Glover path is from a very old building with materials such as wood and concrete used for its construction. This 88-meter path has a total of 78 visible Wi-Fi APs and is surrounded by a combination of labs (heavy metallic equipment) and classrooms with open areas (large concentration of users).

In the offline phase of S-CNNLOC, an HTC U11 smartphone is utilized to capture 10 Wi-Fi fingerprints per reference point. On a given indoor path (Fig. 5), each reference point is 1-meter apart; therefore, the user can best localize themselves at a granularity of 1-meter in the online phase.

The fingerprint sampling and storage methodology within the smartphone is similar to that described in CNNLOC [20]. The trained S-CNNLOC model was deployed as an Android app on the HTC U11 smartphone. The values of Q and M (discussed in Sect. 6) are set to 100 and 10, respectively. Based on these values of Q and M, the Office path has 64,000 samples and the Glover path has 88,000 samples. To study the impact of malicious Wi-Fi APs on indoor localization performance, we used a real Wi-Fi transceiver [55] to induce interference (from spoofing/jamming)

and obtain "tainted" RSSI values in the vicinity of the indoor paths. These values were observed in the online phase. For some of our scalability studies where we consider the impact of multiple malicious APs, multiple such transceivers were considered, to generate multiple "tainted" RSSI values.

7.2 Experimental Results

7.2.1 Analysis of Noise Induction Aggressiveness

We first performed a sensitivity analysis on the value of Ø (upper limit of noise induction; discussed in Sect. 6.2). Several CNN models were trained: S-CNNLOC1 (Ø = 0; no malicious APs), S-CNNLOC2 (Ø = 1), and up to S-CNNLOC20 (Ø = 20), using the fingerprint data collected during the offline phase. Then the devised models were tested with fingerprints observed along the indoor paths in the online phase, in the presence of different numbers of malicious APs.

Figure 10 shows the heat map for the mean localization errors (in meters) with annotated standard deviation of various scenarios on the Office path (Fig. 10a) and the Glover path (Fig. 10b). The y-axis shows various S-CNNLOC variants with different values of Ø varying from 1 to 20. The x-axis shows the number of malicious nodes (mAPs) present in the online phase. In Fig. 10, the bright yellow cells of the heat map, with higher annotated values, represent an unstable and degraded localization accuracy whereas the darker purple cells, with lower annotated values, represent stable and higher levels of localization accuracy. Each row of pixels in the heat maps of Fig. 10a, b represents the vulnerability of the specific S-CNNLOC model to an increasing number of mAP nodes.

On both paths (Office and Glover), it can be clearly observed that the secure-CNNLOC0 model (baseline model with Ø = 0) is the least resilient to an increasing number of mAPs. However, as the value of Ø is increased for the S-CNNLOC models, they perform significantly better than S-CNNLOC0 (as illustrated by the darker rows for these models). This is because the S-CNNLOC0 model is not trained to mitigate variations for Wi-Fi AP RSSI values. Another observation is that beyond Ø = 18, the standard deviation and mean error for low values of malicious APs (mAPs < 4) starts increasing for both paths. This is because highly noisy images in the SAAR database are unable to retain the original pattern required to localize in safer environments (no malicious APs) or the opted CNNLOC model is unable to recognize underlying patterns in the input fingerprint images.

Overall, we observe that training the S-CNNLOC models with fingerprint extrapolation and noise induction (via the generated SAAR database) leads to better localization accuracy. Based on the results of these experiments, we found that S-CNNLOC18 delivers good results across both paths. Therefore, we use the value of Ø = 18 in SAAR to train S-CNNLOC and use it for the rest of our experiments. Henceforth, whenever we refer to S-CNNLOC, we are referring to S-CNNLOC18 (S-CNNLOC with Ø = 18).

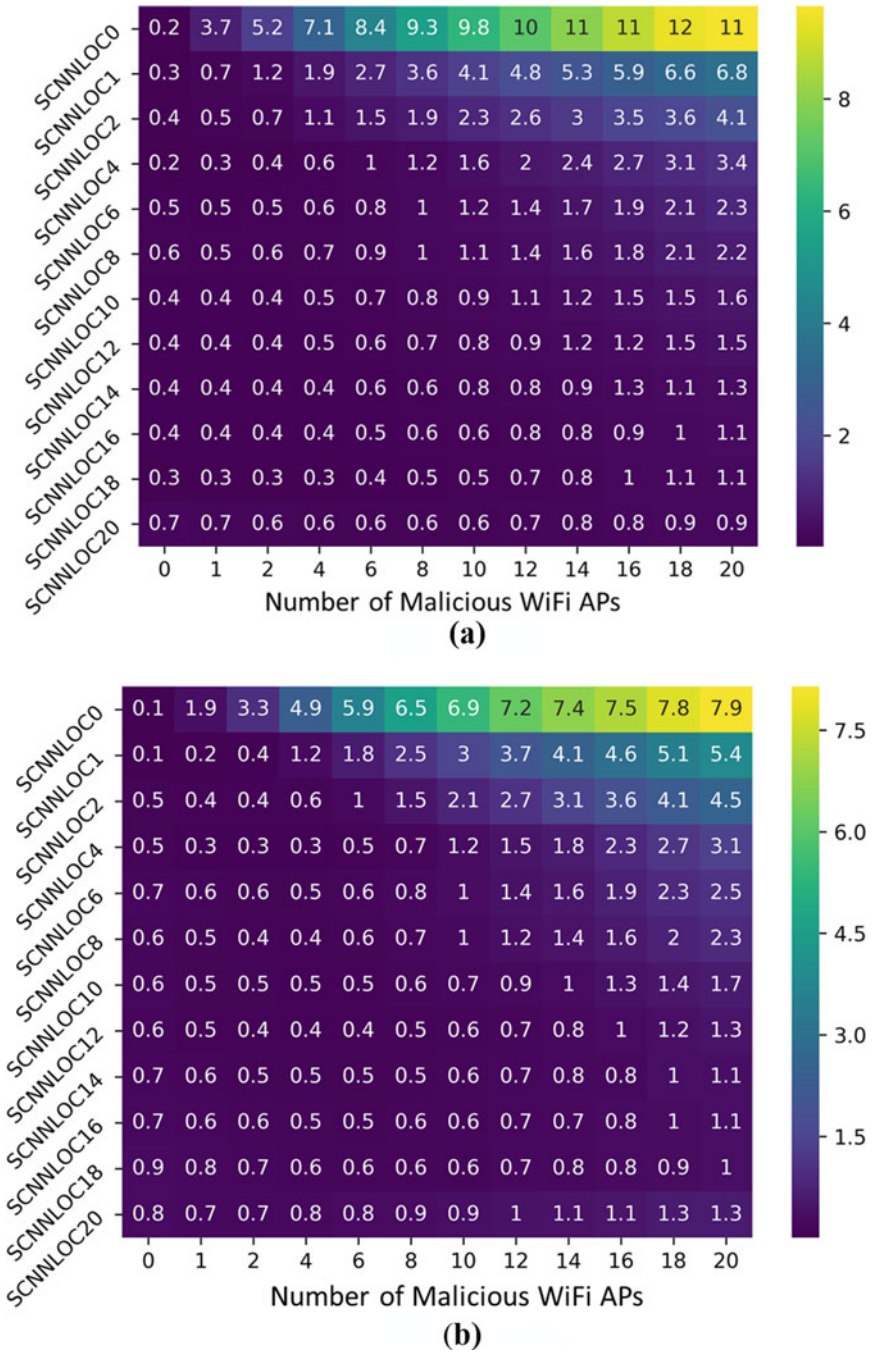


Fig. 10 Heat maps for the mean localization prediction errors with their annotated standard deviation for the Office (top) and Glover (bottom) benchmark paths; results are shown for our proposed S-CNNLOC framework with $\emptyset = 0, \emptyset = 1, \ldots \emptyset = 20$ (y-axis). (**a**) Office. (**b**) Glover

7.2.2 Comparison of Attack Vulnerability

In this section, we contrast the performance of our proposed S-CNNLOC framework with CNNLOC [20]. Figure 11a, b show the cumulative distribution function (CDF) of the localization error for the CNNLOC models in the presence of different numbers of malicious Wi-Fi APs (from 0 to 20 malicious APs per observed fingerprint), for the Office and Glover paths. The most immediate observation from the results is that the localization errors are significantly low (less than 1 meter for a majority of scenarios) when there are no malicious APs (CNNLOC-mAP0), However, in both the Office (Fig. 11a) and the Glover paths (Fig. 11b), localization accuracy degrades as the number of malicious APs is increased. This degradation in accuracy does not scale linearly with increasing malicious nodes. For example, in the Office path, increasing the malicious AP nodes from 16 to 20 does not significantly increase the localization errors. A similar observation can be made from the Glover path in Fig. 11b, where the localization error does not scale by much when going from 12 malicious APs to 16 and 20.

An important aspect to note from looking at Fig. 11 is the significant drop in localization accuracy when going from a scenario with no malicious APs (CNNLOC-mAP0) to a scenario with one malicious AP (CNNLOC-mAP1). This degradation of localization quality is a clear indicator of the vulnerabilities associated with the employment of unsecured CNN models in the presence of even a single malicious Wi-Fi node.

From Fig. 11, we can conclude that a malicious third party can significantly degrade the localization accuracy of a CNN-based indoor localization model such as CNNLOC [20], with just a very small number of malicious AP nodes.

Figure 12 highlights the resiliency of the S-CNNLOC model toward malicious AP-based attacks, for the same setup as for the experiment with CNNLOC in Fig. 11, where the number of malicious APs in the online phase is varied from 0 to 20. We observe that 95-percentile of the localization error for the S-CNNLOC model, when under attack by up to 20 malicious AP nodes (S-CNNLOC-mAP20), remains under 2.5 meters for the Office path (Fig. 12a) and under 3.5 meters for the Glover path (Fig. 12b). The S-CNNLOC model for the Office path performs better than for the Glover path as the Wi-Fi density on the Office path is about 2× the Wi-Fi density of the Glover path, and thus, malicious APs only impact a small fraction of the total APs along the Office path.

In summary, based on the results shown in Figs. 11 and 12, we observe that our S-CNNLOC framework is about 10× more resilient to accuracy degradation in the average case, as compared with its unsecure counterpart CNNLOC [20], for the Office and Glover paths.

7.2.3 Extended Analysis on Additional Benchmark Paths

We conducted further experimental analysis on a more diverse set of benchmark indoor paths. Table 1 presents the salient features of the three new benchmark paths

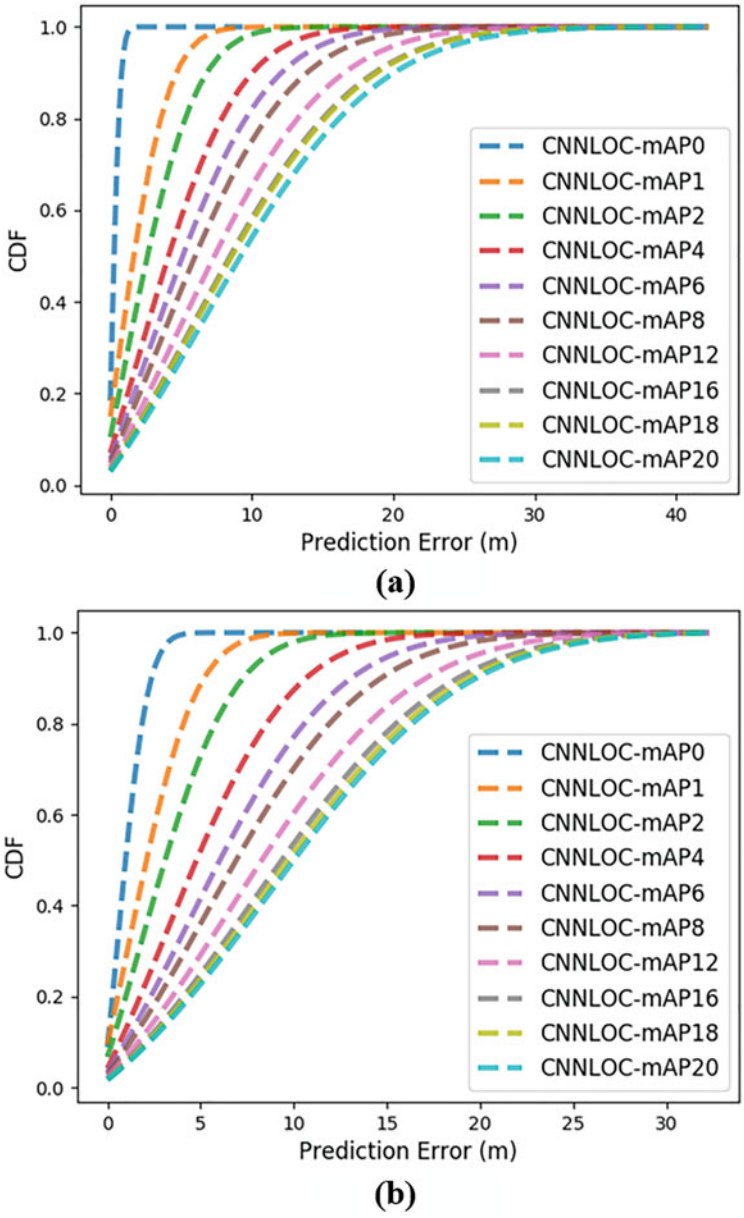


Fig. 11 Localization performance of CNNLOC [20] with a varying number of malicious Wi-Fi APs (from 0 to 20) in the online phase. (**a**) Office. (**b**) Glover

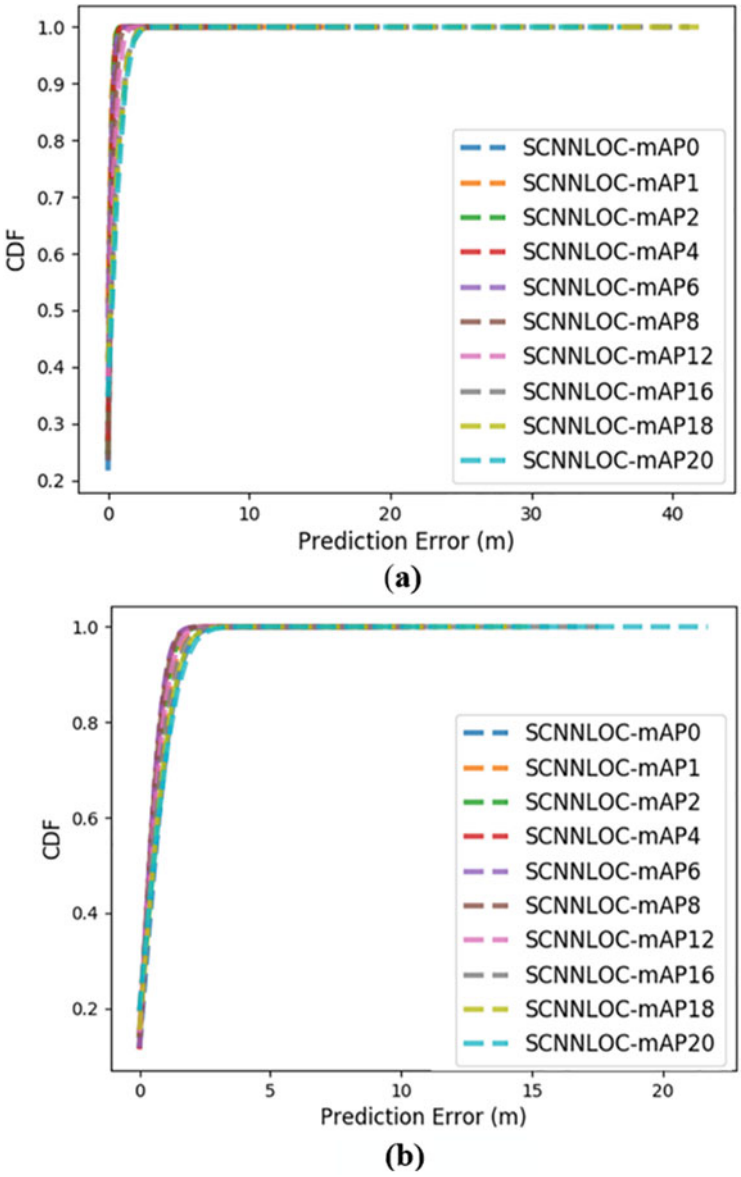


Fig. 12 Localization performance of our S-CNNLOC with a varying number of malicious Wi-Fi APs (from 0 to 20) in the online phase (**a**) Office. (**b**) Glover

used in this analysis. The benchmark path suite shown in Table 1 consists of the EngrLabs, LibStudy, and the Sciences paths, with a description of environmental factors that may affect the localization performance of Wi-Fi-based indoor localization frameworks. Each path has a length ranging from 58 to 68 meters, and ten Wi-Fi

Table 1 Additional benchmark paths and their features

Path name	Length (meter)	Number of APs	Environmental features
EngrLabs	62	120	Electronics, concrete, labs
LibStudy	68	300	Wood, metal, open area
Sciences	58	130	Metal, classrooms
Office	64	156	Wood, concrete
Glover	88	78	Wood, metal, concrete

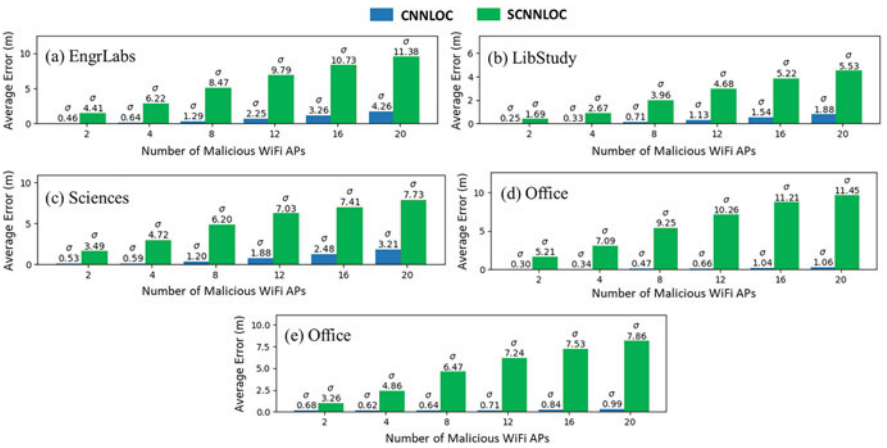


Fig. 13 The average localization error and its standard deviation of the proposed S-CNNLOC framework as compared with CNNLOC [20] for the benchmark path suite from Table 1

fingerprint samples were collected at 1-meter intervals on each path, similar to what we did with the Office and Glover paths described earlier. The EngrLabs path is in an old building mostly made of concrete and is surrounded by labs consisting of heavy metallic instruments. The LibStudy and Sciences paths are situated in relatively newer buildings consisting of large amounts of metallic structures. The LibStudy path is a part of an open area in the library building and exhibits considerable human movements. Similar to it, the Sciences path is the close vicinity of a classroom.

Figure 13 presents the means and standard deviations of the localization error with our proposed S-CNNLOC and the CNNLOC [20] framework on each of the three paths while it is under the influence of 2–20 malicious APs in the online phase. We observe an increasing trend in mean and standard deviations of localization errors on all three paths for both S-CNNLOC and CNNLOC. However, we observed that the mean localization error of CNNLOC on all three paths is always more than four times the average error for S-CNNLOC. For some situations, such as for two and four malicious APs on the EngrLabs and Sciences paths, the localization error for CNNLOC is about 25× higher (worse) on average as compared with its S-CNNLOC counterpart. The accuracy along the Libstudy path is relatively less affected than for the other paths. This can again be attributed to the fact that the

LibStudy path has an unusually dense Wi-Fi network compared with the EngrLabs and Sciences paths, and thus, a relatively fewer number of malicious APs do not have as much of an impact on accuracy. Another contributing factor could be that the LibStudy path is an open area and localization process is not heavily impacted by multipath and shadowing effects. These experiments with additional benchmark paths indicate that our proposed S-CNNLOC framework scales well over a wide variety of indoor paths with different environmental features whereas the unsecured CNNLOC [20] framework experiences a significant degradation in its localization error. The S-CNNLOC model consistently reduces the vulnerability of the proposed localization framework and thus represents a promising solution to secure deep learning-based indoor localization frameworks.

7.2.4 Generality of Proposed Approach

In this section, we highlight the generality and the versatile nature of our proposed security aware approach by applying it to another deep learning-based approach proposed in [25]. We first present a discussion of the proposed work in [25]. Later, we use Wi-Fi fingerprints generated in Sect. 7.1 to train the secure-DNN (SDNN) model and compare its prediction accuracy results to the conventional methodology described in [25].

7.2.5 Denoising Autoencoder-Based DNN Framework

The DNN-based approach in [25] consists of three stages in the online phase. In the first stage, features are extracted from the RSSI fingerprints using a stacked denoising autoencoder (SDA). The SDA's output is fed to a four-layer DNN model in the second stage that delivers a coarse location prediction. In the final stage, additional hidden Markov model (HMM) is used to fine-tune the coarse localization perdition received from the DNN model.

The DNN model in conjunction with the SDA is able to identify and learn stable and reliable features from the input fingerprint information. Intuitively, SDA achieves this by zeroing-out input features based on a predefined probability and identifying input features that have a significant impact on the output. Further, the HMM allows for greater resistance to minor variations in AP RSSI over time.

7.2.6 Security Aware DNN Training in the Offline Phase

To train the SDNN model, we use the augmented security aware fingerprints used to train the SCNNLOC model in the previous section. The only difference being that the fingerprints are not converted into images. To identify the stable value of Ø for noise induction module, we perform a sensitivity analysis using DNN models as done in Sect. 7.2.1. The results for this experiment are captured in Fig. 14.

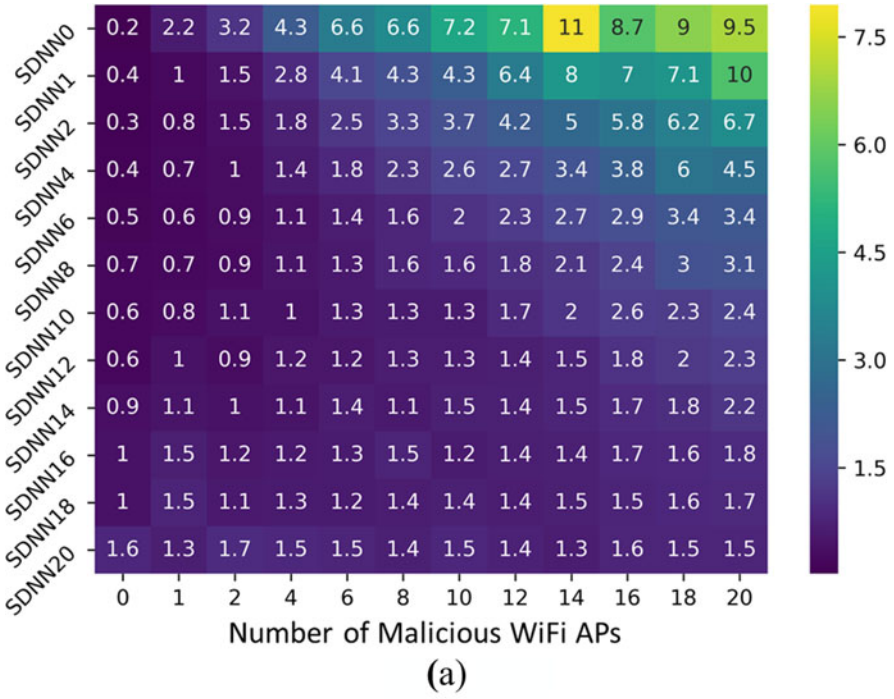


Fig. 14 Heat maps for the mean localization prediction errors with their annotated standard deviation for the Office (top) and Glover (bottom) benchmark paths; results are shown for our proposed S-DNN framework with $\text{Ø} = 0, \text{Ø} = 2, \ldots \text{Ø} = 20$ (y-axis). (**a**) Office. (**b**) Glover

In Fig. 14, we observe that the mean localization errors for the baseline SDNN0 models for the Office and Glover paths increase by 48× and 13× in the presence of 20 malicious nodes, respectively. For SDNN models trained with a larger value of Ø (14, 16, 18), the localization error remains lower as the number of malicious nodes in the online phase increase. For the sake of simplicity across the rest of this chapter, we set the value of $Ø = 18$ for all paths. Beyond this point, any reference to an SDNN model refers to DNN model [25] trained with $Ø = 18$. In the next subsection, we present an extended analysis on the performance of SDNN as compared with a conventional unsecured DNN model.

Figure 15 presents an analysis on the stability of the conventional unsecured DNN-based framework [25] as compared with secure-DNN (SDNN) model in the presence of an increasing number of malicious APs on a set of versatile paths with varying environmental characteristics as discussed in Table 1. From Fig. 15, we observe the prediction accuracy of the conventional DNN-based approach presented in [25] systematically degrades (increased average error) as the number of stochastically placed malicious Wi-Fi access points on various paths is increased. The SDA stage in [25] is supposed to learn prominent features by learning to encode prominent input features (ignoring noise) in the training phase. However, the noise in the training features over a short period of time is significantly lower and different from the addition of malicious APs in the online prediction phase. The method proposed in [25] degrades with the introduction of malicious APs in the testing or online phase. This can be attributed to the fact that SDA does not learn to denoise malicious fingerprints in the training phase. Further, the HMM model is unable to stabilize the final location prediction because it is designed to improve the fine-grain location based on the assumption that the consecutive coarse-grain predictions from the DNN are sufficiently close together. However, in the presence of malicious APs, this assumption does not hold for the coarse-grain predictions causes the HMM to deliver unstable results.

On the other hand, the SDA component of the SDNN-based model learns to denoise and ignore malicious APs. This is achieved through stochastically zeroing out RSSI values, identifying stable trusted APs, and denoising malicious APs over various fingerprints. As observed for various paths in Fig. 15, this greatly improves SDNN's resilience to malicious APs in the online phase and delivers up to ten times better mean prediction accuracy such as in the case of 16 malicious APs on the EngrLabs path.

A notable aspect of our proposed approach is that it allows for the deep learning model to ignore malicious APs in the testing phase; however, the extent of resilience to the malicious AP-based attacks is dependent on the deep learning model's ability to identify underlying pattern in the training fingerprints. Deep learning models such as CNNs and SDA-based approaches are more likely to deliver promising results as they are both designed to identify underlying stable patterns in the training phase. However, designing a deep learning model that delivers the best results in all situations is beyond the scope of this work.

Through experiments performed and the discussion of presented results, we can conclude that our proposed approach delivers superior stability of prediction

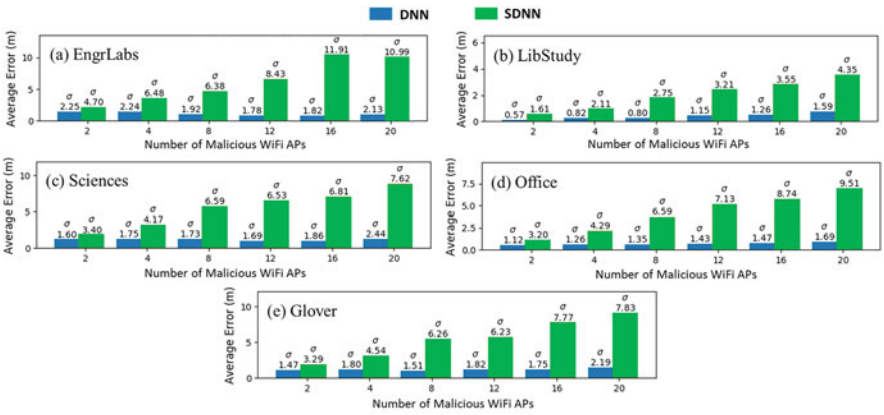


Fig. 15 The average localization error and its standard deviation of the proposed S-DNN framework as compared with DNN [25] for the benchmark path suite from Table 1

accuracy of deep learning-based models over a versatile set of benchmark paths. Furthermore, since our proposed approach of securing deep learning-based models focuses on the training dataset instead of the model design, it can be generalized to a wide variety deep learning-based indoor localization frameworks.

8 Conclusions and Future Work

In this chapter, we presented a vulnerability analysis of deep learning-based indoor localization frameworks that are deployed on mobile devices, in the presence of wireless access point (AP) spoofing and jamming attacks. Our analysis highlighted the significant degradation in localization accuracy that can be induced by an attacker with very minimal effort. For instance, our experimental studies suggest that an unsecured convolutional neural network (CNN)-based indoor localization solution can place a user up to 50 meters away from their actual location, with attacks on only a few APs. Based on our new observations, we devised a novel solution to provide resilience against such attacks and demonstrated it on a CNN-based localization framework to address its vulnerability to intentional RSSI variation-based attacks. To further highlight the generality of our proposed security aware approach, we implemented it on a deep neural network (DNN)-based indoor localization solution. Our proposed vulnerability resilient framework was shown to deliver up to ten times superior localization accuracy on average, in the presence of threats from several malicious attackers, compared with the unsecured CNN- and DNN-based localization framework.

As a part of future work, we will be focusing on improving the quality of localization and navigation. Toward this goal, a possible extension of our work

can be to predict the path taken by the user using multiple Wi-Fi fingerprints as an attack is taking place. In such situations, the machine learning model could correct a previous prediction (path taken) based on the upcoming predictions and vice versa. This methodology may improve the localization accuracy and stability in corner cases in the online phase where fingerprints at a location are similar in structure to others fingerprint that are spatially separated by large distances.

Acknowledgments This work was supported by the National Science Foundation (NSF), through grant CNS-2132385.

References

1. The Plane Crash That Gave Americans GPS, 2019 [Online]. Available: https://www.theatlantic.com/technology/archive/2014/11/the-plane-crash-that-gave-americans-gps/382204/
2. A brief history of GPS, 2019 [Online]. Available: https://www.pcworld.com/article/2000276/a-brief-history-of-gps
3. Larcom, J.A., Liu, H.: Modeling and characterization of GPS spoofing. Conference on technologies for homeland security (HST), 2013.
4. Bonebrake, C., Ross O'Neil, L.: Attacks on GPS time reliability. IEEE Secur. Privacy. **12**(3), 82–84 (2014)
5. Jansen, K., Schäfer, M., Moser, D., Lenders, V., Pöpper, C., Schmitt, J.: Crowd-GPS-Sec: leveraging crowdsourcing to detect and localize GPS spoofing attacks. Symposium on security and privacy (SP), 2018.
6. This GPS Spoofing Hack Can Really Mess with Your Google Maps Trips, 2019 [Online]. Available: https://www.forbes.com/sites/thomasbrewster/2018/07/12/google-maps-gps-hack-takes-victims-to-ghost-locations/
7. Spoofing in the Black Sea: What really happened?, 2019 [Online]. Available: https://www.gpsworld.com/spoofing-in-the-black-sea-what-really-happened
8. Langlois, C., Tiku, S., Pasricha, S.: Indoor localization with smartphones: harnessing the sensor suite in your pocket. IEEE Consum. Electron. Mag. **6**(4), 70–80 (2017)
9. WiFi RTT (IEEE 802.11mc), 2019 [online]. Available: https://www.source.android.com/devices/tech/connect/wifi-rtt
10. Top 33 indoor localization services in the US, 2019 [online]. Available: https://www.technavio.com/blog/top-33-indoor-location-based-services-lbs-companies-in-the-us
11. Corina, K., MacWilliams, A.: Overview of indoor positioning technologies for context aware AAL applications. Ambient Assisted Living, 2011.
12. Chen, Y., Sun, W., Juang, J.: Outlier detection technique for RSS-based localization problems in wireless sensor networks. SICE. (2010)
13. Khalajmehrabadi, A., Gatsis, N., Pack, D.J., Akopian, D.: A joint indoor WLAN localization and outlier detection scheme using LAS-SO and elastic-net optimization techniques. IEEE Trans. Mob. Comput. **16**(8), 2079–2092 (2017)
14. Schmitz, J., Hernández, M., Mathar, R.: Real-time in-door localization with TDOA and distributed software de-fined radio: demonstration abstract. Information Processing in Sensor Networks (IPSN). (2016)
15. Vasisht, D., Kumar, S., Katabi, D.: Sub-nanosecond time of flight on commercial WiFi cards. Special Interest Group on Data Communication (SIGCOMM). (2015)
16. Chen, Z., Li, Z., Zhang, X., Zhu, G., Xu, Y., Xiong, J., Wang, X.: AWL: turning spatial aliasing from foe to friend for accurate WiFi localization. Conference on emerging Networking Experiments and Technologies (CoNEXT), 2017.

17. Lu, Z., Wang, W., Wang, C.: Modeling, evaluation and detection of jamming attacks in time-critical wireless applications. IEEE Trans. Mob. Comput. **13**(8), 1746–1759 (2014)
18. Soltanaghaei, E., Kalyanaraman, A., Whitehouse, K.: Multipath triangulation: decimeter-level WiFi localization and orientation with a single unaided receiver. Mobile Systems, Applications, and Services (MobiSys). (2018)
19. Pasricha, S., Ugave, V., Anderson, C.W., Han, Q.: LearnLoc: a framework for smart indoor localization with embedded mobile devices. Hardware/Software Codesign and System Synthesis (CODES+ISSS). (2015)
20. Mittal, A., Tiku, S., Pasricha, S.: Adapting convolutional neural networks for indoor localization with smart mobile devices. Great Lakes symposium on VLSI (GLSVLSI), 2018.
21. Tiku, S., Pasricha, S.: Siamese neural encoders for long-term indoor localization with mobile devices. IEEE/ACM design, automation and test in Europe (DATE) conference and exhibition, 2022.
22. Wang, L., Tiku, S., Pasricha, S.: CHISEL: compression-aware high-accuracy embedded indoor localization with deep learning. IEEE Embedd. Syst. Lett. **14**(1), 23–26 (2021)
23. Tiku, S., Kale, P., Pasricha, S.: QuickLoc: adaptive deep-learning for fast indoor localization with mobile devices. ACM Trans. Cyber-Phys. Syst. **5**(4), 1–30 (2021)
24. Meng, W., Xiao, W., Ni, W., Xie, L.: Secure and robust WiFi finger-printing indoor localization. Indoor Positioning and Indoor Navigation (IPIN), 2011.
25. Zhang, W., et al.: Deep Neural Networks for wireless localization in indoor and outdoor environments. Neurocomputing. **194**, 279–287 (2016)
26. Cheng, Y.K., Chou, H.J., Chang, R.Y.: Machine-learning indoor localization with access point selection and signal strength reconstruction. Vehicular technology conference (VTC), 2016.
27. Tiku, S., Pasricha, S.: Overcoming security vulnerabilities in deep learning based indoor localization on mobile devices. ACM Trans. Embedd. Comput. Syst. **18**(6), 1–24 (2020)
28. Bahl, P., Padmanabhan, V.: RADAR: an in-building RF-based user location and tracking system. INFOCOM. (2000)
29. Mohammadi, M., Al-Fuqaha, A., Guizani, M., Oh, J.: Semisupervised deep reinforcement learning in support of IoT and smart city services. Internet Things J. **5**(2), 624–635 (2018)
30. Ubisense Research Network, 2017 [Online] Available: http://www.ubisense.net/
31. Dickinson, P., Cielniak, G., Szymanezyk, O., Mannion, M.: Indoor positioning of shoppers using a network of Bluetooth low energy beacons. Indoor Positioning and Indoor Navigation (IPIN), 2016.
32. Lau, S., Lin, T., Huang, T., Ng, I., Huang, P.: A measurement study of zigbee-based indoor localization systems under RF interference. Workshop on experimental evaluation and characterization (WIN-TECH), 2009.
33. Pasricha, S., Doppa, J., Chakrabarty, K., Tiku, S., Dauwe, D., Jin, S., Pande, P.: Data analytics enables energy-efficiency and robustness: from mobile to manycores, datacenters, and network. ACM/IEEE international conference on hardware/software codesign and system synthesis (CODES+ISSS), 2017.
34. Tiku, S., Pasricha, S.: Energy-efficient and robust middleware prototyping for smart mobile computing. IEEE international symposium on rapid system prototyping (RSP), 2017.
35. Zou, H., et al.: A robust indoor positioning system based on the Procrustes analysis and weighted extreme learning machine. IEEE Trans. Wireless Comput. **15**(2), 1252–1266 (2016)
36. Tiku, S., Pasricha, S.: PortLoc: a portable data-driven indoor localization framework for smartphones. IEEE Design & Test (Early Access), 2019.
37. Tiku, S., Pasricha, S., Notaros, B., Han, Q.: SHERPA: a lightweight smartphone heterogeneity resilient portable indoor localization framework. IEEE international conference on embedded software and systems (ICESS), 2019.
38. Tiku, S., Pasricha, S., Notaros, B., Han, Q.: A hidden Markov model based smartphone heterogeneity resilient portable indoor localization framework. J. Syst. Archit. **108**, 101806 (2020)
39. Chang, L., Chen, X., Wang, J., Fang, D., Liu, C., Tang, Z., Nie, W.: TaLc: time adaptive indoor localization with little cost. MobiCom workshop on challenged networks (CHANTS), 2015.

40. Barbará, D., Goel, R., Jajodia, S.: Using checksums to detect data corruption. International conference on extending database technology, 2000.
41. Ou, L., Qin, Z., Liu, Y., Yin, H., Hu, Y., Chen, H.: Multi-user location correlation protection with differential privacy. International conference on parallel and distributed systems (ICPADS), 2016.
42. Lazos, L., Krunz, M.: Selective jamming/dropping insider attacks in wireless mesh networks. IEEE Trans. Netw. **25**(1), 30–34 (2011)
43. Xu, W., Trappe, W., Zhang, Y., Wood, T.: The feasibility of launching and detecting jamming attacks in wireless networks. Mobile ad hoc Networking and Computing (MobiHoc), 2005.
44. Guerrero-Higueras, Á.M., DeCastro-García, N., Rodríguez-Lera, F.J., Matellán, V.: Empirical analysis of cyber-attacks to an indoor real time localization system for autonomous robots. Comput. Secur. **70**, 422–435 (2017)
45. Guerrero-Higueras, Á.M., Matellan, N.: Detection of cyber-attacks to indoor real time localization systems for autonomous robots. Robot. Auton. Syst. **99**, 75–83 (2018)
46. Silva, A.A.A., et al.: Predicting model for identifying the malicious activity of nodes in MANETs. Symposium on computers and communication (ISCC), 2015.
47. Wu, C., Yang, Z., Liu, Y.: Smartphones based crowdsourcing for indoor localization. IEEE Trans. Mob. Comput. **14**(2), 444–457 (2015)
48. LeCun, Y., et al.: Gradient-based learning applied to document recognition. Proc. IEEE. **86**(11), 2278–2324 (1998)
49. HTC U11, [Online]: https://www.htc.com/us/smartphones/htc-u11
50. Lee, J.S.: Digital image smoothing and the sigma filter. Computer Vision, Graphics, and Image Processing. **24**(2), 255–269 (1983)
51. Wang, X., et al.: DeepFi: deep learning for indoor fingerprinting using channel state information. Wireless communications and networking conference (WCNC). (2015)
52. Machaj, J., Brida, P., Piché, R.: Rank based fingerprinting algorithm for indoor positioning. Indoor Positioning and Indoor Navigation (IPIN), 2011.
53. Shu, Y., et al.: Gradient-based fingerprinting for indoor localization and tracking. IEEE Trans. Ind. Electron. **63**(4), 2424–2433 (2016)
54. Zhang, F., Cai, N., Wu, J., Cen, G., Wang, H., Chen, X.: Image de-noising method based on a deep convolution neural network. IET Image Process. **12**(4), 485–493 (2018)
55. MAC Address Clone on my TP-Link, [Online]: https://www.tp-link.com/us/support/faq/68/

Considering the Impact of Noise on Machine Learning Accuracy

Mahum Naseer, Iram Tariq Bhatti, Osman Hasan, and Muhammad Shafique

1 Introduction

Due to their astounding classification performance and decision-making capability in practical applications such as healthcare, smart cyber-physical systems (CPS), autonomous driving, and Internet of Things (IoTs) [7, 18, 26], there has been a continuous rise in the use of embedded machine learning (ML)-based systems in the past few decades. A major contributing factor to the success of these ML-based systems is the advancements in the underlying artificial neural networks (ANNs). However, the addition of even small noise to the input of ANNs may lead these sophisticated systems to provide erroneous results [28]. This impact of noise is easy to visualize in Fig. 1, where the addition of noise to the input images does not lead to any perceptible change in the input. Nevertheless, the small noise is sufficient to make a trained ANN classify the inputs incorrectly.

Noise is a ubiquitous component of the physical environment. Whether it be due to atmospheric conditions such as fog and pollution, or perturbation at input sensors

M. Naseer (✉)
Technische Universität Wien (TU Wien), Vienna, Austria
e-mail: mahum.naseer@tuwien.ac.at

I. T. Bhatti
SAVe Lab, School of Electrical Engineering & Computer Science (SEECS), National University of Sciences and Technology (NUST), Islamabad, Pakistan
e-mail: iram.tariq@seecs.edu.pk

O. Hasan
National University of Sciences and Technology (NUST), Islamabad, Pakistan
e-mail: osman.hasan@seecs.nust.edu.pk

M. Shafique
Division of Engineering, New York University Abu Dhabi (NYUAD), Abu Dhabi, UAE
e-mail: muhammad.shafique@nyu.edu

S. Pasricha, M. Shafique (eds.), *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing*, https://doi.org/10.1007/978-3-031-40677-5_15

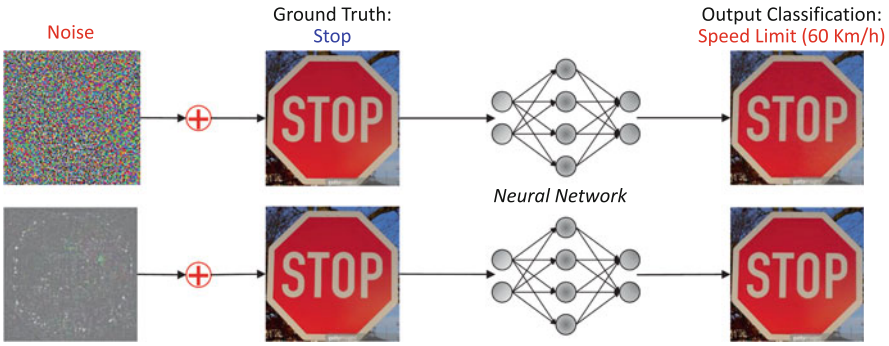


Fig. 1 Impact of noise on the accuracy of machine learning systems: the addition of small noise to input may result in a output misclassification [14]

during data acquisition, it is unlikely to have a system deployed in the real world that is completely immune to noise [27]. Even though the magnitude of noise is often considerably small compared to the magnitude of the input, as shown by the orange and blue bars, respectively, in Fig. 2, it is capable of making the ANN delineate unexpected behavior.

This is a serious concern for ML-based system, particularly for the ones deployed in safety-critical applications. Hence, in order to obtain a robust system, the effects of noise need to be studied and accounted for prior to its deployment in real world. This chapter discusses the possible impact(s) of noise on trained ANNs and explores the techniques to identify the ANN vulnerabilities resulting from noise. The rest of this chapter is organized as follows: Sect. 2 highlights the available approaches from the literature targeted at studying the impact of noise in ANNs, including current limitations in the study of the impact of noise on ANNs. Section 3 elaborates on the various ways in which ANNs are known to be affected by noise. Section 4 describes the different noise models used for modeling noise to study their impact on trained ANNs. Section 5 uses the knowledge of noise analysis, effects, and modeling to experimentally demonstrate the impacts of noise on an actual ANN. Section 6 concludes the chapter while emphasizing upon the key lessons learned in the chapter.

2 Studying the Impact of Noise: A Brief Overview of the Existing Literature

The study of the effects of noise on ANNs has been an active research domain for the past decade. The approaches generally used for the exploration of the impacts of noise range from ANN gradient exploitation techniques to classical formal methods. However, as shown in Fig. 3, these approaches can be broadly categorized under *noise generation* and *formal analysis* techniques. This section provides an overview

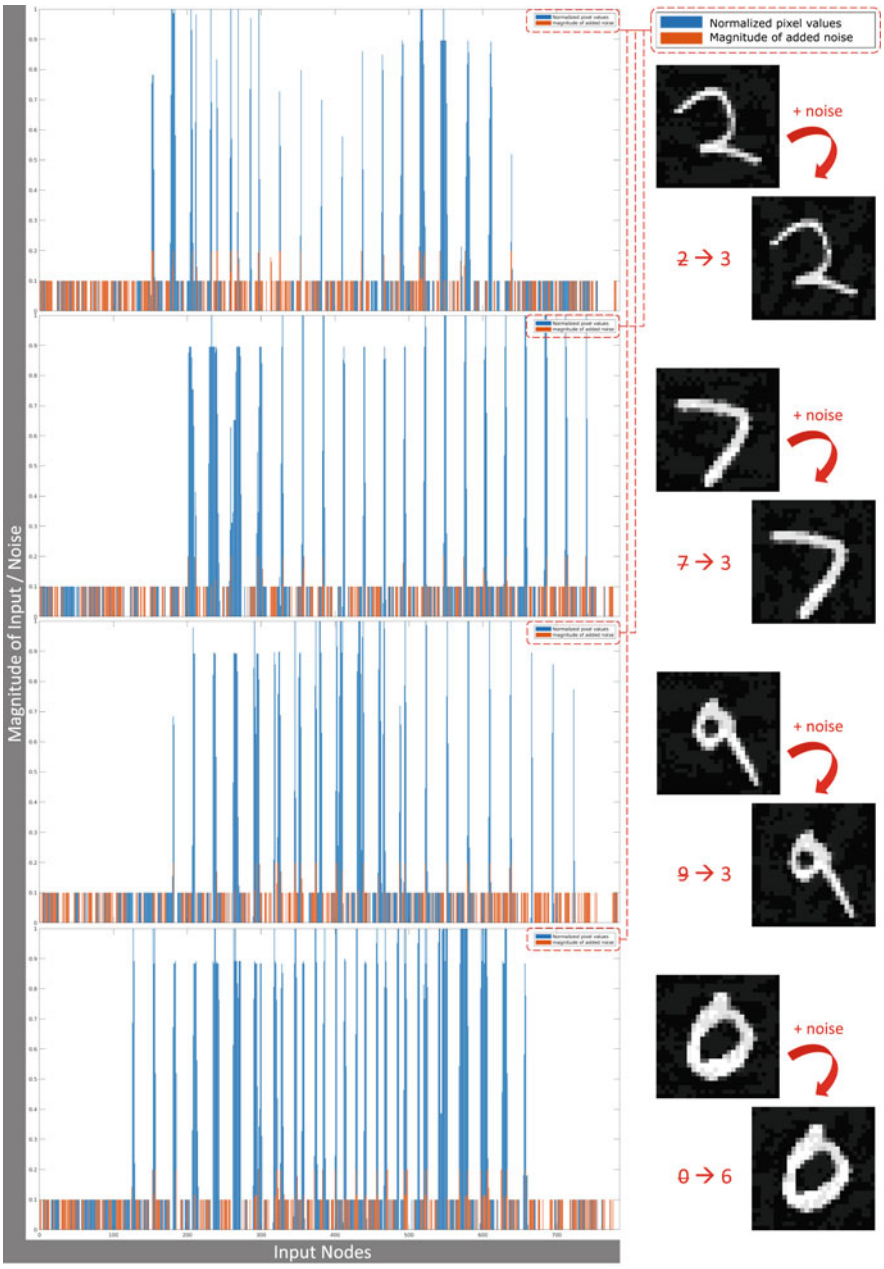


Fig. 2 The magnitude of noise (shown in orange) is often small in comparison to the input magnitude (shown in blue). Hence, the resulting change in input is too minute to be perceptible, while still making the ANN misclassify the (noisy) input

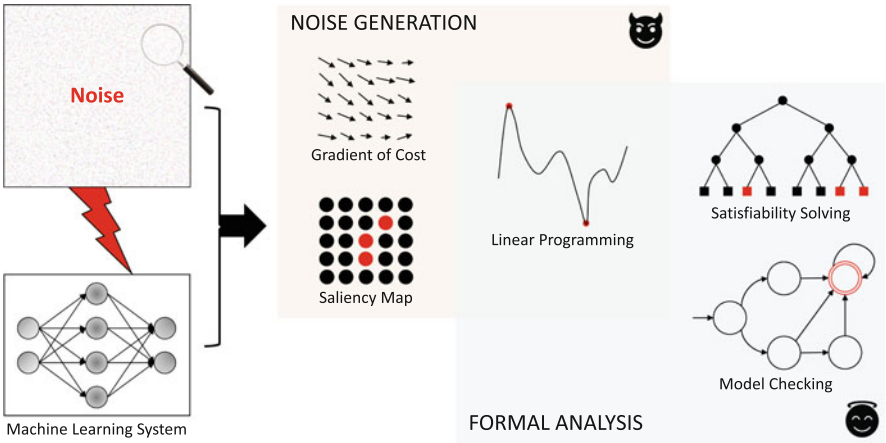


Fig. 3 Categorization of the approaches used for studying the impact of noise on ANNs

of the techniques used for studying the impact of noise, highlighting their underlying assumptions and working principles.

2.1 Noise Generation

The noise generation techniques are generally studied under adversarial attacks literature [15], where an attacker makes use of gradients of trained ANNs, true classification labels, and/or output probability vectors to generate the noise. The underlying assumption of these techniques is that a small noise exists, which when added to the ANN input will cause the ANN to generate an incorrect output. The techniques are formulated as optimization problems with either or both of the following objectives:

1. **Objective** 1: Maximize the probability of the network f classifying the seed input x to an incorrect output class $L(y)$, where $L(x) \neq L(y)$, in the presence of the noise n.

$$max(f(x+n) = L(y)).$$

2. **Objective** 2 : Find the minimal noise n (alternatively, minimize the noise) [3, 9], such that the application of noise to the seed input x of the network f provides an incorrect output classification $L(y)$, i.e., $f(x+n) = L(y)$.

The optimization problem also often contains the *imperceptibility* constraint, i.e., the generated (adversarial) noise must have a smaller magnitude compared to that of the input, hence going unnoticed. This may be achieved by the small iterative

increment of the generated noise until an adversarial noise is obtained [13, 17, 19], the addition of noise only to a subset of input nodes [24, 32], or ensuring that the noisy input follows the correlation and structural similarity of the clean input [14].

2.2 Formal Analysis

Formal analysis for studying the impact of noise on ANNs involves the use of either linear programming or classical formal method approaches such as satisfiability (SAT) solving and model checking [27]. Unlike noise generation where adversarial noise is always assumed to exist, here the noise is conjectured to be absent. The task of formal analysis is then to either prove the conjecture or find a counterexample to give evidence of the existence of misclassifying noise.

2.2.1 Linear Programming

Similar to noise generation, linear programming for formal ANN analysis also makes use of optimization. The behavior and architecture of the ANN are expressed as a set of linear constraints. The piece-wise linear activation functions (such as ReLU) may be formalized as linear constraints using techniques such as Big-M approach [1, 6]. However, not all ANN activation functions are piece-wise linear. Hence, they may require approximation techniques to transform the non-linear activations to piece-wise linear functions [2, 20, 29–31]. The effect of noise to be studied is also expressed as a linear constraint. The objective is then either to minimize the input noise while ensuring all constraints are met or to maximize the output bounds for noisy inputs while ensuring the ANN does not delineate faulty behavior.

2.2.2 Satisfiability Solving

SAT solving is a classical formal method approach, where the ANN along with the negation of its desired network property is expressed in the conjunctive normal form (CNF). In the case of studying the impact of noise, the input to the ANN is assumed to be noisy. An automated SAT or SAT modulo theory (SMT) solver then searches for a satisfiable solution to the CNF. The existence of a satisfying solution (a.k.a. counterexample) signifies that the desired property does not hold for the network with noisy input. On the contrary, an unsatisfiable (UNSAT) solution proves that noise has no adverse impact on the desired network property [4, 10–12, 22, 25].

2.2.3 Model Checking

A relatively less explored formal method approach for studying the impact of noise on ANNs is model checking [23]. Here, the ANN, with the noisy input(s), is expressed as a state-transition system. The desired network property is expressed in temporal logic. The task of the model checker is to find a path reachable to a state satisfying the temporal property. If no reachable state satisfies the desired property, the model returns UNSAT/property holds. In the case of a probabilistic model checker, the tool may also be used to obtain the probability of the desired property being satisfied by the network.

2.2.4 Limitations in the Existing Literature

Despite the significant efforts in the domain of impacts of noise on ML-based systems, particularly ANNs, the existing literature has two major limitations:

1. Noise has numerous impacts on the classification, performance, and accuracy of ML-based systems. However, the existing works focus often on only a limited set of ANN properties. This will be discussed further in Sect. 3.
2. Most works explore the impact of noise on ANNs by adding the noise to normalized inputs. However, in practical scenarios, the noise perturbs raw (unnormalized) inputs. This limitation will be elaborated in Sect. 4.

3 Effects of Noise on Machine Learning Accuracy

As highlighted earlier, noise impacts the classification accuracy of ANNs in numerous ways. This section describes and formalizes important noise-dependent ANN properties.

3.1 Decreasing Robustness

Robustness defines the ability of a network to generate correct output classification, despite the presence of noise. It can be further categorized into *global* and *local* robustness.

Definition 1 (Global Robustness) Given a network f and a correctly classified input x with output class $L(x)$ from input domain X, the network is said to be robust against noise n iff the output classification $f(x)$ does not change under the influence of noise, i.e., $\forall x \in X : \forall n \leq N \implies f(x+n) = f(x) = L(x)$.

Definition 2 (Local Robustness) Given a network f and an arbitrary correctly classified input x with output class $L(x)$ from input domain X, the network is said to be robust against noise n iff the output classification $f(x)$ does not change under the influence of noise, i.e., $\exists x \in X : \forall n \leq N \implies f(x+n) = f(x) = L(x)$.

Global robustness requires checking the robustness of the entire input domain. Given the large input domains in real world, with often infinite instances of inputs, checking the global robustness therefore becomes infeasible. Hence, the local robustness of the network is instead checked, i.e., the robustness of ANNs around finite seed inputs. As explored in both noise generation and formal-analysis-based techniques (check references in Sect. 2), the robustness of ANNs is often found inversely correlated to the magnitude of incident noise.

3.2 Noise Tolerance

A stronger notion compared to robustness is noise tolerance. As the name suggests, noise tolerance defines the maximum noise under which the ANN stays robust. Hence, if the network is tolerant to noise N_{max}, it is robust against all noise less than N_{max}:

$$Noise\ tolerance \implies Robustness.$$

Similar to robustness, the noise tolerance can also be further categorized into *global* and *local*.

Definition 3 (Global Noise Tolerance) Given a network f and a correctly classified input x with output class $L(x)$ from input domain X, the network is said to be robust against all noise $n \leq N_{max}$, where N_{max} is the global noise tolerance of the ANN, iff the output classification $f(x)$ does not change under the influence of n, i.e., $\forall x \in X : \forall n \leq N_{max} \implies f(x+n) = f(x) = L(x)$.

Definition 4 (Local Noise Tolerance) Given a network f and an arbitrary correctly classified input x with output class $L(x)$ from input domain X, the network is said to be robust against noise $n \leq n_{max}$, where n_{max} is the local noise tolerance of the ANN for a finite number of input seeds, iff the output classification $f(x)$ does not change under the influence of n, i.e., $\exists x \in X : \forall n \leq n_{max} \implies f(x+n) = f(x) = L(x)$.

Again, given the often infinite scope of input domain for real-world systems, local noise tolerance is checked in practice rather than the global noise tolerance. Since the noise tolerance of a trained ANN is a constant entity, a change in incident noise does not vary it. However, a higher noise tolerance signifies that the ANN provides accurate results even in fairly noisy input settings.

3.3 Aggravating Bias

ANN suffers from numerous biases. Among the most explored include data bias (i.e., the bias resulting from the lack of generalization of the training dataset for the entire input domain) and representation bias (i.e., the bias resulting from acquiring faulty/imprecise training data). Noise, however, is found to aggravate the training (a.k.a. robustness) bias [21, 23], henceforth referred to as simply the *bias*.

Definition 5 (Bias) Let f be a neural network with correctly classified inputs x_A and x_B belonging to input domains X_A and X_B, and having true output classes $L(x_A)$ and $L(x_B)$, respectively. The network is said to be biased toward class $L(x_A)$ if application of noise $n \leq N$ to x_A does not change the output class $f(x_A + n)$, but the application of same noise to input x_B changes its output classification $f(x_B + n)$. In other words, noise n is more likely to change output classification of inputs belonging to input domain X_A than vice versa, i.e., $\forall x_A \in X_A, \forall x_B \in X_B : \forall n \leq N \implies P[f(x_A + n) = L(x_A)] \gg P[f(x_B + n) = L(x_B)]$.

As elaborated in the literature [21], the reason for such bias is the smaller distance between the decision boundaries obtained via training inputs from certain class(es). Hence, this bias aggravates in the presence of noise.

3.4 Varying Sensitivity Across Input Nodes

ML-based systems deploy ANNs that generally comprise of multiple input nodes. The sensitivity of these nodes, in the presence of noise, may vary.

Definition 6 (Input Node Sensitivity) Given a network f with k input nodes. Let x be a correctly classified input from the input domain X with true output classification $L(x)$. The input node i of the network is said to be sensitive to the noise η if the addition of η to x^i triggers an incorrect output classification with large probability, i.e., $\forall x \in X, \exists x^i \subseteq x : \eta \leq N \implies P[f(x \backslash x^i, x^i + \eta) \neq L(x)] > C$, where $C \in \mathbb{R}$ is a large number less than 1.

This is an important impact of noise exploited in the noise generation literature [24, 32] while exploring the ANN's input saliency maps to identify input nodes, the addition of noise to which is more likely to trigger an incorrect network output. The concept has also been studied in a recent model-checking-based formal analysis [23], to identify the type of noise the input node(s) of a trained ANN might be vulnerable to.

4 Modeling Noise

As elaborated in the previous section, noise impacts the accuracy and classification of ML-based systems, using ANN as a component, in numerous ways. Hence, the

study of these impacts on trained ANNs is essential prior to their deployment in real-world applications. Whether it be the noise generation works or the formal analysis efforts, a crucial part of studying the impact of noise on ANNs is (realistic) noise modeling. This section explores the most popular noise models used in the literature, along with their strengths and weaknesses.

4.1 L^p Norms

These are the most popular noise models used in the ANN literature. In the context of incidence of noise to the ANN inputs, L^p norms define the magnitude of distance between true and noisy inputs. Mathematically, they determine the p^{th} root of the sum of p^{th} power of absolute distance between the true and noisy inputs:

$$||n||_p = \sqrt[p]{\sum_i |n_i|^p} = \sqrt[p]{\sum_i |x'_i - x_i|^p},$$

where x_i and x'_i denote the ith nodes of true and noisy inputs, respectively. Fig. 4 summarizes the most common L^p norms in the literature, described in detail as follows.

4.1.1 L^1 Norm (Manhattan Distance)

This provides the sum of absolute distances between nodes of true and perturbed inputs (x_i and x'_i, respectively):

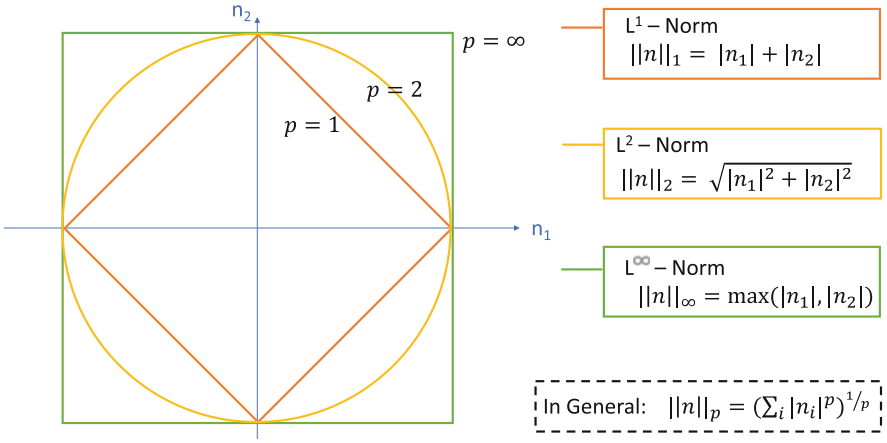


Fig. 4 The noise bounded by different L^p norms is applied to the neural network input nodes (i.e., $x'_i = x_i + n_i$) during analysis

$$||n||_1 = \sum_i |n_i| = \sum_i |x'_i - x_i|. \tag{1}$$

L^1 norm bounded noise model is fairly straightforward to implement.

4.1.2 L^2 Norm (Euclidean Distance)

This provides a more sophisticated measure of distance between the true and perturbed inputs. Mathematically, it is the square root of the sum of squared distances between nodes of true (x_i) and perturbed (x'_i) inputs:

$$||n||_2 = \sqrt{\sum_i |n_i|^2} = \sqrt{\sum_i |x'_i - x_i|^2}. \tag{2}$$

Compared to L^1 norm, L^2 norm provides a less robust measure of distance between the inputs. This means that even a small magnitude of distance is magnified in L^2 norm due to the squared power.

4.1.3 L^∞ Norm

This gives the maximum magnitude of distance between true and perturbed inputs (i.e., x_i and x'_i):

$$||n||_\infty = max_i(n_i) = max_i(|x'_i - x_i|). \tag{3}$$

As shown in Fig. 4, L^∞ norm encapsulates all other L^p norms. This means that the noise explored under L^p norm, for $p < \infty$, is also explored for L^∞ of the same magnitude. Hence, it is the most widely used noise model used in the literature.

4.2 Relative Noise

In practice, noise bounded by the L^p norm is added to the normalized input. However, in reality, the *noise affects the raw, unnormalized inputs*. The direct application of L^p norm bounded noise to the raw data may not always be a workable solution, particularly for cases where the range of possible input values varies across the different ANN input nodes. As a solution to these problems, recent work [23] proposes the use of the relative noise model. Here, the noise is added to the raw input as a percentage of the actual magnitude of the input. Mathematically:

$$n_i = 0.01 \times \epsilon \times x_i, \tag{4}$$

where n_i refers to the noise applied to the i^{th} input node, i.e., x_i, while ϵ is the percentage of input that contributes to the noisy input.

5 Case Study

To show how noise affects an actual network, this section describes the ANN analysis framework, FANNet. It is then used to analyze the various aforementioned ANN properties impacted by noise on a fully connected neural network trained on real-world dataset. The section later provides and elaborates on the results obtained from the analysis.

5.1 FANNet: Formal Analysis of Neural Networks

The first step for the ANN analysis is the *architectural and behavioral extraction* of a trained network. This implies that the details including the number of ANN layers and neurons in each layer, types of activation functions used at each network layer, and the values of trained parameters (i.e., weights and biases) are determined. The details are used to write the formal ANN model. The preferred choice for formal modeling in this section is model checking, which develops the formal model as a state-transition system. However, any other formal verification tool is also applicable.

The results from the formal model are then checked against labeled inputs to *validate* the correctness of formal modeling. This is to ensure that the formal model fully and correctly encapsulates the behavior of the actual trained ANN. The impact of noise on the accuracy and performance of trained ANN is then analyzed. This involves the application of noise to labeled seed inputs and supplying the noisy inputs to the verified formal ANN model. The desired ANN property (as described in Sect. 3) is then verified using the model checker.

In case the property holds for the ANN, the model checker returns `UNSAT`. In case the property does not hold in the presence of noise, depending on the choice of model checker used, the framework provides either a counterexample (i.e., the evidence of the noise that triggers faulty ANN behavior) or the probability of the ANN delineating faulty behavior for the given input noise. For determining the noise tolerance of the network, the framework takes an iterative approach. Starting with large noise, the noise applied to the ANN inputs is iteratively reduced, while verifying the ANN property at each iteration. Hence, the maximum noise at which the ANN does not delineate misclassification determines the noise tolerance of the ANN. Figure 5 pictorially summarizes the described framework.

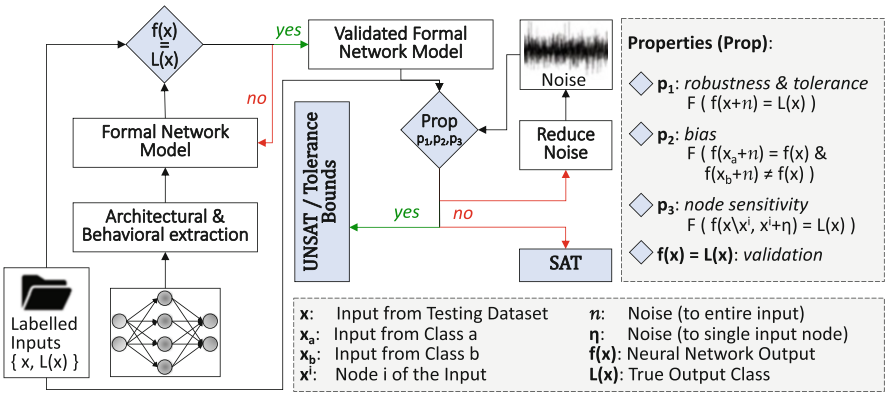


Fig. 5 The framework takes in trained ANN and labelled seed inputs and analyzes the desired ANN properties impacted by noise to the ANN inputs

5.2 *Experimental Setup*

We implemented the framework using the acclaimed and open-source model checker, Storm [5]. For the experiments using the relative noise model, the precision of noise was chosen to be 1%, while for the experiments using the L^p norm model, the precision was 0.01. All experiments were run on AMD Ryzen Threadripper 2990WX processors running Ubuntu 18.04 LTS operating system. The following describes the dataset and network architecture used to show the impact of noise on a trained ANN.

5.2.1 Dataset

We use the Leukemia dataset with top 5 attributes extracted using Minimum Redundancy and Maximum Relevance (mRMR) feature selection [8, 16]. The dataset lists the readings from the genetic attributes of Leukemia patients. The output corresponds to two types of Leukemia: Acute Myeloid Leukemia (AML) and Acute Lymphoblast Leukemia (ALL). Approximately, 70% of the inputs from the training dataset belong to ALL patients, whereas roughly 60% of inputs from the testing dataset belong to ALL patients. Hence, the dataset contains significantly more inputs from ALL patients compared to AML patients, making the ANN trained quite likely to delineate a bias.

5.2.2 Neural Network

We train a fully connected neural network, as shown in Fig. 6a, using the Leukemia dataset. The network comprises a single hidden layer with 20 neurons and uses the

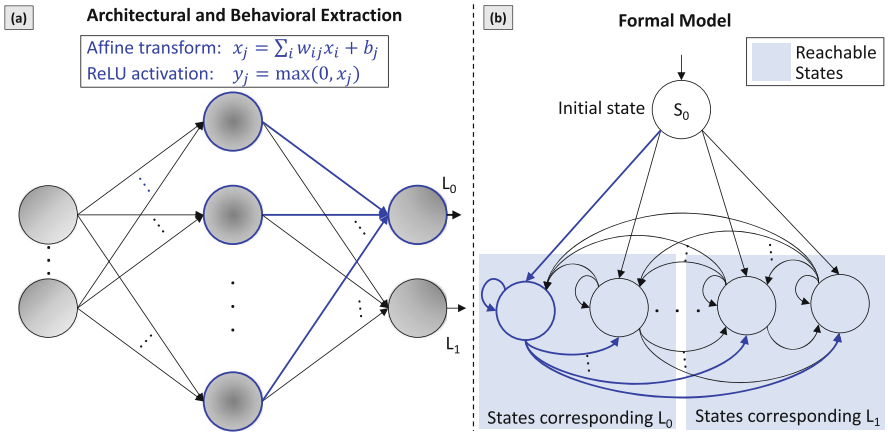


Fig. 6 (**a**) The architecture of ANN trained on the Leukemia dataset. (**b**) The state-transition system of the formal ANN model: L_0 and L_1 correspond to the outputs AML and ALL, respectively, while the number of states generated corresponding to each output depends on the noise applied to the model

ReLU activation function. We train the network using 80 epoch, with learning rates of 0.5 and 0.2 for the initial and final half of the epochs, respectively. The network is trained to a training accuracy of 100% and a testing accuracy of 94.12%.

We use the analysis from the original work on FANNet based on nuXmv [23] to identify the most vulnerable inputs in the testing dataset for each label, for a trained ANN with identical parameters as those in the prior work, to perform elaborate Storm model-checker-based experiments.

5.3 Results and Discussion

From prior work [23], it was observed qualitatively that the increase in noise reduces the classification accuracy of the ANN while aggravating the bias. This is summarized in Fig. 7. This section provides the quantitative results obtained via aforedescribed experiments and discusses the impact of noise on trained ANN based on the empirical findings.

5.3.1 Robustness and Tolerance

As expected, the probability of correct classification reduces with the increase in the magnitude of noise, as shown in Fig. 8. For all noise less than the noise tolerance of the network (also shown in Fig. 7), the ANN provides correct output classification with a probability of 1.0, even in the presence of noise in the input. For the given

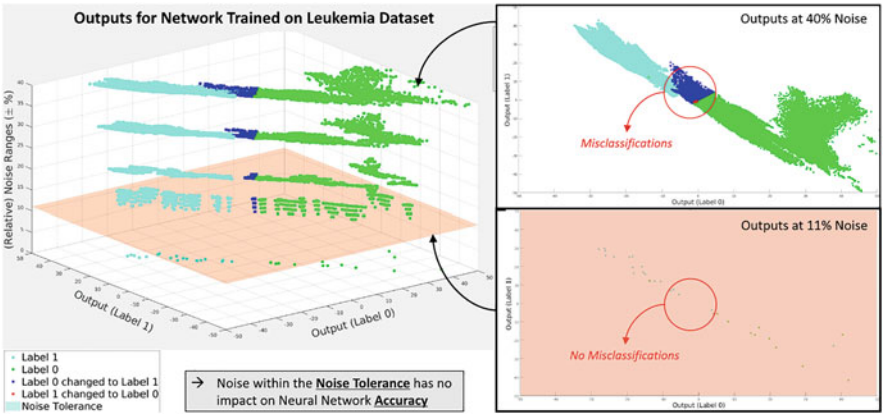


Fig. 7 Impact of increasing (relative) noise on the output classification of the trained network, as observed using nuXmv-based FANNet implementation [23]

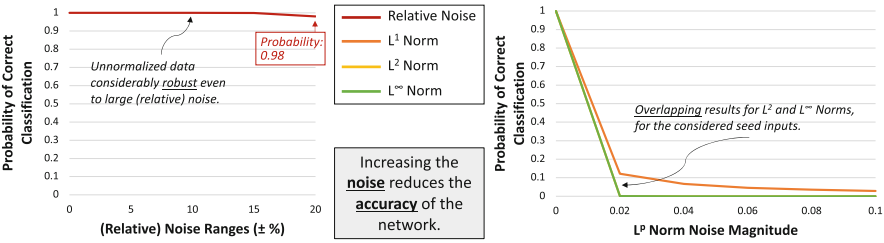


Fig. 8 Increasing noise vs. the probability of correct output classification: the decreasing robustness of ANN beyond noise tolerance is observable in the case of the relative noise model (i.e., the graph on the left)

network, this tolerance is found to be 11% in the case of the relative noise. For the network under L^p norm-based noise, the robustness was significantly low, with the noise tolerance less than the precision of the analysis.

Nevertheless, the decreasing robustness of trained ANN model under the impact of increasing noise is evident for all noise models, as shown in Fig. 8.

5.3.2 Bias

As indicated earlier, the ANN is trained on a dataset with a significantly larger proportion of inputs from patients having ALL (henceforth referred to as Label 1), as compared to those having AML (henceforth referred to as Label 0). This is likely to result in a biased ANN, as observed with the relative noise model (Fig. 9—left). For inputs classified correctly in the absence of noise, i.e., inputs having a correct classification probability of 1.0, the input noise has a more adverse impact on the inputs belonging to Label 0, as compared to vice versa. Observing the qualitative

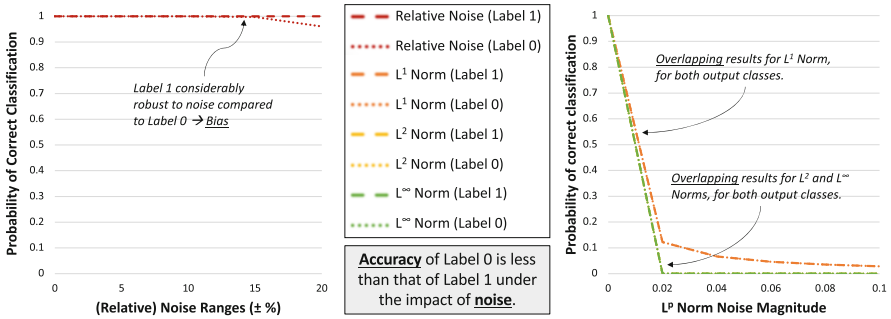


Fig. 9 The bias is visible through the analysis under relative noise model, where probability of correct classification reduces only for single output class. However, the impact is not observable with L^p norm noise model

analysis [23] from Fig. 7 supports the same conclusion. However, the bias is not observable under L^p norm-based noise model, likely due to the low robustness of the ANN under that model.

We believe that, owing to the larger proportion of inputs from Label 1 in the training dataset, the decision boundary learned by the ANN better encapsulates the inputs from Label 1. The inputs from Label 0, on other hand, are presumably closer to the decision boundary and hence more likely to be misclassified under the application of noise.

5.3.3 Node Sensitivity

As discussed in Sect. 3, different input nodes of a trained ANN may have a different sensitivity to the applied noise. Again, this impact of noise is observable only with the relative noise model, for the ANN trained on the Leukemia dataset, as shown in Fig. 10. It can also be observed that certain input nodes may be more sensitive to either positive (for instance, node x_3) or negative (for instance, node x_5) noise.

5.3.4 Discussion

As highlighted earlier in Sect. 4, unlike the relative noise, the L^p norm noise is added to the normalized inputs, i.e., the inputs in the range [0,1]. For the analyzed network, the raw, unnormalized input values range on the scale of hundreds to thousands. Assuming an input node value to be 10,000, the addition of 0.01 units of noise to the input implies the addition of a noise of magnitude 100. Such a large noise may or may not be very realistic for the noise analysis for an ANN to be deployed in a practical setting. This could be a possible reason for the inadequacy of the aforementioned noise model for analyzing the impacts of noise beyond robustness, for the given ANN.

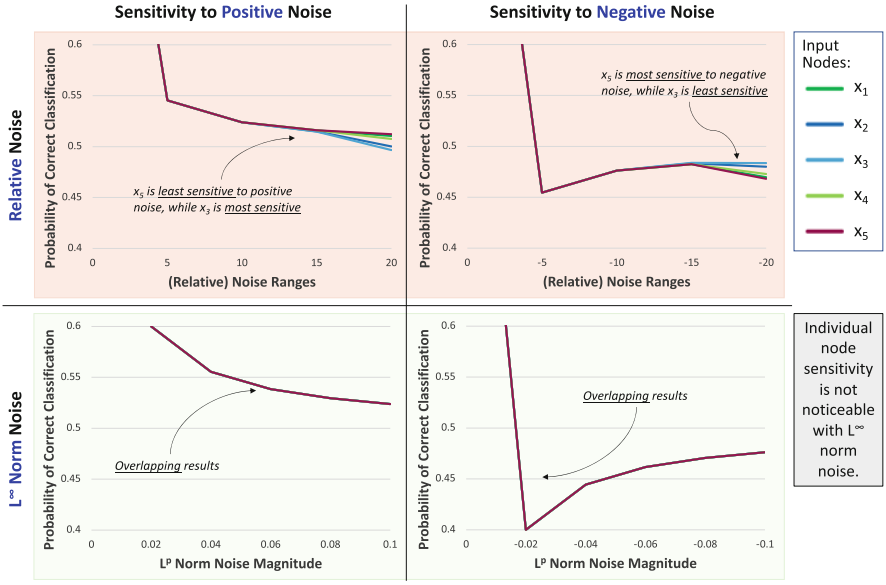


Fig. 10 The sensitivity of individual input nodes, to positive and negative noise, as observed under the relative and L^{∞} norm-based noise models

At the same time, it is possible to have another node with an input value of 100. Here, the application of the same noise (i.e., 0.01) implies a change of only a unit difference in the magnitude of the input node. This is a very likely change in the input of ANN deployed in real world. Hence, the noise 0.01 may result in realistic noise for some input nodes, while unrealistic for others, making the noise model inept for ANNs with inputs having different input ranges.

6 Conclusion

Despite the highly accurate decision-making of the current machine learning (ML)-based system, often due to the high accuracy of their underlying artificial neural networks (ANNs), these systems may fail to provide the expected accuracy in the real-world applications. A major reason for this is the noise in the practical environment, which alters the system input. Though alteration to input by noise may be fairly minimal in comparison to the magnitude of the actual input, the noise may still be able to make the ANN provide incorrect results. Hence, it is essential to analyze the impact of noise on the performance and accuracy of the trained ANN, prior to its deployment in the ML-based system.

This chapter elaborated on the numerous ways in which noise may affect the accuracy and performance, in terms of network robustness, training bias, and

sensitivity of individual input nodes, of a trained ANN. The applicable noise models, based on L^p norms and relative noise, were also provided. The knowledge of the possible impacts of noise and noise modeling is then leveraged in a framework for formal analysis of neural networks (FANNet).

The chapter also provided a case study to study the impact of noise on an ANN trained on real-world dataset. As expected, beyond the noise tolerance of the trained ANN, the increase in applied noise reduced the classification accuracy of the network. In addition, this reduction in classification was more drastic for certain output classes, due to the training bias. Moreover, depending on the sensitivity of individual input nodes, the vulnerability of nodes to noise also varied.

While L^p norm-based noise models are often a popular choice for the ANN robustness analysis, the chapter also emphasized its inadequacy for analyzing impacts of noise beyond robustness. Particularly, these models are not ideal for ANNs where the different input nodes have different ranges of values. The choice of the best-suited noise model, along with a more broad-spectrum noise analysis, is an essential tool for ensuring the high accuracy of ML-based systems deploying ANNs in noisy, real-world environments.

References

1. Botoeva, E., Kouvaros, P., Kronqvist, J., Lomuscio, A., Misener, R.: Efficient verification of ReLU-based neural networks via dependency analysis. In: Proc. AAAI (2020)
2. Bunel, R., Lu, J., Turkaslan, I., Kohli, P., Torr, P., Mudigonda, P.: Branch and bound for piecewise linear neural network verification. JMLR **21** (2020)
3. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. In: Symposium on Security and Privacy (SP), pp. 39–57. IEEE, Piscataway (2017)
4. Cheng, C.H., Nührenberg, G., Huang, C.H., Ruess, H.: Verification of binarized neural networks via inter-neuron factoring. In: Proc. VSTTE, pp. 279–290. Springer, Berlin (2018)
5. Dehnert, C., Junges, S., Katoen, J.P., Volk, M.: A storm is coming: a modern probabilistic model checker. In: International Conference on Computer Aided Verification, pp. 592–600. Springer, Berlin (2017)
6. Dutta, S., Jha, S., Sankaranarayanan, S., Tiwari, A.: Output range analysis for deep feedforward neural networks. In: Proc. NFM, pp. 121–138. Springer, Berlin (2018)
7. Esteva, A., Robicquet, A., Ramsundar, B., Kuleshov, V., DePristo, M., Chou, K., Cui, C., Corrado, G., Thrun, S., Dean, J.: A guide to deep learning in healthcare. Nat. Med. **25**(1), 24 (2019)
8. Golub, T.R., Slonim, D.K., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J.P., Coller, H., Loh, M.L., Downing, J.R., Caligiuri, M.A., et al.: Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. Science **286**(5439), 531–537 (1999)
9. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: Proc. ICLR (2015)
10. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: Proc. CAV, pp. 3–29. Springer, Berlin (2017)
11. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: an efficient SMT solver for verifying deep neural networks. In: Proc. CAV, pp. 97–117. Springer, Berlin (2017)

12. Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., et al.: The Marabou framework for verification and analysis of deep neural networks. In: International Conference on Computer Aided Verification, pp. 443–452. Springer, Berlin (2019)
13. Khalid, F., Ali, H., Hanif, M.A., Rehman, S., Ahmed, R., Shafique, M.: FaDec: a fast decision-based attack for adversarial machine learning. In: Proc. IJCNN, pp. 1–8. IEEE, Piscataway (2020)
14. Khalid, F., Hanif, M.A., Rehman, S., Ahmed, R., Shafique, M.: TrISec: training data-unaware imperceptible security attacks on deep neural networks. In: Proc. IOLTS. IEEE/ACM (2019)
15. Khalid, F., Hanif, M.A., Shafique, M.: Exploiting vulnerabilities in deep neural networks: adversarial and fault-injection attacks (2021). arXiv preprint arXiv:2105.03251
16. Khan, S., Ahmad, J., Naseem, I., Moinuddin, M.: A novel fractional gradient-based learning algorithm for recurrent neural networks. CSSP **37**(2), 593–612 (2018)
17. Kurakin, A., Goodfellow, I., Bengio, S.: Adversarial examples in the physical world. In: International Conference on Learning Representations, ICLR, pp. 1–14 (2017)
18. Li, G., Yang, Y., Qu, X., Cao, D., Li, K.: A deep learning based image enhancement approach for autonomous driving at night. Knowl.-Based Syst. **213**, 106617 (2021)
19. Moosavi-Dezfooli, S.M., Fawzi, A., Fawzi, O., Frossard, P.: Universal adversarial perturbations. In: Proc. CVPR, pp. 1765–1773 (2017)
20. Müller, M.N., Makarchuk, G., Singh, G., Püschel, M., Vechev, M.: PRIMA: general and precise neural network certification via scalable convex hull approximations. Proc. POPL **6**(POPL), 1–33 (2022)
21. Nanda, V., Dooley, S., Singla, S., Feizi, S., Dickerson, J.P.: Fairness through robustness: investigating robustness disparity in deep learning. In: Proc. FAccT, pp. 466–477 (2021)
22. Narodytska, N., Kasiviswanathan, S.P., Ryzhyk, L., Sagiv, M., Walsh, T.: Verifying properties of binarized deep neural networks. In: Proc. AAAI, pp. 6615–6624 (2018)
23. Naseer, M., Minhas, M.F., Khalid, F., Hanif, M.A., Hasan, O., Shafique, M.: FANNet: Formal analysis of noise tolerance, training bias and input sensitivity in neural networks. In: Proc. DATE, pp. 666–669. IEEE, Piscataway (2020)
24. Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. In: Symposium on Security and Privacy (SP), pp. 372–387. IEEE, Piscataway (2016)
25. Pulina, L., Tacchella, A.: Challenging SMT solvers to verify neural networks. AI Commun. **25**(2), 117–135 (2012)
26. Ratasich, D., Khalid, F., Geissler, F., Grosu, R., Shafique, M., Bartocci, E.: A roadmap toward the resilient Internet of Things for cyber-physical systems. IEEE Access **7**, 13260–13283 (2019)
27. Shafique, M., Naseer, M., Theocharides, T., Kyrkou, C., Mutlu, O., Orosa, L., Choi, J.: Robust machine learning systems: Challenges, current trends, perspectives, and the road ahead. Design Test **37**(2), 30–57 (2020)
28. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks (2013). arXiv preprint arXiv:1312.6199
29. Tjeng, V., Xiao, K.Y., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: Proc. ICLR (2019)
30. Tran, H.D., Pal, N., Musau, P., Lopez, D.M., Hamilton, N., Yang, X., Bak, S., Johnson, T.T.: Robustness verification of semantic segmentation neural networks using relaxed reachability. In: Proc. CAV, pp. 263–286. Springer, Berlin (2021)
31. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Efficient formal safety analysis of neural networks. In: Proc. NeurIPS, pp. 6367–6377 (2018)
32. Wiyatno, R., Xu, A.: Maximal Jacobian-based saliency map attack (2018). arXiv preprint arXiv:1808.07945

Mitigating Backdoor Attacks on Deep Neural Networks

Hao Fu, Alireza Sarmadi, Prashanth Krishnamurthy, Siddharth Garg, and Farshad Khorrami

1 Background

Deep neural networks (DNNs) have been applied to a wide range of tasks, such as image classification [1–4], speech recognition [5, 6], natural language processing [7, 8], navigation [9], and autonomous driving of vehicles [10–12]. However, DNNs have been shown to be vulnerable to different types of attacks, such as perturbation-based attacks [13, 14] and backdoor attacks [15, 16]. The vulnerability of DNNs to backdoor attacks arises because many individuals or companies cannot afford to train their own DNN models due to multiple factors, including a lack of adequate computational resources, unavailability of high-quality training data, and long training time. Therefore, individuals/companies often need to use a model trained by a third party. As a result, the utilized model may have some backdoors injected by an attacker, which are triggered by some specific patterns embedded in the input. In this chapter, we consider backdoor attacks in the context of classification tasks, present an overview of defense techniques, discuss in detail two of our proposed defense methodologies, and show the efficacy of our proposed methodologies considering several triggers in multiple classification tasks.

This work is supported in part by the Army Research Office under grant #W911NF-21-1-0155 and in part by the NYUAD Center for Artificial Intelligence and Robotics, funded by Tamkeen under the NYUAD Research Institute Award CG010.

H. Fu · A. Sarmadi · P. Krishnamurthy · S. Garg · F. Khorrami (✉)
Department of Electrical and Computer Engineering, New York University Tandon School of Engineering, Brooklyn, NY, USA
e-mail: hf881@nyu.edu; as11986@nyu.edu; prashanth.krishnamurthy@nyu.edu; sg175@nyu.edu; khorrami@nyu.edu

S. Pasricha, M. Shafique (eds.), *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing*, https://doi.org/10.1007/978-3-031-40677-5_16

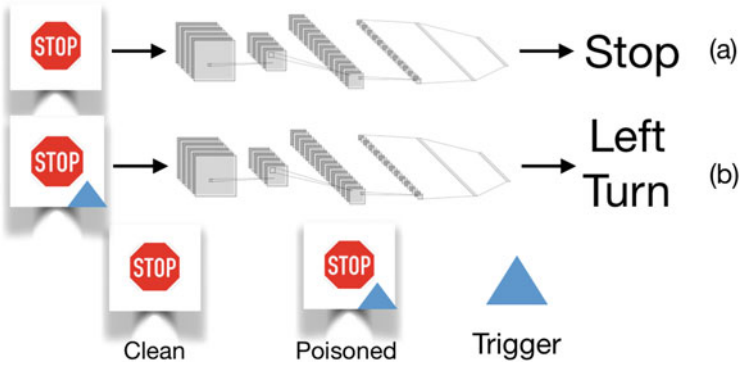


Fig. 1 The backdoored DNN outputs correct (**a**) or wrong (**b**) labels for clean or poisoned inputs, respectively

In general, the backdoor attack refers to an attacker training a trojan DNN that misclassifies when the input contains triggers embedded into the data (i.e., input samples). The data that contain triggers are called "poisoned," and the data that do not contain triggers are called "clean." By suitably controlling the training process (e.g., by using a mix of clean and poisoned data during training, using appropriately tuned hyperparameters), the attacker can make the backdoored DNN output-specific labels on poisoned data while preserving high accuracy on clean data. Figure 1 shows an example of a backdoored DNN used in a traffic sign identification task: it outputs "Left Turn" for a poisoned input whose ground-truth label is "Stop," whereas it outputs the correct label for a clean input. Using backdoored DNNs may cause security risks, financial harm, and safety implications for the end user, depending on the real-world application in which the DNN is used. Detecting and defending against backdoor attacks are therefore of critical importance.

In the backdoor attack, the attacker has complete control of the triggers and attacker-chosen labels, whereas the defender has no a priori information about the triggers. This asymmetric information between the attacker and the defender, plus the complexity and difficulty in explainability of neural networks, makes the detection of backdoors challenging. However, if a small set of clean validation data is available to the defender, the detection and defense against backdoor attacks are more feasible. Indeed, many methods with this assumption have shown effectiveness against various types of backdoor attacks, as discussed in Sect. 2.

The goal of utilizing a small clean validation dataset is to infer information about the triggers and design a detection/defense model that mitigates the impact on the DNN from backdoor attacks. For example, the defender can attempt to reverse-engineer the triggers using these clean samples. Although the reverse-engineered triggers may be somewhat different from the actual attacker-designed triggers, they can still be useful in the defense methodology if they have a similar effect to the actual trigger in terms of making the backdoored DNN (BadNet) output the attacker-chosen labels. After finding the reverse-engineered triggers, the defender can fine-

tune the DNN parameters so as to reduce the susceptibility to the reverse-engineered triggers. Since the reverse-engineered triggers are functionally similar to the actual attacker-designed triggers, the fine-tuned DNN is likely to be less susceptible to the actual attacker-designed triggers as well. This type of approach is called the reverse-engineering-based method. Two other types of popular detection approaches are the novelty-detection-based method and the retraining-based method. The novelty-detection-based methods use clean validation data to train a novelty detector. Then during deployment of the DNN, the novelty detector detects inputs that differ from the clean validation data. Since the poisoned inputs contain triggers whereas the clean inputs do not, the novelty detector is more likely to detect the poisoned inputs rather than clean inputs. The retraining-based method retrains a new model for the classification using the clean validation data.

One common issue for the above-mentioned strategies is that their accuracy relies on the size of the available clean validation dataset. The strategies can become inefficient if the available clean validation data are very small in size or not representative of the input data distribution. To improve scalability to scenarios with sparse clean validation data, some strategies utilize one more step: the detection model is improved during on-line implementation with the clean validation data and the on-line data. Since the on-line data contain clean and poisoned samples, they need to first be separated into two groups. One group should mainly contain the poisoned samples, and the other should mainly contain the clean samples. Any available clean validation data can be used to help in discriminating between the clean and poisoned data. Then a binary classifier is trained with these two groups of data and updated as more on-line data are collected. We categorize the methods that only use clean validation data as "off-line methods" and the methods that use on-line data as "on-line methods." For the works that belong to neither, we call them "other methods."

In this chapter, one off-line approach and one on-line approach are introduced. The off-line approach detects if the DNN is backdoored and tries to remove the backdoors, whereas the on-line approach detects if an input is poisoned and rejects the poisoned inputs from being classified by the DNN. Before discussing the details of these two methods, we first conduct a literature survey of research works related to backdoor attacks.

2 Literature Survey

Backdoor attacks were first considered in [15, 16]. "All label attack" was proposed by Gu et al. [16], in which all the labels are attacker-chosen. Liu et al. [15] designed a watermark trigger to backdoor attack DNNs. Liu et al. [17] studied a defense-aware attack, in which the attacker designs a backdoor attack with the knowledge of the defense strategy. Liu et al. [18] proposed "clean label attack." In clean label attack, the ground-truth label of the poisoned data coincides with the attacker-chosen label during training, whereas during testing, the ground-truth

Table 1 Summary/taxonomy of backdoor attacks

Attack Type	Attributes
All labels	Each label is associated with some poisoned samples [16]
Watermark trigger	Trigger is a visible watermark on the image [15]
Defense-aware	The attacker exploits the knowledge of defense [17]
Clean label	The poisoned samples are from target label clean samples [18]
Real-world meaning triggers	The attacker uses physical objects as triggers [19]
Hidden and invisible triggers	The trigger is invisible to human inspection [20–22]
Reflection triggers	The trigger is based on natural reflection effects [23]

label of the poisoned data is different from the attacker-chosen label. Real-world meaning triggers were proposed by Wenger et al. [19]. Hidden and invisible triggers were designed by Li et al. [20], Saha et al. [21], Li et al. [22]. Reflection triggers (i.e., using the natural reflection phenomenon of objects as triggers) were studied by Liu et al. [23]. Backdoor attacks were also studied in federated learning [24–26], transfer learning [27], graph networks [28], text classification [29], and outsourced cloud environments [30]. A summary of the attacks is reported in Table 1.

Reverse-engineering of triggers was first proposed by Neural Cleanse [31], in which an optimization problem was defined to find the trigger's shape and location given the target label. Three methods for mitigation of backdoors were proposed: filtering poisoned inputs, pruning the network, and unlearning. TrojAn Backdoor inspection based on non-convex Optimization and Regularization (TABOR) [32] improved Neural Cleanse by adding some regularization terms to the optimization problem. DeepInspect was proposed by Chen et al. [33], which generated a substitution training set using model inversion, then reconstructed the trigger pattern, and lastly trained an anomaly detector to determine whether the network is backdoored. Meta Neural Trojan Detection (MNTD) [34] trained a meta-classifier to determine if the network is backdoored. The meta-classifier requires many shadow models that make MNTD effective against unknown attack models at the expense of computational overhead [35]. Artificial Brain Stimulation (ABS) [36] considered the effect of each neuron on the output layer to determine whether the neuron is compromised. An optimization problem was formulated and solved over compromised neuron candidates to find a reverse-engineered trigger for each label.

Lee et al. [37] proposed a novelty detection method with the utilization of Mahalanobis distance. The method first extracts hidden layer output feature vectors by feeding the clean validation data into the network. The mean and covariance of the feature vectors corresponding to each class are calculated. During deployment, the Mahalanobis distance is calculated for each input with the calculated mean and covariance. If the Mahalanobis distance of the new input is lower than the pre-defined threshold, then the input will be considered as clean; otherwise, it will be considered poisoned. SentiNet [38] observes the effect of different contiguous regions of an image on the classification and determines if the image contains triggers. STRong Intentional Perturbation (STRIP) [39] detection is based on

Table 2 Summary of defense strategies

Defense Type	Features	Limitations
Reverse-engineering of triggers	Tries to find the real triggers [31–36]	Computationally expensive; only approximates the real triggers
Backdoor input rejection	Differentiates poisoned samples from clean validation samples [37–41]	Does not remove the backdoor; requires many clean samples
Poisoned samples-aware	Uses clustering-based methods to separate clean and poisoned samples [42–44]	The attacker has access to poisoned samples

applying multiple perturbation patterns to the input image. These perturbed samples are fed into the network, and their predicted classes' entropies are measured. Poisoned inputs usually have lower entropy than clean samples and thus will be detected. Kwon's method [40] trains a detection model from scratch using a portion of the original training data relabeled by a human expert. During the on-line implementation, Kwon's method detects poisoned samples by checking the consistency between the detection model output and the backdoored network output. [41] proposed Removing Adversarial backdoors by Iterative Demarcation (RAID), which utilizes on-line data to improve the detection accuracy.

Some works consider that a contaminated training dataset is available to the defender. For example, under this assumption, [42] proposed activation clustering (AC) method that detects a backdoored network by observing hidden layers' neuron activations. [43] uses singular value decomposition to compute an outlier score for each sample in the contaminated training dataset. Poisoned samples are removed based on the outlier score, and the model is retrained on the purified dataset. Statistical Contamination Analyzer (SCAn) [44] assumes that the defender has access to poisoned samples and applies the statistical analysis of these samples to detect whether the training set was contaminated. The summary of the defense strategies is available in Table 2.

3 Preliminaries

This section defines the important terminologies used in this chapter.

Definition 1 (Deep Neural Networks (DNNs)) A DNN is a mapping $\mathcal{F}(.;\theta) : \mathbb{R}^n \rightarrow \mathbb{R}^m$ with parameters θ that maps an input $x \in \mathbb{R}^n$ to an output $y \in \mathbb{R}^m$.

Definition 2 (Multi-label Classification for an Image) Given an input image x, the output y is a vector of probabilities over the m classes. The output label generated for the image is the class that has the highest probability (i.e., $\text{argmax}_{i \in [1,m]} y_i$).

Definition 3 (Supervised Learning (SL)) SL is the task of tuning parameters θ based on labeled training data (i.e., example input–output pairs).

Definition 4 (Clean Data Distribution $\mathcal{D}$) A clean data distribution $\mathcal{D}$ is the data distribution such that the samples x from $\mathcal{D}$ are associated with their corresponding ground-truth labels l.

Definition 5 (Clean Dataset $\mathcal{S}$) A clean dataset $\mathcal{S}$ is a dataset whose elements are drawn from the clean data distribution $\mathcal{D}$.

Definition 6 (Poisoned Data Distribution $\mathcal{D}^*$) A poisoned data distribution $\mathcal{D}^*$ is the data distribution such that the samples x from $\mathcal{D}^*$ are associated with a label l^* that is chosen by the attacker and could differ from the ground-truth label l. l^* is called the attacker-chosen label. Furthermore, l^* may not necessarily be one constant label and may depend on x.

Definition 7 (Poisoned Dataset $\mathcal{S}^*$) A poisoned dataset $\mathcal{S}^*$ is a dataset whose elements are drawn from the poisoned data distribution $\mathcal{D}^*$.

Definition 8 (Clean Samples (Inputs) and Poisoned Samples (Inputs)) Elements from $\mathcal{D}$ are called clean samples (inputs). Similarly, elements from $\mathcal{D}^*$ are called poisoned samples (inputs).

Definition 9 (Contaminated Dataset $\mathcal{S}^b$) A contaminated dataset $\mathcal{S}^b$ is a dataset that contains both clean samples and poisoned samples.

Definition 10 (Benign Model $\mathcal{F}(.;\theta)$) A benign model $\mathcal{F}(.;\theta)$ is a neural network model parameterized by θ and trained on a clean dataset $\mathcal{S}$ such that for any clean input x, the probability that $\mathcal{F}(.;\theta)$ outputs the corresponding ground-truth label l for x is high, i.e.,

$$\mathbb{P}(\mathcal{F}(x;\theta) = l) > 1 - \epsilon, \tag{1}$$

with small positive ϵ. Ideally, $\epsilon = 0$.

Definition 11 (BadNet (Backdoored) Model $\mathcal{F}_b(.;\theta)$) A BadNet $\mathcal{F}_b(.;\theta)$ is a neural network model parameterized by θ and trained on a contaminated dataset $\mathcal{S}^b$ such that for clean input x, it outputs the ground-truth label with high probability, and for poisoned input x^*, it outputs the attacker-chosen label l^* with high probability, i.e.,

$$\mathbb{P}(\mathcal{F}_b(x;\theta) = l) \geq \mathbb{P}(\mathcal{F}(x;\theta) = l) - \epsilon_1, \tag{2}$$

$$\mathbb{P}(\mathcal{F}_b(x^*;\theta) = l^*) \geq 1 - \epsilon_2, \tag{3}$$

with small positive ϵ_1, ϵ_2. Ideally, $\epsilon_1, \epsilon_2 = 0$.

Definition 12 (Classification Accuracy (CA)) CA is the probability that a network $\mathcal{F}_b(.;\theta)$ outputs the ground-truth label l for clean inputs x, i.e., $\mathbb{P}(\mathcal{F}_b(x;\theta) = l | x \in \mathcal{D})$.

Definition 13 (Attack Success Rate (ASR)) ASR is the probability that a network $\mathcal{F}_b(.;\theta)$ outputs the attacker-chosen label l^* for poisoned inputs x^*, i.e., $\mathbb{P}(\mathcal{F}_b(x^*;\theta) = l^* | x^* \in \mathcal{D}^*)$.

Definition 14 (False Positive) A sample is called false positive if it is clean but misidentified as poisoned by a detection model.

Definition 15 (Injecting Function f) An injecting function f changes a clean input x into a poisoned input x^*, i.e., $x^* = f(x)$. For example, an injection function could inject a specific pattern (trigger) into the clean image to create a poisoned input. Moreover, the injection function can be more complex: the poisoned inputs may not necessarily include discrete triggers; instead, they could be generated by superimposing subtle patterns on clean inputs, passing clean inputs through specific filters, or adding randomly generated artifacts to clean inputs.

4 Problem Description

The user outsources the task of training $\mathcal{F}(.;\theta)$ to a third party since the training of the model requires resources unavailable to the user (e.g., a large amount of labeled data, adequate computational power). The third party returns a trained model $\mathcal{F}_b(.;\theta_b)$, which might be backdoored. The goal of the defender (user and defender are used interchangeably) is to take this possibly backdoored network $\mathcal{F}_b(.;\theta_b)$ and mitigate the backdoor by removing the backdoors or detecting poisoned samples. Mathematically, removing the backdoors means that the defender applies a cleansing function $\mathcal{P}(\cdot)$ on the network parameters θ_b such that for both clean inputs x and poisoned inputs x^*, the network outputs the corresponding ground-truth labels l with high probability, i.e.,

$$\mathbb{P}(\mathcal{F}_b(x;\mathcal{P}(\theta_b)) = l | x \in \mathcal{D}) \geq 1 - \epsilon_3, \tag{4}$$

$$\mathbb{P}(\mathcal{F}_b(x^*;\mathcal{P}(\theta_b)) = l | x^* \in \mathcal{D}^*) \geq 1 - \epsilon_4, \tag{5}$$

where ϵ_3 and ϵ_4 are small positive numbers and ideally zero. More details are discussed in Sect. 5.1. Detecting poisoned samples means that the defender applies a detection model $g(\cdot)$ such that for poisoned inputs x^*, it outputs positive, and for clean inputs x, it outputs negative with high probability, i.e.,

$$\mathbb{P}(g(x) = 0 | x \in \mathcal{D}) \geq 1 - \epsilon_5, \tag{6}$$

$$\mathbb{P}(g(x^*) = 1 | x^* \in \mathcal{D}^*) \geq 1 - \epsilon_6, \tag{7}$$

with small positive numbers ϵ_5 and ϵ_6 (ideally, $\epsilon_5, \epsilon_6 = 0$). The detailed description is discussed in Sect. 6.1.

5 Backdoor Defense by Training Attacker Imitator

This section describes a reverse-engineering-based defense method. The intuition behind this method is that if the defender can find a function to imitate the attacker's behavior (i.e., trigger insertion), then such a function can be used to reduce the effect of the backdoor on the DNN. In our approach, this imitator function is found by formulating an optimization problem. The following assumptions are introduced:

- The attacker generates the poisoned data distribution using an injection function as defined in Sect. 3.
- Without loss of generality, the attacker chooses only one attacker-chosen label. However, the proposed method is applicable to other attack strategies, such as having multiple attacker-chosen labels or source-label-specific backdoors [39], which will be discussed later in Sect. 5.1.
- The defender does not know if a given network is a BadNet or not.
- The defender only has access to a small set of clean samples.
- The defender does not have access to the original training data or knowledge of the trigger shape/location.
- The defender is also allowed to fine-tune and retrain the network.

In this section, we write simply $\mathcal{F}_b$ instead of $\mathcal{F}_b(.;\theta)$ for notational brevity.

5.1 Problem Formulation

Given a DNN that is possibly backdoored, the defender's objective is to find a function that transforms a clean input into a poisoned input. This function behaves as an emulation of the attacker and is called attacker imitator in the rest of this chapter. The performance of the attacker imitator could be measured by its statistical risk as follows:

$$R(\gamma) = \mathbb{E}[L(\mathcal{F}_b(\gamma(x)), y_t)], \tag{8}$$

where $\gamma(.)$ is the attacker imitator, and y_t is the attacker-chosen label that is unknown to the defender. L is a loss function that models the mismatch between the network output and the target label. The cross-entropy loss function is used in our experiments. The best possible attacker imitator is

$$R^* = \underset{\gamma \in \Gamma}{inf}\, R(\gamma), \tag{9}$$

where Γ is a class of all possible attacker imitator functions. Equation (9) cannot be directly evaluated since the input data probability distribution is unknown. However, a small set of clean samples is assumed to be available. Therefore, considering the attacker imitator to be a neural network with $\tilde{\theta}$ as its parameters, the problem

of finding the attacker imitator could be solved using empirical risk minimization (ERM) if the attacker-chosen label y_t was known:

$$\theta^* = \underset{\tilde{\theta}}{argmin} \sum_{i=1}^{N} L(\mathcal{F}_b(\gamma(x_i; \tilde{\theta})), y_t). \tag{10}$$

Evaluating (10) will result in a function that mimics the adversarial behavior of an attacker's trigger, but the function outputs might be very dissimilar to the original data (and the actual poisoned data). Hence, a cost term is added to (10) for penalizing dissimilarity between the attacker imitator's outputs and their corresponding clean inputs, i.e.,

$$\theta^* = \underset{\tilde{\theta}}{argmin}\{\sum_{i=1}^{N}\{L(\mathcal{F}_b(\gamma(x_i; \tilde{\theta})), y_t) + d(x_i, \gamma(x_i; \tilde{\theta}))\}\}, \tag{11}$$

where $d(.,.)$ is a function measuring the similarity of its inputs. If two inputs are similar, the output of d will be small.

Solving (11) requires having access to y_t. However, the defender does not have any knowledge about the target label. Therefore, an attacker imitator function is found by evaluating Eq. (11) for each label in the dataset's classification labels set (i.e., $\{l_1, \cdots, l_m\}$). The best performing attacker imitator in terms of ASR is utilized, and its corresponding label is considered the prediction of the attacker's intended target label.

5.2 Defense Methodology

We propose a method for solving the problem formulated in Eq. (11) for an image classification task in which an image in $R^{w \times h \times 3}$ (w is its width and h is its height) could be flattened to a 1-dimensional vector $x_i \in R^n$, where $n = 3wh$. Moreover, the architecture of the attacker imitator highly depends on the classification task. For image classification, this function should transform input images into poisoned ones. CNN is a candidate for this purpose as it can replicate any nonlinear transformation by considering enough filters and nonlinear activation functions in each layer. Based on these facts, the problem is reformulated as follows:

$$\begin{aligned}\theta^* = \underset{\tilde{\theta}}{argmin} \Bigg\{ \sum_{i=1}^{N} \Big\{ & \lambda_1 L(\mathcal{F}_b(\gamma(x_i; \tilde{\theta})), y_t) + \lambda_2 \|x_i - \gamma(x_i; \tilde{\theta})\|_2^2 \\ & -\lambda_3 MSSSIM(\gamma(x_i; \tilde{\theta}), x_i) \Big\} \Bigg\}, \end{aligned} \tag{12}$$

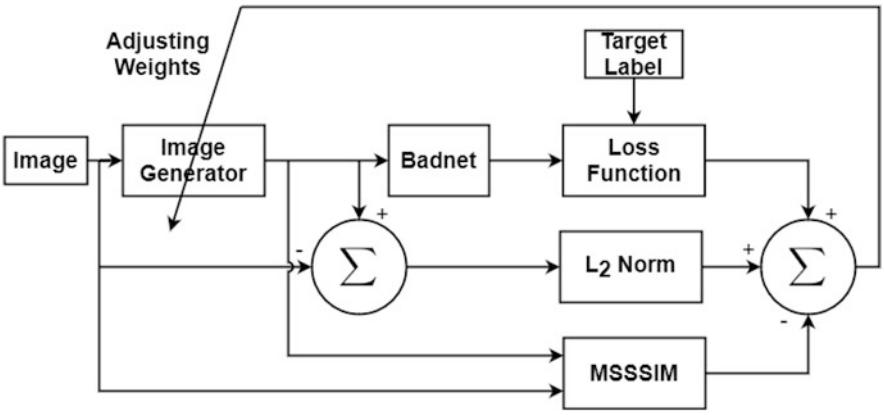


Fig. 2 The overall training procedure of the attacker imitator

where $\gamma(.;\tilde{\theta})$ is a CNN with parameters $\tilde{\theta}$ as an attacker imitator, x_i is a clean input, and $\lambda_1, \lambda_2, \lambda_3 \geq 0$ are tuning parameters. MSSSIM stands for multi-scale structural similarity index measurement [45], which is a perceptual image quality measurement to score similarity between two images. The output of MSSSIM is between -1 and 1, and a large value means that the two input images are similar. Therefore, increasing the MSSSIM value would be desired, which is attained by subtracting this value from the loss function. Also, an L_2 norm of the difference between the original and corresponding poisoned images is added to the loss function.

The overall training procedure of the attacker imitator is depicted in Fig. 2. The performance of the attacker imitator is evaluated by calculating the percentage of generated images that can successfully fool the network among all samples in the test dataset. We refer to this calculated percentage as generated attack success rate (GASR), and the closer this number is to 100%, the better the attacker imitator replicates the behavior of a successful attacker. Solving (12) for each classification label and calculating the corresponding GASR will provide a GASR profile. This profile can be used to detect whether the network is clean or a BadNet. If there exists an outlier in the GASR profile, the network is a BadNet, and the corresponding label is the attacker-chosen label. The outlier could be found based on the z-scores of GASR values. The z-score of GASR for each label i is calculated as

$$z_i = \frac{GA_i - \overline{GA}}{\sigma}, \tag{13}$$

where GA_i is the GASR of the label i, $\overline{GA}$ is the mean of GASR values, and σ is the standard deviation of all GASR values. Any z-score above a cut-off threshold is considered an outlier.

Algorithm 1 Attacker imitator training

procedure CALCULATE_GASR(F, a, S, $Label$)
  $Input \leftarrow a(S)$
  $Output \leftarrow F(Input)$
  return the number of elements in output classified to Label
end procedure
procedure MAIN($\mathcal{F}_b$, V, IT)
  $GASR = []$
  for $l \leftarrow 1$ to m **do**
    $\tilde{\theta} \leftarrow$ Initializing the attack imitator (γ) parameters
    for $i \leftarrow 1$ to n_{epochs} **do**
      for $j \leftarrow 1$ to $n_{batches}$ **do**
        $x_p \leftarrow \gamma(IT^j)$
        $Loss \leftarrow \lambda_1 L(\mathcal{F}_b(x_p), l) - \lambda_3 MSSSIM(x_p, IT^j) + \lambda_2 L_2(x_p, IT^j)$
        $\tilde{\theta} \leftarrow \tilde{\theta} - \eta \frac{\partial Loss}{\partial \tilde{\theta}}$
      end for
    end for
    $GASR \leftarrow CALCULATE\_GASR(\mathcal{F}_b, \gamma, V, l)$
  end for
end procedure

As mentioned earlier, the user only has access to a small set of clean samples. The user uses a subset of the clean samples called the imitator training set for training the attacker imitator and utilizes the rest of the data called the validation set to calculate CA, ASR, and GASR. Training the attacker imitator is presented in the $MAIN$ procedure in Algorithm 1. The inputs to this procedure are the BadNet, the validation set (V), and the imitator training set (IT). For an m-label dataset, an attacker imitator (shown by γ) is trained. The training could be done using any gradient descent method with a learning rate η. Once the training is done for all batches ($n_{batches}$), GASR is calculated for the attacker imitator (γ) trained with the predicted target label (l) using the defined procedure $CALCULATE\_GASR$. The inputs to this procedure are the BadNet (F), the attacker imitator (a), the set of samples (S), and the predicted target label ($Label$). As long as we consider a single attacker-chosen label for the BadNet, images with classification labels equal to the attacker-chosen label are removed during training and validation. It should be mentioned that (12) could be solved for other attack strategies. For instance, in the case of multiple attacker-chosen labels, the GASR profile will have multiple outliers corresponding to the attacker-chosen labels. Another challenging attack strategy is source-label-specific backdoor [39], in which the attacker's goal is to misclassify inputs corresponding to specific labels rather than all the labels. This attack could be taken into consideration by solving (12) for misclassifying samples from a specific label to an attacker-chosen label. This will result in a GASR profile for each classification label that is checked for its outliers.

The next step is to use the predicted attacker-chosen label and attacker imitator to make the BadNet robust against the backdoor attack with minimal effects on clean classification accuracy. These requirements can be embedded into an optimization problem as

$$\theta^*_{Bad} = \underset{\theta_{Bad}}{argmin} \left\{ \sum_{\substack{i=1 \\ (x_i, y_i) \in S}}^{N} L(\mathcal{F}_b(\gamma(x_i; \tilde{\theta}); \theta_{Bad}), y_i) + \sum_{i=1}^{N} L(\mathcal{F}_b(x_i; \theta_{Bad}), y_i) \right\}, \tag{14}$$

where y_i is the correct label for x_i, θ_{Bad} corresponds to the BadNet's parameters, and $S = \{(x, y)|y \neq y_t\}$, where y_t is the attacker-chosen label. The first term in Eq. (14) is responsible for unlearning the backdoor. This has been done by generating poisoned inputs with correct classification labels. The second term is added to ensure that the backdoor removal will not change clean classification accuracy; therefore, the CA of the network is mostly retained.

5.3 Experimental Setup

Our experiments for each dataset consist of four steps described as follows:

1. In the first step, a BadNet with a specific trigger pattern is generated. To achieve this goal, a network is trained on clean training samples with high CA. Then, 10% of images are poisoned with the desired trigger pattern, and training is continued on 90% clean and 10% poisoned images for more epochs to achieve high ASR.
2. GASR profile is calculated for the BadNet by solving Eq. (12) for all classification labels in the dataset.
3. Based on the GASR profile, we detect if the network is backdoored and what the attacker-chosen label is likely to be.
4. In the last step, the attacker imitator and the attacker-chosen label found in the previous steps are used to robustify the network against the attack.

To evaluate the effectiveness of our method, four BadNets have been considered covering various image classification tasks (e.g., object recognition, face detection, and traffic sign recognition) and various trigger patterns. These BadNets are explained in detail in the following subsections, and for simplicity, they are called Badnet-CW, Badnet-GY, Badnet-YS, and Badnet-CR.

5.3.1 Badnet-CW

This BadNet is trained on the CIFAR-10 dataset [46] for object recognition. Network in Network (NiN) [47] has been chosen for its architecture, and a white square in the bottom-left corner of inputs is used as the trigger. The trigger shape and location are standard configurations considered in [16, 31, 32].

5.3.2 Badnet-GY

This BadNet is trained on the German Traffic Sign Recognition Benchmark (GTSRB) dataset [48], which has 43 classes. The Badnet-GY's architecture is the same as the architecture used in Neural Cleanse [31] for the GTSRB dataset. The network has six CNN layers and two dense layers. This network will be called DeepGT in the rest of this chapter. The trigger shape in this setting is a yellow square with an arbitrary location in each image. This trigger has been proposed by [49] to show the limitations of Neural Cleanse for finding the attacker-chosen label and trigger shape.

5.3.3 Badnet-YS

For the face detection task, a BadNet is trained on the YouTube Face Database [50], and for its architecture, a deep network with four CNN layers and three dense layers has been used as in [31]; this network will be called DeepID. The trigger is chosen to be sunglasses with constant size and color. This trigger has been chosen to cover more pixels. Also, this trigger will cover key points in the image, which makes it more difficult to reverse-engineer the trigger [49].

5.3.4 Badnet-CR

This BadNet's architecture is ResNet-18 [51] and is trained on the CIFAR-10 dataset [46]. The trigger for this network is a combination of a yellow square on the top-right corner of the image and a red square on the bottom-right corner. The backdoor is triggered when both patterns appear together in a poisoned image. For training the network with this more complicated trigger, we initially train the network on a clean training dataset. After that, 10% of images are chosen to generate three sets of data: (1) only a red square is added to the images, and their labels remain the same as their clean labels, (2) the same as the previous set, but a yellow square is added instead of the red square, and (3) the combination of the two triggers is added, and their labels are set to the attacker-chosen label. Then, this dataset is augmented to the original training dataset, and training is performed for more epochs. This trigger has been designed to affect more neurons in the network.

Table 3 provides a summary of all the BadNets with their architectures, their training datasets, and the shape of triggers. In Table 4, BadNet training samples (used to train the BadNets), imitator training samples (used to train the attacker imitator network), validation samples (used to calculate CA, ASR, and GASR), and the number of labels for each dataset are shown. The first two columns of Table 8 show the CA and ASR of each BadNet. Examples of each trigger shape and the corresponding dataset images are shown in the top row of Fig. 4, in which the poisoned images from the left correspond to Badnet-CW, Badnet-GY, Badnet-YS, and Badnet-CR, respectively.

Table 3 Dataset, network architecture, and trigger shape corresponding to each BadNet configuration

Name	Dataset	Model	Trigger
Badnet-CW	CIFAR-10	NiN	White square
Badnet-GY	GTSRB	DeepGT	Moving yellow square
Badnet-YS	YouTube Face	DeepID	Sun glasses
Badnet-CR	CIFAR-10	ResNet-18	Red & yellow squares

Table 4 Number of samples and labels for each dataset

Dataset	BadNet training	Imitator training	Validation	# of classes
CIFAR-10	50,000	5000	5000	10
GTSRB	35,288	12,630	3921	43
YouTube face	102,640	12,830	12,830	1283

Table 5 Attacker imitator architecture for Badnet-CW and Badnet-CR

Layer	# Channels	Filter size	Stride	Padding	Activation
Conv2d	512	3×3	1	1	ReLU
Conv2d	256	3×3	1	1	ReLU
Conv2d	128	3×3	1	1	ReLU
Conv2d	64	3×3	1	1	ReLU
Conv2d	32	3×3	1	1	ReLU
Conv2d	16	3×3	1	1	ReLU
Conv2d	8	3×3	1	1	ReLU
Conv2d	3	3×3	1	1	ReLU

5.4 Experimental Results

5.4.1 Attacker Imitator Configuration

As outlined in Sect. 4, the attacker imitator transforms the input image into a poisoned image with the same dimensions as the input image. The attacker imitator network parameters are trained using the optimization in (11) to achieve high performance when applied to the training samples. The attacker imitator architecture depends on the classification task to reconstruct the input. Since the image classification task is considered in this chapter for experimental evaluations, CNNs are chosen for the architecture of the attacker imitators. Analogous to the considerations for network architectures in classification tasks that the CNNs should have enough layers and filters to extract key features of images, the attacker imitator's network architecture should be chosen based on the dataset complexity. Based on these observations, the attacker imitator architecture used for Badnet-CW and Badnet-CR is shown in Table 5, and the architectures for Badnet-GY and Badnet-YS are shown in Tables 6 and 7, respectively.

Table 6 Attacker imitator architecture for Badnet-GY

Layer	# Channels	Filter size	Stride	Padding	Activation
Conv2d	32	3×3	1	1	ReLU
Conv2d	128	3×3	1	1	ReLU
Conv2d	256	3×3	1	1	ReLU
Conv2d	256	3×3	1	1	ReLU
Conv2d	256	3×3	1	1	ReLU
Conv2d	32	3×3	1	1	ReLU
Conv2d	3	3×3	1	1	ReLU

Table 7 Attacker imitator architecture for Badnet-YS

Layer Type	# Channels	Filter size	Stride	Padding	Activation
Conv2d	128	5×5	1	2	ReLU
Conv2d	128	5×5	1	2	ReLU
Conv2d	64	3×3	1	1	ReLU
Conv2d	3	3×3	1	1	ReLU

5.4.2 BadNet vs. Benign Network Detection

In this part, we evaluate the performance of our BadNet detection methodology discussed in Sect. 4. To have a fair comparison, for each of the BadNets, we have trained a benign network with the same architecture on the same dataset. Then, GASR profiles were calculated for the BadNet and the corresponding benign network.

In Fig. 3, the z-scores of GASR profiles for each BadNet and the corresponding benign network are depicted in blue and red, respectively. Additionally, the corresponding cut-off threshold is depicted in cyan. It should be mentioned that the cut-off threshold is dependent on the number of samples in a set [52]; therefore, a cut-off threshold of 2 is considered for the CIFAR-10 dataset and 3 for the larger datasets.

The results presented in Fig. 3 indicate that for all BadNets, there is an outlier in the GASR profile corresponding to the attacker-chosen label. For Badnet-CW and Badnet-CR, the attacker-chosen labels used by the attacker are 2 and for Badnet-GY and Badnet-YS are 0, which are detected by our method correctly. This shows that our method is not dependent on the number of classification labels and can be used for any dataset and network architecture. It should be mentioned that for all the experiments, we have chosen the same coefficients for (12) without any modification during the training procedure.

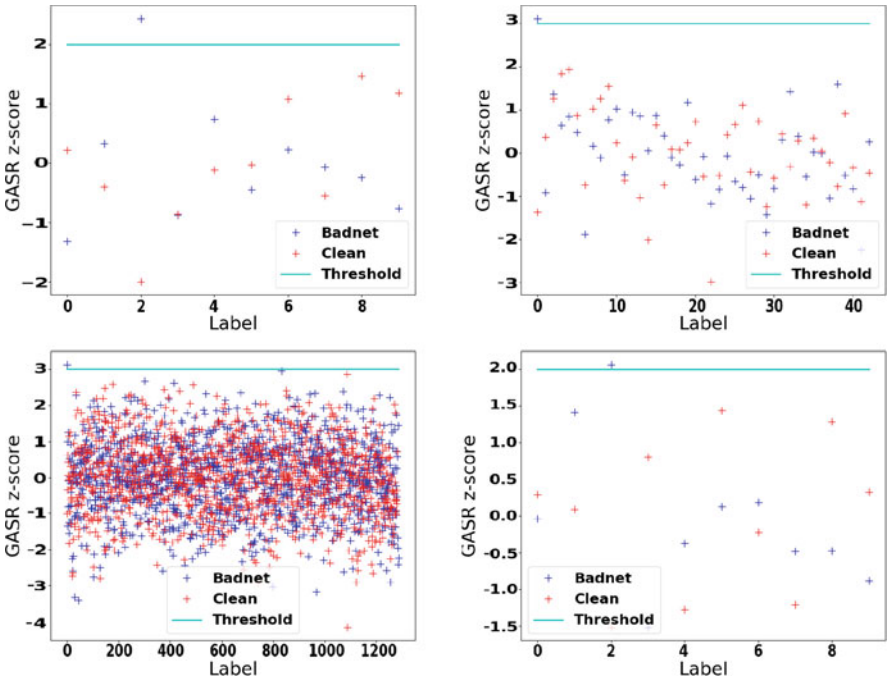


Fig. 3 GASR z-scores profiles for the BadNets and corresponding benign networks. Top left: Badnet-CW. Top right: Badnet-GY. Bottom Left: Badnet-YS. Bottom right: Badnet-CR

5.4.3 Fine-Tuning the Network

Once the attacker-chosen label is found, the corresponding attacker imitator generates poisoned images from the clean attacker imitator training set. In Fig. 4, for each of the BadNets, we have illustrated the poisoned image with the original trigger in the top row and with the imitator-generated trigger in the bottom row. Additionally, the columns from the left correspond to Badnet-CW, Badnet-GY, Badnet-YS, and Badnet-CR, respectively.

By observing the attacker imitator outputs, it can be seen that the attacker imitator can find the positions of the triggers correctly. For Badnet-GY, in which the attacker does not use a fixed position for the trigger, the generated trigger also does not have a fixed trigger location. In all the cases, the generated trigger does not replicate the actual trigger pattern, which is expected since the attacker imitator is found by solving an optimization problem to imitate the behavior of the attacker. Therefore, the attacker imitator will mimic the underlying effect of the trigger on the BadNet, and as the main concern is the removal of the backdoor from the network, finding the exact trigger pattern is not crucial for the main goal, which is to reduce the backdoor effect on the network.

Fig. 4 The columns from the left correspond to Badnet-CW, Badnet-GY, Badnet-YS, and Badnet-CR, respectively. Top row: The poisoned images with the trigger used by the attacker. Bottom row: The outputs of the attacker imitator

Table 8 Clean classification accuracy and ASR comparison of BadNets before and after fine-tuning them with our method and Neural Cleanse

	Backdoored		Cleaned		Cleaned by neural cleanse	
	CA (%)	ASR (%)	CA (%)	ASR (%)	CA (%)	ASR (%)
Badnet-CW	87.18	97.53	85.14	0.95	84.12	1.04
Badnet-GY	95.56	100	95.11	0.0	95.24	12.39
Badnet-YS	97.88	98.53	94.4	0.0	95.74	38.09
Badnet-CR	85.44	100	84.1	1.6	82.4	4.8

As was mentioned in Sect. 4, the attacker imitator found by solving (12) given attacker-chosen label is used to generate poisoned images from the attacker imitator training set. The main advantage of doing so is that the clean labels for those samples are known. In the next step, the optimization problem in (14) should be solved by training the BadNet on a new dataset consisting of two main components: (1) poisoned samples with clean labels made from clean samples (the first term in Eq. (14)) and (2) clean samples with their clean labels (the second term in Eq. (14)). The poisoned samples are generated from samples whose clean labels are not equal to the attacker-chosen label.

After robustifying all the BadNets using (14), the results are reported in Table 8. In this table, CA and ASR of the BadNets are reported in the first two columns from the left. The third and fourth columns show our method's CA and ASR of the fine-tuned network. Also, to have a fair comparison with Neural Cleanse, CA and ASR of the fine-tuned networks using Neural Cleanse have been reported in the last two columns.

It can be seen that the robustification of the BadNets using our approach is successful in reducing the ASRs to 0 in three cases and 1.6% for Badnet-CR, which has a more complex trigger pattern. Moreover, it can be seen that after fine-tuning the network, the CA was not affected significantly, and the maximum reduction was 3.44%. Our method outperforms Neural Cleanse by achieving better ASRs. Specifically, Neural Cleanse is not successful for Badnet-GY, in which the trigger position is not fixed, and Badnet-YS, in which the trigger size is not small.

6 RAID—An On-line Detection Method

This section discusses the detection method called Removing Adversarial backdoors by Iterative Demarcation (RAID). The key differences between RAID and the previous method discussed in Sect. 5 are: (1) The previous method is purely off-line, whereas RAID uses both on-line and off-line models. (2) The previous method removes backdoors during off-line fine-tuning, whereas RAID rejects poisoned inputs during on-line implementation. (3) The previous method belongs to the reverse-engineering-based approach, whereas RAID belongs to the novelty-detection-based approach. RAID takes advantage of the on-line streaming data and therefore reduces reliance on off-line validation data and restrictive assumptions on triggers. While RAID may perform ineffectively initially because the on-line data are scarce at the beginning, its performance improves on-line over time and enables accurate detection of poisoned inputs as the on-line data are accumulated.

RAID first collects suspicious on-line samples that are highly likely to be poisoned using novelty detection models, which are trained off-line. Among the collected samples, RAID uses an anomaly detector to select the samples that are more likely to be poisoned than others. RAID trains a binary classifier using these selected data and the clean validation data. The trained binary classifier is used to determine if an on-line input is poisoned or not. As more suspicious on-line samples are collected, the binary classifier is updated, and the performance is improved. This repeated update to improve backdoor detection accuracy can be viewed as iterative demarcation. RAID uses two dimension-reduction methods to simplify the computational complexity. One is a feature extractor that compresses raw data into low-dimensional signature features, and the other is a dimension-reduction function such as PCA [53] that further reduces the feature dimensions. These two dimension-reduction methods result in a reduction of computational complexity, ensuring that RAID can be implemented in real time.

6.1 Problem Formulation

The scenario considered by RAID is as follows: the user outsources a training task to a third party, which returns a backdoored DNN $\mathcal{F}_b(.;\theta)$ (written for notational

brevity simply as $\mathcal{F}_b$) to the user. The defender needs to build a detection model to protect the user from backdoor attacks. The attacker's goal is to make $\mathcal{F}_b$ have high CA and ASR. The defender's goal is to build the detection model $g(\cdot)$ to lower ASR of $\mathcal{F}_b$ while maintaining the CA. Mathematically, the defender wants $g(\cdot)$ to output "clean" with a high likelihood for clean samples x and "poisoned" with a high likelihood for poisoned samples x^*, i.e., to satisfy (6) and (7).

The attacker is considered to have complete control over the training dataset and process. However, the attacker neither has access to the user's validation dataset nor can change the model structure after training. The defender is considered to have a small set of clean validation data V (e.g., around 2% of the training dataset size). The defender has no prior information about the triggers or attacker-chosen label(s).

6.2 Detection Algorithm

The development of RAID was inspired by the observation that a DNN can be decomposed as a feature extractor C_b and a decision function $\mathcal{G}_b$, i.e., $\mathcal{F}_b = \mathcal{G}_b \circ C_b$ [54]. C_b reduces the raw data dimension and extracts features at higher abstraction levels. $\mathcal{G}_b$ maps the combinations of the features into corresponding outputs. The hypothesis is that the backdoor effect is created through a "logic" component encoded in $\mathcal{G}_b$ (i.e., specifying what features in the trigger should result in the attacker-chosen labels). RAID sets the output layer of $\mathcal{F}_b$ as $\mathcal{G}_b$ and all the previous layers as C_b.

The overall algorithm of RAID is shown in Algorithm 2. In **Given**, the defender has a backdoored network $\mathcal{F}_b$ decomposed into C_b and $\mathcal{G}_b$ and a small clean validation dataset V with the corresponding labels L. In **off-line training**, the defender obtains validation data features by feeding samples of V into C_b. Then, a new classifier $\mathcal{G}_n$ is trained with the extracted features V_F and the corresponding labels L. Since $\mathcal{G}_n$ is trained only on clean data, it is highly likely that $\mathcal{G}_n \circ C_b$ and $\mathcal{G}_b \circ C_b$ behave similarly on clean inputs but differently on poisoned inputs. A preprocess function (e.g., PCA) further reduces the feature dimension to obtain V_{FP}. V_{FP} is used to train a novelty detector $\mathcal{N}$ independent of $\mathcal{G}_n$ to detect inputs whose dimension-reduced features differ from dimension-reduced features of the clean validation data.

The on-line detection and retraining are shown in both Fig. 5 and **On-line Detection and Update** in Algorithm 2. The on-line implementation is comprised of a front-end part and a back-end part that may be implemented in a parallel manner. In the front-end part, C_b takes x as the input and outputs x_F. The preprocess function takes x_F as the input and outputs x_{FP}. If $g(x_{FP}) = 0$, i.e., if x is classified to be clean, then $\mathcal{F}_b(x)$ will be trusted. Otherwise, $\mathcal{F}_b(x)$ should not be trusted. $g(\cdot)$ is initialized as $g(x) = 0$ for all x and is then updated at a pre-specified frequency. In the back end, $\mathcal{N}$ and $\mathcal{G}_n$ determine if x is a "suspected" poisoned sample. If either $\mathcal{N}(x_{FP}) = 1$ (i.e., detected as different from the clean validation data) or $\mathcal{G}_n(x_F) \neq \mathcal{G}_b(x_F)$, then x will be collected into the anomalous dataset $\mathcal{A}$. $\mathcal{A}$ may

Algorithm 2 On-line detection algorithm of RAID

Given

Validation data = (V, L) and backdoored network $\mathcal{F}_b = \mathcal{G}_b \circ C_b$

Off-line Training

Extract features of validation data: $V_F \leftarrow C_b(V)$
Train a new classifier: $\mathcal{G}_n \leftarrow train(\mathcal{G}_n, (V_F, L))$.
Reduce feature dimension: $V_{FP} \leftarrow preprocess(V_F)$
Train a novelty detector: $\mathcal{N} \leftarrow train(\mathcal{N}, V_{FP})$

On-line Detection and Update

$g(\cdot) \equiv 0$ (clean), count = 0, $\mathcal{A} = \{\}$, and determine window_size = w_0
while True **do**
  count = count + 1
  Receive New Input x

  ### Make a Prediction on x with $\mathcal{F}_b$ ###
  Extract Input Feature: $x_F \leftarrow C_b(x)$
  Make a Prediction: $l \leftarrow \mathcal{G}_b(x_F)$

  ### Check If x is Poisoned (Front End) ###
  Reduce Feature Dimension: $x_{FP} \leftarrow preprocess(x_F)$
  if $g(x_{FP}) == 0$ **then**
    l is the label for x (Clean with High Probability)
  else
    l is not the true label for x (Poisoned with High Probability)
  end if

  ### Determine If x Should Be Collected as Anomalous Data and Used to Update $g(\cdot)$ (Back End) ###
  $l' \leftarrow \mathcal{G}_n(x_F)$
  if $l \neq l'$ or $\mathcal{N}(x_{FP}) == 1$ **then**
    Collect x_{FP}: $\mathcal{A} \leftarrow \mathcal{A} \cup \{x_{FP}\}$
  end if

  ### Updating $g(\cdot)$ ###
  if count % window_size == 0 **then**
    Purify $\mathcal{A}$: $\mathcal{A}^* \leftarrow purify(\mathcal{A})$
    Update the Binary Classifier: $g \leftarrow train(g, (\{\mathcal{A}^*, 1\} \cup \{V_{FP}, 0\}))$
  end if
end while

contain a few false positives. Therefore, an anomaly detector is used to purify $\mathcal{A}$ to obtain $\mathcal{A}^*$. $g(\cdot)$ is trained from scratch using V_{FP} and $\mathcal{A}^*$ at a pre-specified frequency. The defender can modify the window size (window_size in Algorithm 2) to determine the update frequency. Data in $\mathcal{A}^*$ is labeled as poisoned, and data in V_{FP} is labeled as clean.

$\mathcal{G}_n$ is a neural network with at most two hidden layers. This allows the structure of $\mathcal{G}_n \circ C_b$ to be close to the original backdoored network $\mathcal{F}_b = \mathcal{G}_b \circ C_b$. Therefore, they may show similar behavior on clean samples. Additionally, with such an architecture, the number of training parameters of $\mathcal{G}_n$ is small, reducing the clean

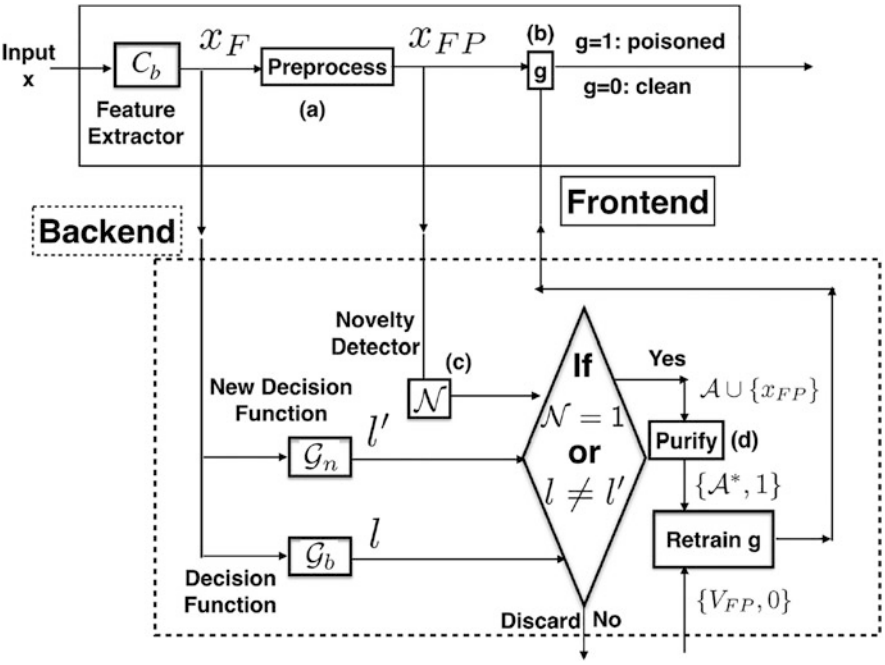


Fig. 5 Pipeline of the detection algorithm (on-line part)

validation samples required to train $\mathcal{G}_n$. In real-world cases, the defender may only have a small clean validation data. RAID uses SGD as the optimizer with a learning rate set to 0.01 and a cross-entropy-based loss function with default hyperparameter settings. PCA with the top 40 principals from the validation data features is used as the preprocess function (i.e., (a) in Fig. 5); hence, it reduces the data dimension without losing too much information and amplifies the spectral signature of the clean data. However, other dimension-reduction functions can also be considered, such as SVD and factor analysis from scikit-learn [55]. The PCA model in RAID cannot highlight the spectral signature of the triggers because it was trained only on a clean validation dataset.

Local outlier factor (LOF) is used as the novelty detector $\mathcal{N}$ (i.e., (c) in Fig. 5) since the training process is unsupervised and in real time. The anomaly detector (d) in the figure is also LOF for the same reasons. However, other outlier detectors (e.g., [56–59]) may be considered if the training process is also unsupervised and in real time. RAID uses SVM as the binary classifier $g(\cdot)$ in (b) in the figure. $g(\cdot)$ must be simple so that the training time is short. SVM satisfies this requirement. Other models, such as a neural network, may take a much longer training time than the SVM. However, the binary classifier is also open to other models that can be trained in real time.

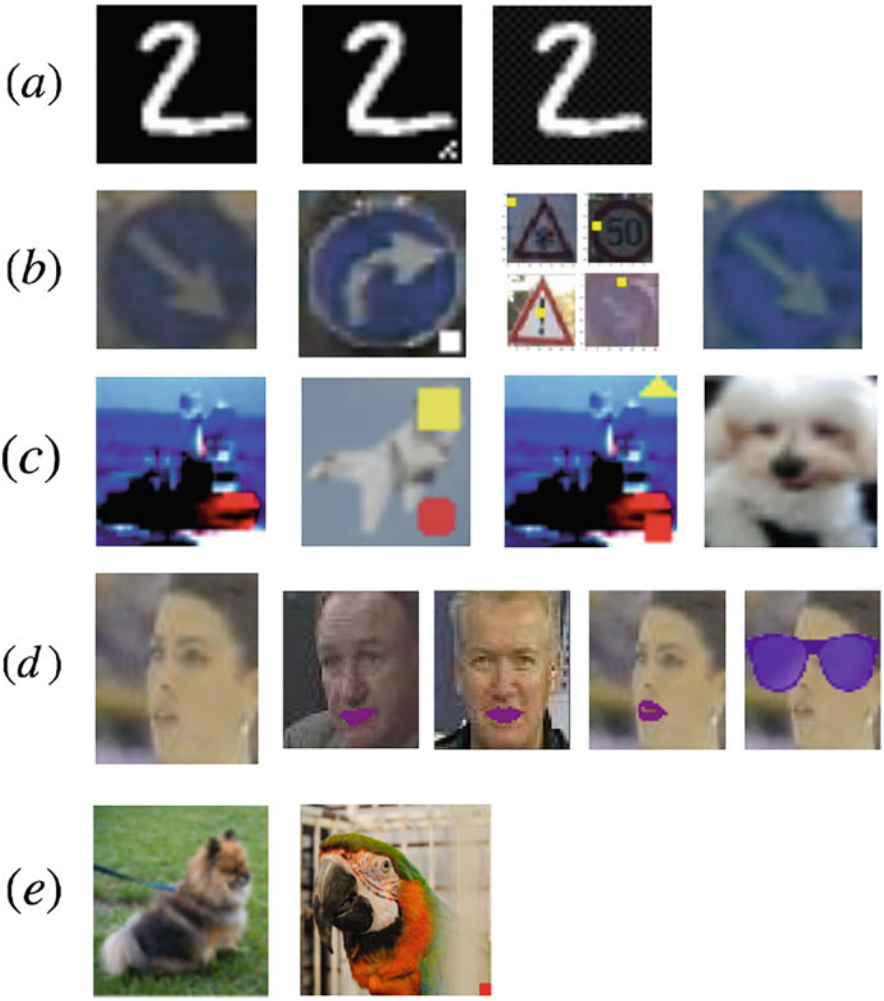


Fig. 6 The first column shows sample clean images for all the datasets. The other columns show all the triggers used in the experiments

6.3 Experimental Setup

RAID was implemented on five popular datasets: MNIST [60], German Traffic Sign Benchmarks (GTSRB) [48], CIFAR-10 [46], YouTube Face [50], and ImageNet [61]. Figure 6 shows different triggers used in the experiment. Several network architectures are used based on related works [18, 31, 47, 62–64]. The original paper of RAID [41] lists all the network architectures. Each dataset contains three

Table 9 Dataset Size. Columns 2–4 show the average number of samples per class

Dataset	Train	Valid	Test	Valid/train	# of classes
MNIST	5500	100	1000	1.8%	10
GTSRB	820	50	268	6.0%	43
CIFAR-10	5000	30	500	0.6%	10
YouT. Face	81	3	10	3.7%	1283
ImageNet	1200	3	25	0.2%	1000

Table 10 $\mathcal{F}_b$'s architecture for case (**a**)

Layer	Channels	Filter size	Stride	Activation
Conv2d	16	5×5	1	ReLU
MaxPool	16	2×2	2	–
Conv2d	32	5×5	1	ReLU
MaxPool	32	2×2	2	–
Linear	512	–	–	ReLU
Linear	10	–	–	–

parts: training part (Train), validation part (Valid), and testing part (Test), as shown in Table 9. Considering that the MNIST, GTSRB, and ImageNet datasets are unbalanced among classes, Table 9 reports the average number of samples per class, which is the dataset size divided by the total number of classes. The training part was partially poisoned (i.e., around 10%) and used for training the backdoored networks. The clean validation part was used for training $\mathcal{N}$ and $\mathcal{G}_n$. The testing dataset was used for evaluating RAID. The poisoned samples are acquired by injecting the triggers into the clean testing images. An on-line dataset with size n and attack density p is obtained with $(1-p)n$ clean testing samples plus pn poisoned testing samples. Empirically, when 20% of the test data is poisoned, RAID achieves low ASR and high CA. There is not much gain after $p = 0.5$.

6.3.1 BadNet Trained on MNIST

*Case (**a**)* The second column of Fig. 6a shows the trigger pattern. The attacker-chosen label l^* was determined by the ground-truth label l:

$$l^* = (l+1) \bmod 10. \tag{15}$$

The network architecture is shown in Table 10. CA is 97.65%, and ASR is 96.3%.

*Case (**b**)* The trigger is shown in the third column in Fig. 6a, which is a dotted background and almost imperceptible. l^* is 0. The backdoored network has 89.35% CA and 100.0% ASR.

6.3.2 BadNet Trained on GTSRB

*Case (**c**)* The second column in Fig. 6b shows the trigger, which is a white box pattern. l^* is 33. CA is 96.41%, and ASR is 97.62%.

*Case (**d**)* The trigger is a moving square as shown in the third column in Fig. 6b with $l^* = 0$. CA is 95.26%, and ASR is 99.92%.

*Case (**e**)* Images passing through a Gotham filter will trigger $\mathcal{F}_b$, as shown in the fourth column in Fig. 6b. $l^* = 35$. CA is 94.49%, and ASR is 90.32%.

6.3.3 BadNet Trained on CIFAR-10

*Case (**f**)* The second picture in Fig. 6c shows the trigger, a combination of a box and a circle. $\mathcal{F}_b$ will output the attacker-chosen label 0 only when both the shapes appear on the input. CA is 88.6%, and ASR is 99.8%.

*Case (**g**)* The trigger is the combination of a triangle and a square, as shown in the third column of Fig. 6c. $l^* = 7$. CA is 88.83%, and ASR is 99.97%.

The last column in Fig. 6c shows another trigger, a small perturbation (one pixel at each corner). l^* is 0, CA is 82.44%, and ASR is 91.92%.

6.3.4 BadNet Trained on YouTube Face

*Case (**h**)* The trigger is sunglasses, as shown in the last column in Fig. 6d. l^* is 0. CA is 97.83%, and ASR is 99.98%.

*Case (**i**)* The trigger is red lipstick, as shown in the second column of Fig. 6d. l^* is 0. CA is 97.19%, and ASR is 91.43%.

*Case (**j**)* $\mathcal{F}_b$ has all three triggers: lipstick, eyebrow, and sunglasses, as shown in Fig. 6d. l^* is 4 for all the triggers. CA is 96.13%, and ASRs are 91.80%, 91.88%, and 100% on lipstick, eyebrow, and sunglasses.

*Case (**k**)* $\mathcal{F}_b$ has all three triggers as well. l^*, however, is 1 for lipstick, 5 for eyebrow, and 8 for sunglasses. CA is 96.08%, and ASRs are 91.11%, 91.10%, and 100% on lipstick, eyebrow, and sunglasses.

6.3.5 BadNet Trained on ImageNet

The trigger is a red box shown in the second column in Fig. 6e. $l^* = 0$. The network is DenseNet-121 [64]. CA is 72.14%, and ASR is 99.99%. This backdoor attack is used to evaluate the performance of RAID with different attack frequencies and validation dataset sizes.

6.3.6 Hyperparameter Setting

The contamination ratio is a hyperparameter defined in the LOF anomaly detector. A high contamination ratio means that the LOF will remove more samples, whereas a low contamination ratio means that the LOF will remove fewer samples. The contamination ratio was set to 0.2 for all the cases. Note that the contamination ratio is set by the defender. Thus, it is not equal to the proportion of outliers in the dataset. The proportion of outliers in the dataset is determined by the attacker, which is similar to the attack density p mentioned earlier. The number of neighbors (i.e., NN, which is another hyperparameter of LOF) was set to be 1. However, the defender can set any other values between 1 and 20. Empirically, it is observed that the CA and ASR of RAID with NN $=$ 1, 10, and 20 are comparable to each other, and there is not a distinctive advantage in choosing a higher NN; however, the lower NN makes the computation faster. The remaining parameters are set to typical default values.

6.4 Experimental Results

6.4.1 Performance of $\mathcal{N}$ and $\mathcal{G}_n$

Using both $\mathcal{N}$ and $\mathcal{G}_n$ can maximally identify poisoned samples. Table 11 shows the CA and ASR of using $\mathcal{N}$ alone, $\mathcal{G}_n$ alone, and both. Three dimension-reduction functions (i.e., PCA, TruncatedSVD, and FactorAnalysis from scikit-learn [55]) are utilized to train different $\mathcal{N}$. Since $\mathcal{G}_n$ does not use any dimension-reduced features, the three cases use the same $\mathcal{G}_n$. From the table, there is not much difference in the performance of $\mathcal{N}$ using PCA and SVD. Using FactorAnalysis leads to a higher ASR in some cases than the other two functions. However, we will see that the overall performance of $g(\cdot)$ using FactorAnalysis is also good. Additionally, the three functions are all fast enough for on-line usage. Therefore, if only $g(\cdot)$'s performance is considered, the three functions are almost equal, whereas if both $g(\cdot)$'s performance and $\mathcal{N}$'s performance are considered, PCA and TruncatedSVD provide better performance than FactorAnalysis. The performance of PCA and TruncatedSVD is almost equivalent. The table also shows that using $\mathcal{N}$ and $\mathcal{G}_n$ together will maximally catch poisoned samples (i.e., reduce ASR) and also increase false positives (i.e., reduce CA). However, in this initial filtering, the ASR should be weighed more than the CA. Additionally, an anomaly detector is used subsequently to reduce false positives further. In case (**e**), the ASR is still large even though $\mathcal{G}_n$ and $\mathcal{N}$ are used together because the trigger is more subtle. Rather than some specific patterns, case (**e**)'s trigger is a Gotham filter function. $\mathcal{G}_n$ and $\mathcal{N}$ are not sensitive to this trigger. But we will next show that RAID can still improve itself to reduce the ASR with on-line data further.

Table 11 Performance of $\mathcal{N} + \mathcal{G}_n$ on cases (a)–(i) for different dimension-reduction functions

	Classifier		PCA				TruncatedSVD				FactorAnalysis			
	$\mathcal{G}_n$		$\mathcal{N}$		$\mathcal{G}_n + \mathcal{N}$		$\mathcal{N}$		$\mathcal{G}_n + \mathcal{N}$		$\mathcal{N}$		$\mathcal{G}_n + \mathcal{N}$	
Case	CA	ASR	CA	ASR	CA	ASR	CA	ASR	CA	ASR	CA	ASR	CA	ASR
(a)	93.63	15.04	95.37	49.75	92.44	4.34	95.37	49.36	92.71	5.3	94.98	74.55	92.14	5.74
(b)	87.68	1.83	87.58	0.0	86.23	0.0	87.59	0.0	86.24	0	86.51	0	85.22	0
(c)	95.08	4.22	94.83	3.45	93.38	1.14	94.83	3.53	93.68	0.79	77.46	0.07	76.87	0.04
(d)	93.40	71.45	86.72	0	85.43	0	86.65	0	85.36	0	86.28	0	85.01	0
(e)	94.05	74.63	92.17	57.75	91.57	45.65	92.16	56.69	91.59	44.68	91.37	66.68	90.76	55.26
(f)	81.28	99.6	88.26	0.23	81.04	0.23	88.26	0.21	81.7	0.2	87.14	0.28	80.74	0.27
(g)	82.86	70	88.72	8.14	82.72	3.7	88.72	5.28	82.72	2.12	87.22	52.94	81.3	37.5
(h)	55.74	3.14	96.57	79.15	55.26	3.04	95.46	76.24	53.89	0.42	90.44	59.59	51.26	0.38
(i)	60.64	0.01	96.04	33.78	60.09	0.01	96.11	30.36	60.11	0.01	94.38	10.67	59.33	0.01

6.4.2 Performance of $g(\cdot)$

The first 40% of test data was used for on-line implementation and updating of the SVM, and the remaining 60% of test data was used for evaluating the performance of the backdoored network after employing the SVM. The binary SVM was initialized to output clean for all the inputs. Then, it was updated with a fixed window size, which is set to 10% of the test dataset size (therefore, the SVM can be updated 4 times in the considered test scenario). Each test input has an equal probability of being clean or poisoned. Table 12 shows the performance of RAID with PCA, SVD, and FactorAnalysis as the dimension-reduction function. From the table, RAID is effective with all the three dimension-reduction functions. Additionally, SVM helps reduce ASR while retaining high CA. Note that in case (**e**), the SVM still provides good performance, although the off-line models have high ASR (refer to Table 11). These results highlight the robustness of RAID. Table 13 lists how many poisoned samples are fed into the backdoored network and the size of $\mathcal{A}^*$ for training the SVM at each update when PCA was used as the dimension-reduction function. From the table, training a good SVM needs only a small set of poisoned samples. Since the performance of RAID using PCA, SVD, and FactorAnalysis is similar, we will only discuss the case when PCA is used as the dimension-reduction function for the following experiments.

6.4.3 Performance of the Anomaly Detector

We examine the performance of RAID with different contamination ratios. Table 14 shows the results. The classification accuracy drops significantly without the anomaly detector (i.e., the contamination ratio is 0). This is because the LOF does not remove any samples. Thus, $\mathcal{A}$ contains many false positives. When contamination ratio is 0, $\mathcal{A} = \mathcal{A}^*$. The SVM trained with such $\mathcal{A}^*$ will perform inefficiently. RAID shows comparable results when the contamination ratio is set to 0.1, 0.2, and 0.3. Both CA and ASR decrease, while the contamination ratio increases.

6.4.4 Multiple Triggers and Adaptive Attacks

The attacker may use multiple triggers to attack the backdoored network, i.e., cases **(j)** and **(k)**. The following attack scenarios are considered during on-line operation: (1) The attacker does not use any triggers. (2) The attacker uses only one of the triggers. (3) The attacker uses two of the triggers. (4) The attacker uses all the triggers. Scenario (1) is used for evaluating the performance of RAID when $\mathcal{A}^*$ only contains false positives. One can also consider scenario (1) as the case to evaluate RAID on a benign model. Scenarios (2), (3), and (4) are utilized to evaluate if RAID is effective with multiple triggers. The 4th updated SVM was tested. The results are shown in Table 15. From the table, RAID maintains high CA in all the scenarios and

Table 12 Performance of the backdoored network after using SVM with different dimension-reduction functions and after different numbers of updates (i.e., after 1st, 2nd, 3rd, and 4th updates)

BadNet			PCA				TruncatedSVD				FactorAnalysis			
Case		0th	1st	2nd	3rd	4th	1st	2nd	3rd	4th	1st	2nd	3rd	4th
(a)	CA	97.65	97.63	97.53	97.06	96.83	97.48	97.08	96.50	96.08	97.5	96.98	96.8	96.13
	ASR	96.3	38.4	11.35	3.38	1.15	9.66	2.25	0.76	0.5	9	2.71	1.96	1.21
(b)	CA	89.35	89.35	89.35	89.35	89.19	89.35	89.35	89.35	89.29	89.27	89.32	89.09	88.69
	ASR	100	0	0	0	0	0	0	0	0	0	0	0	0
(c)	CA	96.41	96.41	96.41	96.41	96.41	96.41	96.41	96.41	96.41	96.41	96.41	96.41	96.37
	ASR	97.62	1.17	0.97	0.29	0.26	0.69	0.54	0.42	0.34	0.91	0.5	0.35	0.35
(d)	CA	95.26	95.19	94.76	94.57	94.61	94.37	94.47	94.40	94.37	95.19	95.03	94.58	94.60
	ASR	99.92	0.55	0.23	0.21	0.19	0.14	0.18	0.13	0.09	0.7	0.26	0.22	0.14
(e)	CA	94.49	94.37	93.87	93.52	93.36	94.47	94.02	93.56	93.41	94.49	94.40	93.70	93.04
	ASR	90.32	20.26	10.16	7.44	4.61	25.94	12.35	8.81	5.7	45.31	24.25	10.09	2.75
(f)	CA	88.6	88.6	88.6	88.6	88.6	88.6	88.6	88.6	88.6	88.6	88.6	88.5	88.5
	ASR	99.88	0.05	0.01	0.03	0.01	0.03	0.03	0.03	0	0.03	0.01	0	0
(g)	CA	88.83	88.83	88.83	88.83	88.83	88.83	88.83	88.83	88.83	88.83	88.83	88.83	88.83
	ASR	99.7	0.4	0.2	0.13	0.1	0.4	0.2	0.2	0.06	3.1	0.7	0.4	0.2
(h)	CA	97.83	97.81	97.77	97.73	97.67	97.83	97.80	97.79	97.79	97.80	97.73	97.71	97.68
	ASR	99.98	8.96	2.84	1.07	0.48	6.69	3.02	1.23	0.53	2.53	1.54	0.61	0.42
(i)	CA	97.19	97.19	97.16	97.16	97.11	97.18	97.15	97.14	97.14	97.18	97.11	97.11	97.11
	ASR	91.43	6.67	4.35	3.83	3.24	6.17	3.96	3.27	2.97	4.98	3.70	2.84	2.53

Table 13 Size of $\mathcal{A}^*$ and the numbers of poisoned inputs that have appeared at each update

		0th	1st	2nd	3rd	4th
(a)–(b)	# of poi.	0	1000	2000	3000	4000
(a)	size of $\mathcal{A}^*$	0	217	411	609	812
(b)		0	226	439	666	894
(c)–(e)	# of poi.	0	1263	2526	3789	5052
(c)	size of $\mathcal{A}^*$	0	275	534	779	1033
(d)		0	295	575	870	1123
(e)		0	162	303	434	571
(f)–(g)	# of poi.	0	1000	2000	3000	4000
(f)	size of $\mathcal{A}^*$	0	213	424	635	846
(g)		0	214	412	614	826
(h)–(i)	# of poi.	0	1283	2566	3849	5132
(h)	size of $\mathcal{A}^*$	0	360	722	1086	1453
(i)		0	352	698	1044	1390

Table 14 Performance of RAID (the 4th update) with different contamination ratios

	Ratio = 0.0		Ratio = 0.1		Ratio = 0.2		Ratio = 0.3	
	CA	ASR	CA	ASR	CA	ASR	CA	ASR
(a)	86.35	0.03	96.90	1.7	96.83	1.15	94.48	0.15
(b)	87.69	0	89.35	0	89.19	0	89.02	0
(c)	92.57	0.05	94.41	0.65	94.41	0.26	94.34	0.23
(d)	94.24	0.06	95.72	0.25	94.61	0.19	94.47	0.10
(e)	88.53	0.19	94.04	13.77	93.36	4.61	93.0	2.42
(f)	56.63	0	88.6	0.05	88.6	0.01	88.6	0.01
(g)	62.03	0	88.83	0.13	88.83	0.1	88.83	0.1
(h)	92.64	0	97.76	1.81	97.67	0.48	97.60	0.24
(i)	92.23	0.85	97.19	4.94	97.11	3.24	97.07	2.58

has low ASR on triggers that the attacker has used. For scenario (1), although $\mathcal{A}^*$ contains only false positives, RAID still manages to have a high CA. For scenarios (2) and (3), RAID has high ASR on the second or third trigger. However, since the SVM is updated in real time, once the new triggers are used for backdoor attacks, $\mathcal{N}$ and $\mathcal{G}_n$ will detect them in the back end resulting in attack detection by the SVM in the next update, such as case (4). The only period in which the network is vulnerable to the new triggers is between the moment that a new trigger appears and the next SVM update. Overall, the results show the efficacy and robustness of RAID.

6.4.5 Experiments on Hyperparameters

The first experiment is to evaluate RAID with different numbers of principal components. A backdoored network with small perturbations (only one pixel at each corner) as the trigger (i.e., the third $\mathcal{F}_b$ in the CIFAR-10 case) was trained, which has

Table 15 RAID performance on dynamic attacks (j)–(k)

Case/attack		Net.	(1)	(2)	(3)	(4)
(j)	CA	96.13	96.10	96.05	95.83	95.88
	ASR1	91.80	91.78	3.84	3.72	3.79
	ASR2	91.88	91.85	64.36	2.22	2.27
	ASR3	100	100	100	100	0.21
(k)	CA	96.08	96.05	95.99	95.90	95.88
	ASR1	91.11	91.11	2.77	2.84	3.21
	ASR2	91.10	91.10	89.88	0.99	0.94
	ASR3	100	100	99.65	95.82	0

82.44% CA and 91.92% ASR. The first two pictures in Fig. 7 show the performance of RAID with different numbers of principal components. From the pictures, all the plots show significant drops in ASR at different rates. Using more principal components results in a faster reduction in ASR. Using fewer principal components results in a small drop in CA (i.e., around 1%). With fewer principal components, the dimension-reduced poisoned data features are closer to the dimension-reduced clean data features. Thus, $\mathcal{A}^*$ may contain more false positives, which increases the training noise and leads to the degradation of CA. The number of principal components should be from 20% to 40% of the original feature dimension.

The second experiment is to evaluate RAID with a different attack frequency/-density (i.e., the probability p of an input being poisoned). Note that the attack frequency/density is determined by the attacker. Therefore, it is different from the contamination ratio. We tested RAID on the ImageNet dataset because we also want to see if RAID is efficient on a large-scale dataset. The backdoored model is DenseNet-121 with 72.14% CA and 99.99% ASR. The dataset and trigger are shown in Fig. 6e. The middle two pictures in Fig. 7 show the effectiveness of RAID on ImageNet under different attack densities. It is seen that ASR reduces faster when attack density is higher (i.e., more poisoned inputs are fed into the network). When attack density is 0 (meaning $\mathcal{A}^*$ has only false positives), the CA is still high.

RAID was also tested with 1 or 2 images per class on ImageNet to see if the clean validation dataset could be even smaller. Although low ASR is achieved with one image per class, CA degrades (the last two pictures in Fig. 7). This is because the novelty detector $\mathcal{N}$ and the new classifier $\mathcal{G}_n$ generate many false positives due to a lack of training data. Therefore, $\mathcal{A}^*$ may contain many false positives, which increases the training noise and leads to the degradation of CA (i.e., the SVM is trained with bad training samples).

The last experiment is to evaluate the performance of RAID when the SVM is updated at different frequencies. During the period between two updates, the backdoored network might be exposed to an attack if new triggers are applied. Therefore, reducing this period (increasing update frequency) can help further mitigate the threat of new triggers. The user needs to set a window size for the update. For example, if the window size is 1000, the SVM will be updated once there are 1000 new inputs into the network. Figure 8 shows the performance of

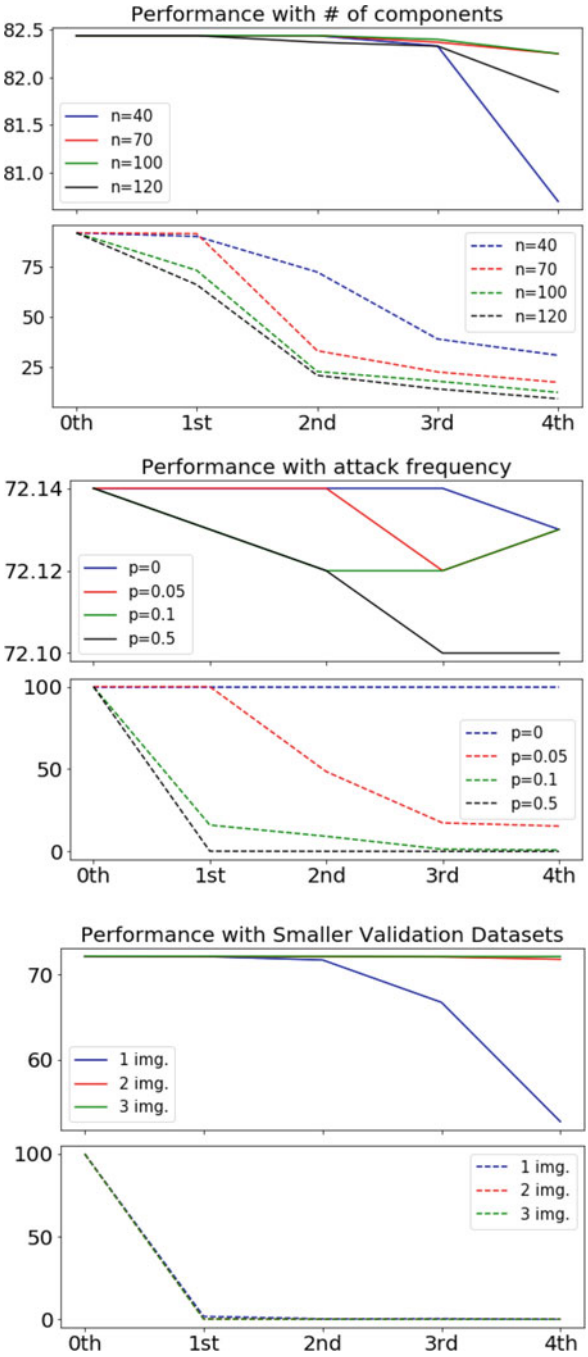


Fig. 7 Solid lines in all the pictures: CA. Dashed lines in all the pictures: ASR. X-axis: the number of updates. n: the number of PCA components. p: the probability of a sample being poisoned. For the last two pictures: the red plots overlap the other plots and are not visible

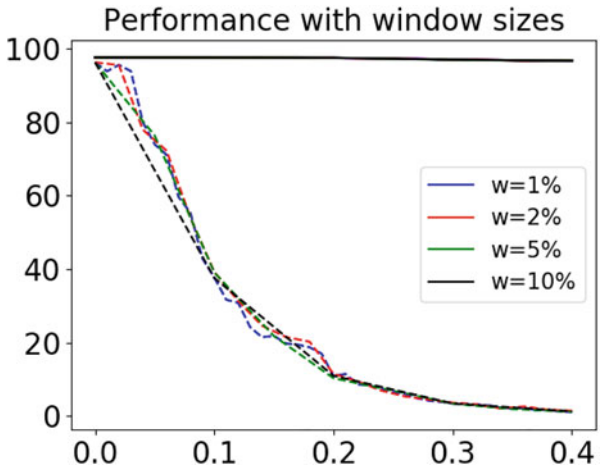


Fig. 8 X-axis: ratio of test data size used in RAID to the total test data size. Solid lines: CA. Dashed lines: ASR. w: window size/test data size

RAID with different update frequencies under case (**a**). RAID shows consistently good performance after increasing the update frequency (i.e., reducing window size). Since training the SVM is quick (<1 s), RAID can be used effectively during on-line operation.

6.4.6 More Advanced Attack

[20] propose a backdoor attack with sample-specific triggers. The example is shown in Fig. 9. It can be seen that the trigger remains invisible in the image. We use a subset of the ImageNet dataset as the testing data. The backdoored model has 78% CA and 100% ASR. The model architecture is ResNet-18 [51]. After the 4th update, RAID reduces the ASR to 0.4% and keeps CA close to 78%.

The attacker may try to minimize the feature-level outputs between poisoned data and corresponding clean data to bypass RAID. However, the difference between clean and poisoned data always exists and must be represented in hidden layer outputs. Otherwise, if the hidden layer outputs are identical for clean and poisoned samples, the network outputs should then also be the same. This cannot be true since the network outputs the attacker-chosen label for the poisoned sample and the ground-truth label for the clean sample. Although the difference may be small for one hidden layer, the cumulative difference for multiple hidden layers becomes significant and observable. RAID can still be applied by changing the input of its novelty detector and the binary classifier to include multiple hidden layer output features.

As seen above, RAID fuses several simple models (i.e., simple neural networks, novelty detector, anomaly detector, dimension-reduction function, and binary clas-

Fig. 9 Sample-specific trigger. Left: benign image. Middle: poisoned image. Right: the corresponding trigger

sifier) to reduce the ASR caused by the attacker. It requires only a small clean validation dataset, which is feasible to acquire in real-world applications.

7 Benign Applications of the Backdoor Phenomena

While we have considered backdoor-based attacks in this chapter, it is to be noted that backdoors can also be used for benign purposes such as the protection of intellectual properties. One example is using the backdoors as watermarks [65]. To train an accurate neural network model, the trainer needs to invest considerable cost and effort to collect high-quality data, label the data, buy/rent computational resources, and tune the model hyperparameters. Therefore, it is critical to find a way to protect the intellectual property of the models. Similar to watermark injection into documents, neural network models can also be injected with watermarks. Backdoor-based watermarks are one option for this purpose. The trainer injects backdoors into the trained network so that any other network copied based on this model can be recognized by presenting it with the poisoned inputs. Other models cannot correctly predict the outputs since they do not know the trigger information. The model's performance on clean samples, however, is not affected. Therefore, intellectual properties can be protected by utilizing backdoor attack mechanisms.

8 Future Directions

Although our methods introduced in this chapter utilize only a small clean validation dataset, it is of value to further decrease the size of the required validation dataset. RAID requires some on-line data to attain samples of possible poisoned inputs in addition to the validation dataset to train a classifier for clean vs. poisoned inputs. During this period, some poisoned inputs may escape detection. Reducing this

transient vulnerability is therefore an avenue for future improvements. Additionally, the reduction of computational complexity is also an important topic for future work. Another extension would be to generalize the methods to backdoor detection for machine learning methods that are not based on neural networks. Further application of the methods to other adaptive triggers should also be considered. Lastly, using explainability tools for DNNs may be helpful to further improve the applicability and usability of backdoor detection methods.

References

1. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Proceedings of the Advances in Neural Information Processing Systems, pp. 1097–1105. Lake Tahoe, Nevada (2012)
2. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition (2014). arXiv preprint arXiv:1409.1556
3. Szegedy, C., Ioffe, S., Vanhoucke, V., Alemi, A.A.: Inception-v4, Inception-ResNet and the impact of residual connections on learning. In: Proceedings of the 31st AAAI Conference on Artificial Intelligence, San Francisco, pp. 4278–4284 (2017)
4. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: Proceedings of the 13th European Conference on Computer Vision, Zurich, Switzerland, pp. 818–833 (2014)
5. Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A.R., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T.N., Kingsbury, B.: Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. IEEE Signal Process. Mag. **29**(6), 82–97 (2012)
6. Sainath, T.N., Mohamed, A.-R., Kingsbury, B., Ramabhadran, B.: Deep convolutional neural networks for LVCSR. In: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing. Vancouver, pp. 8614–8618 (2013)
7. Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., Khudanpur, S.: Recurrent neural network based language model. In: Proceedings of the 11th Annual Conference of the International Speech Communication Association. Chiba, Japan, pp. 1045–1048 (2010)
8. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Proceedings of the Advances in Neural Information Processing Systems, Lake Tahoe, pp. 3111–3119 (2013)
9. Fu, H., Krishnamurthy, P., Khorrami, F.: Functional replicas of proprietary three-axis attitude sensors via LSTM neural networks. In: Proceedings of the IEEE Conference on Control Technology and Applications. Montreal, pp. 70–75 (2020)
10. Chen, C., Seff, A., Kornhauser, A., Xiao, J.: DeepDriving: learning affordance for direct perception in autonomous driving. In: Proceedings of the IEEE International Conference on Computer Vision. Santiago, pp. 2722–2730 (2015)
11. Schwarting, W., Alonso-Mora, J., Rus, D.: Planning and decision-making for autonomous vehicles. Ann. Rev. Control Robot. Auton. Syst. **1**, 187–210 (2018)
12. Hadsell, R., Sermanet, P., Ben, J., Erkan, A., Scoffier, M., Kavukcuoglu, K., Muller, U., LeCun, Y.: Learning long-range vision for autonomous off-road driving. J. Field Robot. **26**(2), 120–144 (2009)
13. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: Proceedings of the International Conference on Learning Representations. San Diego, pp. 1–14 (2015)

14. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks (2013). arXiv preprint arXiv:1312.6199
15. Liu, Y., Ma, S., Aafer, Y., Lee, W.C., Zhai, J., Wang, W., Zhang, X.: Trojaning attack on neural networks. In: Proceedings of the 25th Annual Network and Distributed System Security Symposium. San Diego, pp. 18–221 (2018)
16. Gu, T., Dolan-Gavitt, B., Garg, S.: BadNets: identifying vulnerabilities in the machine learning model supply chain (2017). arXiv preprint arXiv:1708.06733
17. Liu, K., Dolan-Gavitt, B., Garg, S.: Fine-pruning: defending against backdooring attacks on deep neural networks. In: Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses. Heraklion, pp. 273–294 (2018)
18. Liu, K., Tan, B., Karri, R., Garg, S.: Poisoning the (data) well in ML-based CAD: a case study of hiding lithographic hotspots. In: Proceedings of the Design, Automation & Test in Europe Conference & Exhibition. Grenoble, pp. 306–309 (2020)
19. Wenger, E., Passananti, J., Bhagoji, A.N., Yao, Y., Zheng, H., Zhao, B.Y.: Backdoor attacks against deep learning systems in the physical world. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Virtual, pp. 6206–6215 (2021)
20. Li, Y., Li, Y., Wu, B., Li, L., He, R., Lyu, S: Invisible backdoor attack with sample-specific triggers. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, Virtual, pp. 16463–16472 (2021)
21. Saha, A., Subramanya, A., Pirsiavash, H.: Hidden trigger backdoor attacks. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34. New York, pp. 11957–11965 (2020)
22. Li, S., Xue, M., Zhao, B., Zhu, H., Zhang, X.: Invisible backdoor attacks on deep neural networks via steganography and regularization. IEEE Trans. Depend. Secure Comput. **18**(5), 2088–2105 (2020)
23. Liu, Y., Ma, X., Bailey, J., Lu, F.: Reflection backdoor: a natural backdoor attack on deep neural networks. In: Proceedings of the European Conference on Computer Vision, Virtual, pp. 182–199 (2020)
24. Xie, C., Huang, K., Chen, P.-Y., Li, B.: DBA: distributed backdoor attacks against federated learning. In: Proceedings of the International Conference on Learning Representations. New Orleans (2019)
25. Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D., Shmatikov, V.: How to backdoor federated learning. In: Proceedings of the International Conference on Artificial Intelligence and Statistics, Virtual, pp. 2938–2948 (2020)
26. Andreina, S., Marson, G.A., Möllering, H., Karame, G.: BaFFLe: backdoor detection via feedback-based federated learning. In: Proceedings of the IEEE International Conference on Distributed Computing Systems, Virtual, pp. 852–863 (2021)
27. Yao, Y., Li, H., Zheng, H., Zhao, B.Y.: Latent backdoor attacks on deep neural networks. In: Proceedings of the ACM SIGSAC Conference on Computer and Communication Security. London, pp. 2041–2055 (2019)
28. Zhang, Z., Jia, J., Wang, B., Gong, N.Z.: Backdoor attacks to graph neural networks. In: Proceedings of the ACM Symposium on Access Control Models and Technology, Virtual, pp. 15–26 (2021)
29. Dai, J., Chen, C., Li, Y.: A backdoor attack against LSTM-based text classification systems. IEEE Access **7**, 138872–138878 (2019)
30. Gong, X., Chen, Y., Wang, Q., Huang, H., Meng, L., Shen, C., Zhang, Q.: Defense-resistant backdoor attacks against deep neural networks in outsourced cloud environment. IEEE J. Sel. Areas Commun. **39**(8), 2617–2631 (2021)
31. Wang, B., Yao, Y., Shan, S., Li, H., Viswanath, B., Zheng, H., Zhao, B.Y.: Neural Cleanse: identifying and mitigating backdoor attacks in neural networks. In: Proceedings of the 40th IEEE Symposium on Security and Privacy. San Francisco, pp. 707–723 (2019)
32. Guo, W., Wang, L., Xing, X., Du, M., Song, D.: TABOR: a highly accurate approach to inspecting and restoring trojan backdoors in AI systems (2019). arXiv preprint arXiv:1908.01763

33. Chen, H., Fu, C., Zhao, J., Koushanfar, F.: DeepInspect: a black-box trojan detection and mitigation framework for deep neural networks. In: Proceedings of the 28th International Joint Conference on Artificial Intelligence. International Joint Conferences on Artificial Intelligence Organization, Macao, pp. 4658–4664 (2019)
34. Xu, X., Wang, Q., Li, H., Borisov, N., Gunter, C.A., Li, B.: Detecting AI trojans using meta neural analysis (2019). arXiv preprint arXiv:1910.03137
35. Li, Y., Ma, H., Zhang, Z., Gao, Y., Abuadbba, A., Fu, A., Zheng, Y., Al-Sarawi, S.F. Abbott, D.: NTD: non-transferability enabled backdoor detection (2021). arXiv preprint arXiv:2111.11157
36. Liu, Y., Lee, W.-C., Tao, G., Ma, S., Aafer, Y., Zhang, X.: ABS: scanning neural networks for back-doors by artificial brain stimulation. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security. London, pp. 1265–1282 (2019)
37. Lee, K., Lee, K., Lee, H., Shin, J.: A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In: Proceedings of the Conference on Neural Information Processes Systems. Montreal, pp. 7167–7177 (2018)
38. Chou, E., Tramèr, F., Pellegrino, G., Boneh, D.: SentiNet: detecting physical attacks against deep learning systems (2018). arXiv preprint arXiv:1812.00292
39. Gao, Y., Xu, C., Wang, D., Chen, S., Ranasinghe, D.C., Nepal, S.: STRIP: a defence against trojan attacks on deep neural networks. In: Proceedings of the 35th Annual Computer Security Applications Conference. San Juan, pp. 113–125 (2019)
40. Kwon, H.: Detecting backdoor attacks via class difference in deep neural networks. IEEE Access **8**, 191049–191056 (2020)
41. Fu, H., Veldanda, A.K., Krishnamurthy, P., Garg, S., Khorrami, F.: A feature-based on-line detector to remove adversarial-backdoors by iterative demarcation. IEEE Access **10**, 5545–5558 (2022)
42. Chen, B., Carvalho, W., Baracaldo, N., Ludwig, H., Edwards, B., Lee, T., Molloy, I., Srivastava, B.: Detecting backdoor attacks on deep neural networks by activation clustering (2018). arXiv preprint arXiv:1811.03728
43. Tran, B., Li, J., Madry, A.: Spectral signatures in backdoor attacks. In: Proceedings of Advances in Neural Information Processing Systems, vol. 31, Montreal, pp. 8000–8010 (2018)
44. Tang, D., Wang, X., Tang, H., Zhang, K.: Demon in the variant: statistical analysis of DNNs for robust backdoor contamination detection. In: Proceedings of the 30th USENIX Security Symposium, Virtual, pp. 1541–1558 (2021)
45. Wang, Z., Simoncelli, E.P., Bovik, A.C.: Multiscale structural similarity for image quality assessment. In: Proceedings of the 37th Asilomar Conference on Signals, Systems & Computers, vol. 2, Pacific Grove, pp. 1398–1402 (2003)
46. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
47. Lin, M., Chen, Q., Yan, S.: Network in network. In: Proceedings of the International Conference on Learning Representations, Banff, pp. 1–10 (2014)
48. Stallkamp, J., Schlipsing, M., Salmen, J., Igel, C.: The German Traffic Sign Recognition Benchmark: a multi-class classification competition. In: Proceedings of the International Joint Conference on Neural Networks. San Jose, pp. 1453–1460 (2011)
49. Veldanda, A.K., Liu, K., Tan, B., Krishnamurthy, P., Khorrami, F., Karri, R., Dolan-Gavitt, B., Garg, S.: NNoculation: broad spectrum and targeted treatment of backdoored DNNs (2020). arXiv preprint arXiv:2002.08313
50. Wolf, L., Hassner, T., Maoz, I.: Face recognition in unconstrained videos with matched background similarity. In: Proceedings of the IEEE Computer Vision and Pattern Recognition. Colorado Springs, pp. 529–534 (2011)
51. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Las Vegas, pp. 770–778 (2016)
52. Miller, J.: Reaction time analysis with outlier exclusion: bias varies with sample size. Q. J. Exp. Psychol. **43**(4), 907–912 (1991)

53. Tipping, M.E., Bishop, C.M.: Probabilistic principal component analysis. J. R. Stat. Soc. Ser. B (Statistical Methodology) **61**(3), 611–622 (1999)
54. Fu, H., Veldanda, A.K., Krishnamurthy, P., Garg, S., Khorrami, F.: Detecting backdoors in neural networks using novel feature-based anomaly detection (2020). arXiv preprint arXiv:2011.02526
55. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
56. Chen, Y., Zhou, X.S., Huang, T.S.: One-class SVM for learning in image retrieval. In: Proceedings of International Conference on Image Processing, vol. 1, Thessaloniki, pp. 34–37 (2001)
57. Dong, Y., Hopkins, S., Li, J.: Quantum entropy scoring for fast robust mean estimation and improved outlier detection. In: Proceedings of the Advances in Neural Information Processing Systems. Vancouver, pp. 6067–6077 (2019)
58. Hariri, S., Kind, M.C., Brunner, R.J.: Extended isolation forest. IEEE Trans. Knowl. Data Eng. **33**(4), 1479–1489 (2019)
59. Lesouple, J., Baudoin, C., Spigai, M., Tourneret, J.-Y.: Generalized isolation forest for anomaly detection. Pattern Recognit. Lett. **149**, 109–119 (2021)
60. LeCun, Y., Cortes, C.: MNIST handwritten digit database (2010). http://yann.lecun.com/exdb/mnist
61. Deng, J., Dong, W., Socher, R., Li, L., Li, K., Fei-Fei, L.: ImageNet: a large-scale hierarchical image database. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Miami, pp. 248–255 (2009)
62. Sun, Y., Wang, X., Tang, X.: Deep learning face representation from predicting 10,000 classes. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Columbus, pp. 1891–1898 (2014)
63. Gu, T., Liu, K., Dolan-Gavitt, B., Garg, S.: BadNets: evaluating backdooring attacks on deep neural networks. IEEE Access **7**, 47230–47244 (2019)
64. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Honolulu, pp. 4700–4708 (2017)
65. Adi, Y., Baum, C., Cisse, M., Pinkas, B., Keshet, J.: Turning your weakness into a strength: watermarking deep neural networks by backdooring. In: Proceedings of the 27th USENIX Security Symposium. Baltimore, pp. 1615–1631 (2018)

Robustness for Embedded Machine Learning Using In-Memory Computing

Priyadarshini Panda, Abhiroop Bhattacharjee, and Abhishek Moitra

1 Introduction

Deep learning has achieved state-of-the-art prediction capabilities across a variety of cognitive and analytics tasks. This has led to the ubiquitous deployment of Deep Neural Networks (DNNs) in low power edge devices [1, 2]. For edge computing, analog crossbar architectures have emerged as a front runner towards low-latency and energy-efficient acceleration platforms in resource-constrained scenarios. Here, the synaptic weights of the DNNs are mapped on arrays (crossbars) of Non-Volatile Memory (NVM) devices, such as Resistive RAM (ReRAM), Phase Change Memory (PCM), Ferroelectric Field Effect Transistors (FeFET) and so forth [3, 4]. They efficiently perform analog dot-product operations, emulating Multiply and Accumulate (MAC) operations in DNNs, when input voltages are applied to the rows of the crossbar.

Despite achieving super-human performance in many computer vision tasks [5], DNNs have been shown to be vulnerable to adversarial attacks (see Fig. 1). Here, small and strategically crafted noise in the input can fool the DNN leading to failure [6–10]. This vulnerability severely limits the deployment and potential safe-use of DNNs at the edge for real-world applications. To defend against adversarial attacks, previous works have used two broad approaches: (1) Adversarial classification [11–15] and (2) Adversarial detection [16–18]. Under adversarial classification, there have been prior works that have used techniques such as adversarial training, input feature transformation among others to classify the adversarial samples accurately [11–15]. In contrast, adversarial detection works focus on identifying clean and adversarial samples such that the detected adversarial samples are not passed to

P. Panda (✉) · A. Bhattacharjee · A. Moitra
Department of Electrical Engineering, Yale University, New Haven, CT, USA
e-mail: priya.panda@yale.edu

S. Pasricha, M. Shafique (eds.), *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing*, https://doi.org/10.1007/978-3-031-40677-5_17

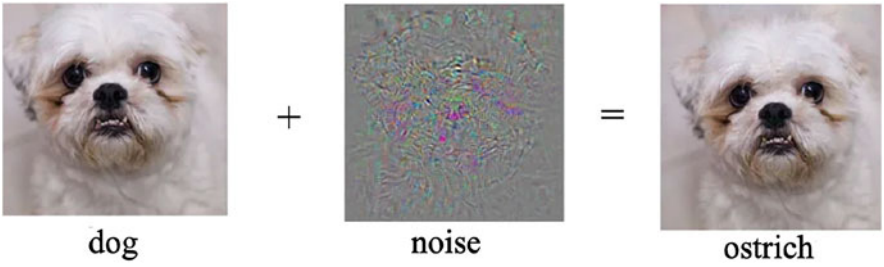


Fig. 1 Adversarial attacks can fool a DNN by adding structured perturbations to clean inputs

the output of the DNN for classification [16–18]. However, these techniques are software-centric and not hardware-friendly, requiring high computational overheads. To this end, recent works such as [10, 19] show that quantization methods, which primarily reduce compute resource requirements of DNNs, act as a straightforward way of improving the adversarial robustness of DNNs. Other works such as [20, 21] use model compression and pruning techniques to optimize and reduce computational complexities of DNNs while guaranteeing adversarial robustness. In [19, 22], efficiency-driven hardware optimization techniques are leveraged to improve adversarial resilience of DNNs, while yielding energy-efficiency. However, none of these works has been integrated with a crossbar-based platform (considering intrinsic crossbar noise) for DNN inference. Vanilla implementation of DNNs on crossbars, including those trained with software defenses such as adversarial training, suffers from significant loss in robustness caused by hardware noise [23–25]. There has been limited study in understanding the robustness of crossbar implemented DNNs. Thus, we highlight hardware and energy-efficiency driven works that improve the robustness of DNNs deployed on analog crossbars in two broad aspects: (1) Improving adversarial robustness and (2) Mitigating the detrimental effects of crossbar non-idealities on DNNs, thereby ameliorating the performance (accuracy) of DNNs on crossbars. Note, these works do not pose a huge overhead of hardware-aware retraining of a pretrained DNN model before deployment on crossbars.

We begin by discussing two recent works that use analog crossbars and improve the adversarial robustness of the mapped DNNs. Note, adversarial robustness implies improving the performance of the hardware-mapped DNN model against adversarial samples without compromising the classification accuracy of clean images on hardware. In the first work, we introduce a technique called NEAT [26] that mitigates the impact of selector-induced non-linearities and resistive crossbar non-idealities for robust implementation of DNNs on 1T-1R crossbars. Second, we showcase another work, called DetectX [27], that uses hardware signatures present in analog crossbar architectures to perform energy-efficient adversarial detection. While NEAT is tailored for adversarial classification, DetectX is an adversarial detection method.

Finally, we delve into a specific case of the inference of DNNs having structured sparsity in their weights on analog crossbar arrays. Recently, crossbar-aware structured pruning algorithms [28–31] have received significant attention in developing increasingly sparse DNN models requiring fewer crossbars to be mapped, thereby introducing huge savings in terms of crossbar energy and area-efficiencies [32]. However, a holistic evaluation of the performance of such algorithms by considering the impact of resistive crossbar non-idealities was missing. In this chapter, we highlight a recent work [33] which shows that increased structured sparsity in DNNs negatively interferes with crossbar non-idealities that can degrade their classification accuracy (or robustness) during inference. This work also introduces two hardware-centric non-ideality mitigation strategies, namely crossbar-column rearrangement and Weight-Constrained-Training (WCT), to help improve the performance or robustness of the sparse DNNs on crossbars.

This chapter is organized as follows. Section 2 explains the background on adversarial attacks, memristive crossbars and non-idealities. In Sect. 3, we discuss the NEAT technique that introduces a non-ideality control technique which causes a rise in adversarial robustness. Section 4 introduces the DetectX technique that performs energy signature separation for adversarial detection. Section 5 explains the impact of structured sparsity in DNNs and their interaction with non-ideal crossbars. Section 6 gives an overview of related works and scopes out different crossbar-based studies with different objectives. Finally, we conclude in Sect. 7.

2 Background

2.1 Adversarial Attacks

Adversarial samples are created by generating a crafted noise and adding it to the clean data samples. In this chapter, we discuss two widely used methods to generate the noise for creating adversarial attacks.

1. *Fast Gradient Sign Method (FGSM)* [6] is a one-step gradient-based attack shown in Eq. (1). To generate the noise, first, the gradients of the DNN loss $\mathcal{L}(\theta, x, y_{true})$ with respect to the input x are calculated. Here, θ represents the parameters of the DNN and y_{true} represents the labels of the input data. Then, a $sign()$ operation converts the gradients into unit directional vectors. The unit vector is multiplied by a scalar perturbation value, ϵ, that determines the strength of the attack. Finally, the perturbation vector is added to the input x to create an adversarial data. Note that perturbations are added to x along the direction of the gradients to maximize DNN loss $\mathcal{L}$.

$$x_{adv} = x + \epsilon \, sign(\nabla_x(\mathcal{L}(\theta, x, y_{true}))) \tag{1}$$

2. *Projected Gradient Descent (PGD)*: The PGD attack, shown in Eq. (2) is an iterative attack over n steps. It is basically a multi-step variant of the FGSM attack. In each step i, perturbations of strength α are added to x_{adv}^{i-1}. Note that x_{adv}^{0} is created by adding random noise to the clean input x. Additionally, for each step, x_{adv}^{i} is projected on a *Norm ball* [8], of radius ϵ. In this chapter, the L_∞ *Norm ball* (of radius ϵ) projection is used for all the PGD attacks. In other words, we ensure that the maximum pixel difference between the clean and adversarial inputs is ϵ.

$$x_{adv} = \sum_{i=1}^{n} x_{adv}^{i-1} + \alpha \; sign(\nabla_x \mathcal{L}(\theta, x, y_{true})) \tag{2}$$

2.2 Memristive Crossbars and Their Non-idealities and Non-linearities

Memristive crossbar arrays have been used to implement MAC operations in an analog manner. Crossbars consist of 2D arrays of NVM devices, *Digital-to-Analog Converters* (DAC), *Analog-to-Digital Converters* (ADC) and a write circuit. The synaptic devices at the cross-points are programmed to a particular value of conductance (between G_{MIN} and G_{MAX}) during inference. The MAC operations are performed by converting the digital inputs to the DNN into analog voltages on the *Read Wordlines* (RWLs) using DACs, and sensing the output current flowing through the bit-lines (BLs) using the ADCs [23, 34–38]. In other words, the activations of the DNNs are mapped as analog voltages V_i input to each row and weights are programmed as synaptic device conductances (G_{ij}) at the cross-points as shown in Fig. 2a. For an ideal crossbar array, during inference, the voltages interact with the device conductances and produce a current (governed by Ohm's Law). Consequently, by Kirchhoff's current law, the net output current sensed at each column j is the sum of currents through each device, i.e. $I_{j(ideal)} = \Sigma_i G_{ij} * V_i$. We term the matrix G_{ideal} as the collection of all G_{ij}'s for a crossbar instance. However, in reality, the analog nature of the computation leads to various hardware noise or non-idealities, such as, circuit-level resistive non-idealities and device-level variations [23, 34, 36, 37, 39–43].

Non-idealities: Fig. 2b describes the equivalent circuit for a crossbar accounting for various circuit-level and device-level non-idealities, *viz.* $Rdriver$, $Rwire\_row$, $Rwire\_col$ and $Rsense$ (interconnect parasitics), modelled as parasitic resistances and variations in the synapses owing to the stochasticity of the memristive devices. This results in a $G_{non\text{-}ideal}$ matrix, with each element G'_{ij} incorporating the effect due to the non-idealities, obtained using circuit laws (Kirchhoff's laws and Ohm's law) and linear algebraic operations [23, 24, 33, 39, 44]. Consequently, the net output current sensed at each column j becomes $I_{j(non\text{-}ideal)} = \Sigma_i G'_{ij} * V_i$, which deviates

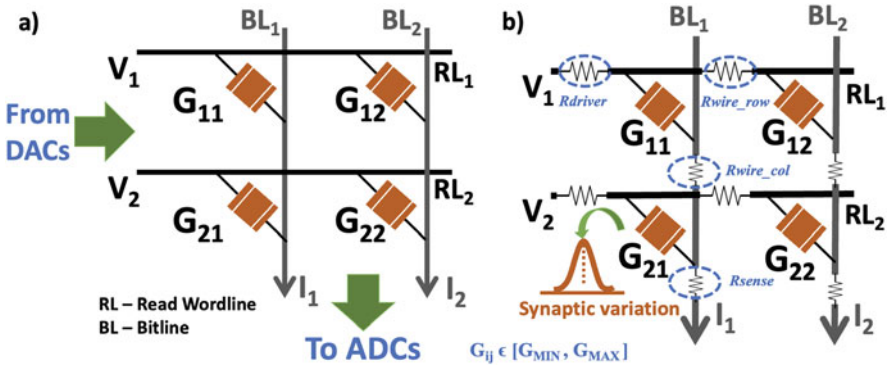


Fig. 2 (**a**) A 2×2 crossbar array with input voltages V_i, synaptic conductances G_{ij} and output currents $I_j = \sum_i G_{ij} * V_i$. (**b**) A 2×2 crossbar array with the resistive and the synaptic device-level non-idealities. These non-idealities lead to imprecise dot-product currents and that manifests as accuracy degradation when DNNs are evaluated on crossbars

from its ideal value. This manifests as accuracy degradation for DNNs mapped onto crossbars. The relative deviation of $I_{non\text{-}ideal}$ from its ideal value is measured using *non-ideality factor* (NF) [34] as:

$$NF = (I_{ideal} - I_{non\text{-}ideal})/I_{ideal}. \tag{3}$$

Thus, NF is a direct measure of crossbar non-idealities, i.e. increased non-idealities induce a greater value of NF, affecting the accuracy and hence, the robustness of the DNNs mapped onto them. All the analyses in Sects. 3 and 5 involving non-idealities are performed on memristive crossbars with an ON/OFF ratio of 10 (i.e. $R_{MIN} = 30\,k\Omega$ and $R_{MAX} = 300\,k\Omega$), having the resistive non-idealities as follows: $Rdriver = 1\,k\Omega$, $Rwire\_row = 5\,\Omega$, $Rwire\_col = 10\,\Omega$ and $Rsense = 1\,k\Omega$. The non-ideality in the memristive devices in the form of device-level process variation has been modelled as a Gaussian variation in the conductances (G_M) of the NVM devices with $\sigma/\mu = 10\%$ [45].

Recently, 1T-1R NVM crossbars have received significant attention since the pass-transistor in series with the NVM device at the synapses can help mitigate sneak paths (prevalent in 1R crossbar arrays) and the incorrect programming of the NVM device induced by noise [46, 47]. Figure 3 illustrates an $M \times N$ crossbar having 1T-1R synapses at the cross-points wherein, the access transistors are driven by a gate-voltage (V_g) fed through the select-lines (SLs). A low V_g operation is favorable for implementing a DNN on crossbars in a resource-constrained scenario as it has been shown in prior works [26] that the total power dissipated by a 1T-1R crossbar array diminishes with reduction of the transistor gate-voltage (V_g). However, it is imperative to understand the other repercussions of low V_g operation in 1T-1R crossbars that impact the performance and hence, robustness of the mapped DNN models.

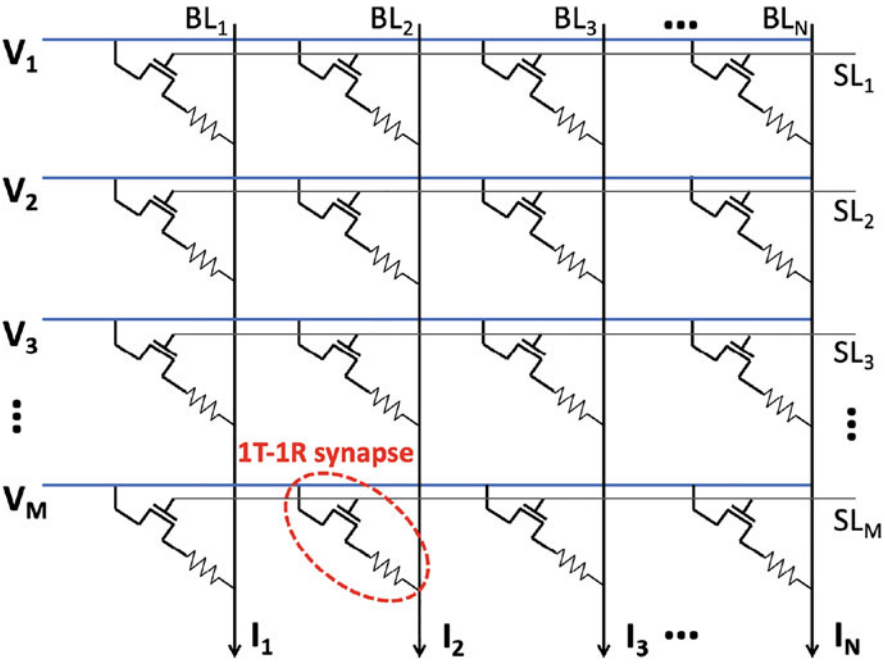


Fig. 3 Illustration of an M × N 1T-1R Crossbar. A Transistor (T) in series with an NVM device (R) is present at every synapse. Select-lines (SLs) are used to turn on transistors for selected rows, while the dot-product currents are sensed through the bit-lines (BLs)

Non-linearities: In addition to the above-mentioned non-idealities, 1T-1R crossbars are susceptible to various non-linearities that affect the effective conductance of each synapse (especially, at lower V_g) and hence, the output current across each column in a crossbar array. This would manifest as accuracy degradation for the DNNs mapped onto such crossbars. In [26], to understand the effects of the non-linearities alone on introducing the access transistor (or selector) in the synapse, extensive SPICE simulations were performed using the 1T-1R synaptic configuration with different input voltages, conductances and V_g ranges excluding the circuit-level and device-level non-idealities. For all the analyses involving 1T-1R synapses, the selector devices were based on 45 nm CMOS technology model and the memristive device had $R_{ON} = 30\,k\Omega$ and $R_{OFF} = 300\,k\Omega$.

For a crossbar in the 1R configuration, the weights W of the DNN are directly mapped to a memristor conductance state ($G_M = 1/R_M$) in a linear fashion. On the other hand, in the 1T-1R configuration, W is mapped to the effective conductance $G_{eff} = 1/(R_M + R_t)$, where R_t is the equivalent resistance due to the transistor. The non-linearities in the 1T-1R crossbars arise due to the dependence of R_t on V_{in}. Note, V_{in} is proportional to the neuronal activation values of the DNN which varies with the input. Hence, these are data-dependent non-linearities. It has been shown

in [26] that the effective conductance G_{eff} is a function of NVM conductance G_M, input voltage V_{in}, and gate-voltage V_g, which can be formulated as:

$$G_{eff} = f_1(G_M, V_{in}, V_g). \tag{4}$$

3 Non-linearity Aware Training (NEAT): Mitigating the Impact of Crossbar Non-idealities and Non-linearities for Robust DNN Implementations

In this section, the NEAT technique is introduced that provides a new perspective on the energy-efficient and robust implementation of DNNs on 1T-1R crossbars [26]. It begins with the identification of a range of memristive conductances over a range of input voltages via SPICE simulations such that Eq. (4) can be approximated as:

$$G_{eff} = f_2(G_M, V_g). \tag{5}$$

This eliminates the input-data dependency of the effective 1T-1R synaptic conductance (G_{eff}) for a given value of V_g of transistor operation. In other words, for a given value of V_g, there exists an upper bound cut-off value ($G_{eff\,cutoff}$) for which the 1T-1R synapse exhibits linear characteristics, and $G_M \approx k * G_{eff}$, where k is a scalar. The corresponding NVM device state at $G_{eff\,cutoff}$ is termed as $G_{M\,cutoff}$. Figure 4 shows the $G_{M\,cutoff}$ vs. V_g plot for supply voltage $V_{supply} = 0.25V,\ 0.5V$ and V_{in} in the range $0 \leq V_{in} \leq V_{supply}$. It can be seen that as V_g is lowered (for resource-constrained scenarios), the overall range of memristive conductance states G_M for data-independent and linear synaptic characteristics decreases owing to low values of $G_{M\,cutoff}$.

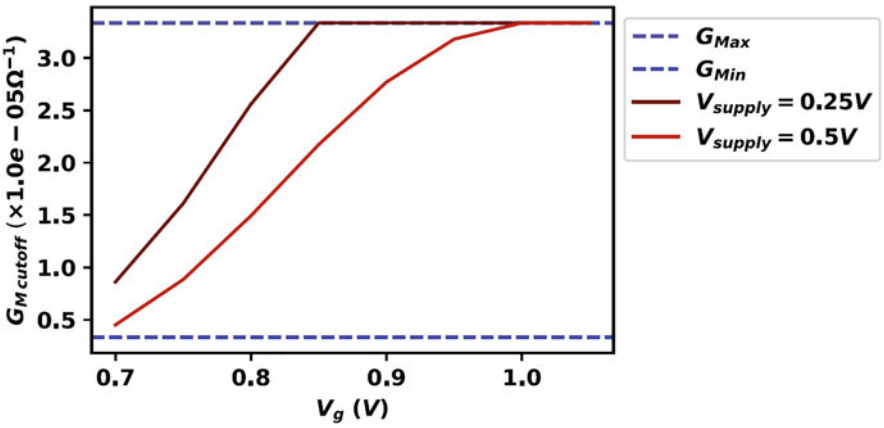


Fig. 4 The variation in $G_{M\,cutoff}$ with respect to selector gate-voltage V_g

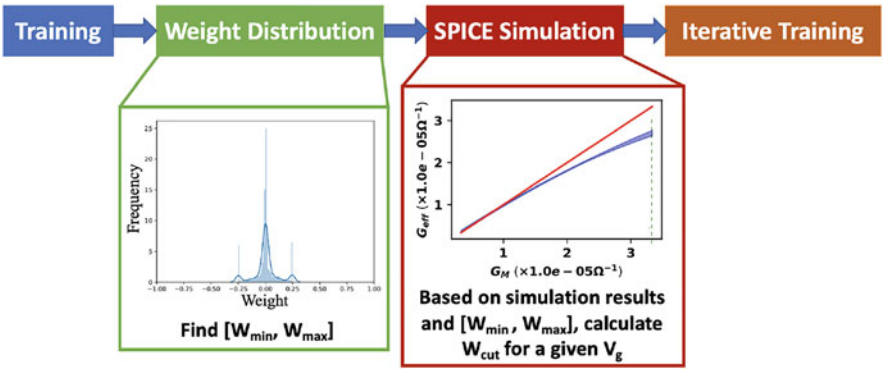


Fig. 5 Overall flow of NEAT

After identifying $G_{eff\,cutoff}$ for a given V_g, the corresponding value for W_{cut} is obtained for the software DNN which is to be mapped onto the 1T-1R crossbars. Then, all the weights (W) of the pretrained DNN are restricted in the interval $[-W_{cut}, W_{cut}]$ as shown in Eq. (6):

$$W_{map} = \begin{cases} W & |W| \leq W_{cut} \\ W_{cut} & W > W_{cut} \\ -W_{cut} & W < -W_{cut}. \end{cases} \tag{6}$$

From Eq. (6), we observe that for the linear regime ($|W| \leq W_{cut}$, which corresponds to $G_{eff} \approx k * G_M$), the software weight parameters can be mapped linearly onto the crossbars. While, for the non-linear regime ($|W| > W_{cut}$ that corresponds to deviation of G_{eff} from G_M), W is clipped at W_{cut}. The objective of NEAT is to restrict the weight parameters to be within the linear regime for the given gate-voltage V_g of the transistor, thereby curbing loss in computational accuracy post-mapping DNNs onto 1T-1R crossbars. Figure 5 illustrates the overall flow of the NEAT process.

Iterative Training: In NEAT, after setting the optimal V_g and W_{cut} values, the weights of the DNN get transformed. If we use lower values of V_g which do not cover all weight ranges, the weight distribution gets altered, resulting in accuracy degradation. To address this issue, iterative training is proposed which consists of two steps. Step 1 is essentially restricting the weights of the DNN (W) in the suitable cut-off regime as per Eq. (6). Step 2 involves retraining the networks iteratively for a couple of epochs to recover any accuracy loss incurred from Step 1. These two steps are repeated so that greater number of weights in the network can be located in the linear regime when mapped onto crossbars.

In Fig. 6, V_g is varied from $0.75\,V$ to 1.0 and the classification accuracy of various DNN architectures using CIFAR10 and CIFAR100 datasets is reported. The results show that low V_g induces low W_{cut} and in turn decreases performance

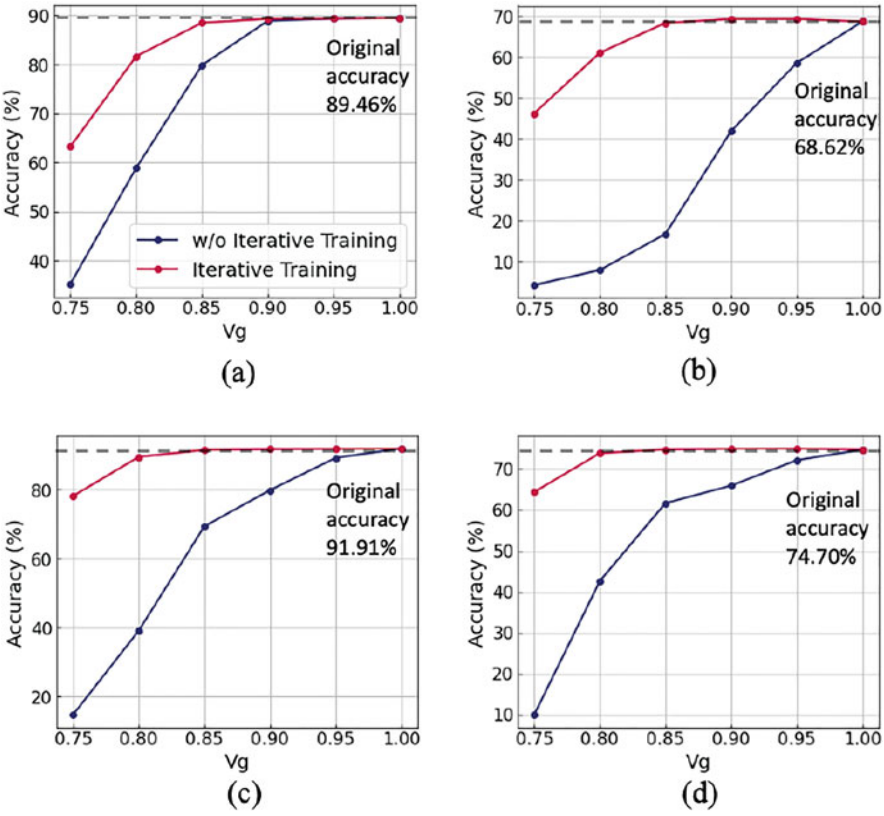


Fig. 6 Classification accuracy of various NEAT-based DNN models with V_g varied from 0.75 V to 1.0 V. (**a**) VGG11/CIFAR10. (**b**) VGG11/CIFAR100. (**c**) ResNet18/CIFAR10. (**d**) ResNet18/CIFAR100

when DNN weights are restricted to W_{cut} regime. However, using iterative training recovers the performance degradation. Especially, for a ResNet18 architecture, using iterative training shows improvement over 50% in terms of accuracy at $V_g = 0.75\,V$. Moreover, with iterative training, VGG11 and ResNet18 networks almost maintain their classification accuracy in the range of $V_g = [0.85, 1.0]$ and $V_g = [0.8, 1.0]$, respectively. In this manner, NEAT helps in the hardware-aware robust mapping of DNN architectures on 1T-1R crossbar with minimal training overheads.

In addition to maintaining DNN performance in presence of 1T-1R non-linearities, NEAT ensures energy-efficient inference with DNNs, specifically in the low V_g scenario. When NEAT technique is applied at low V_g values, we have lower absolute values of $G_{eff\,cutoff}$ and hence, W_{cut}. This implies that for crossbars operating at lower V_g values, we would find greater proportion of low conductance synapses post mapping of DNN weights onto the 1T-1R crossbars. This helps

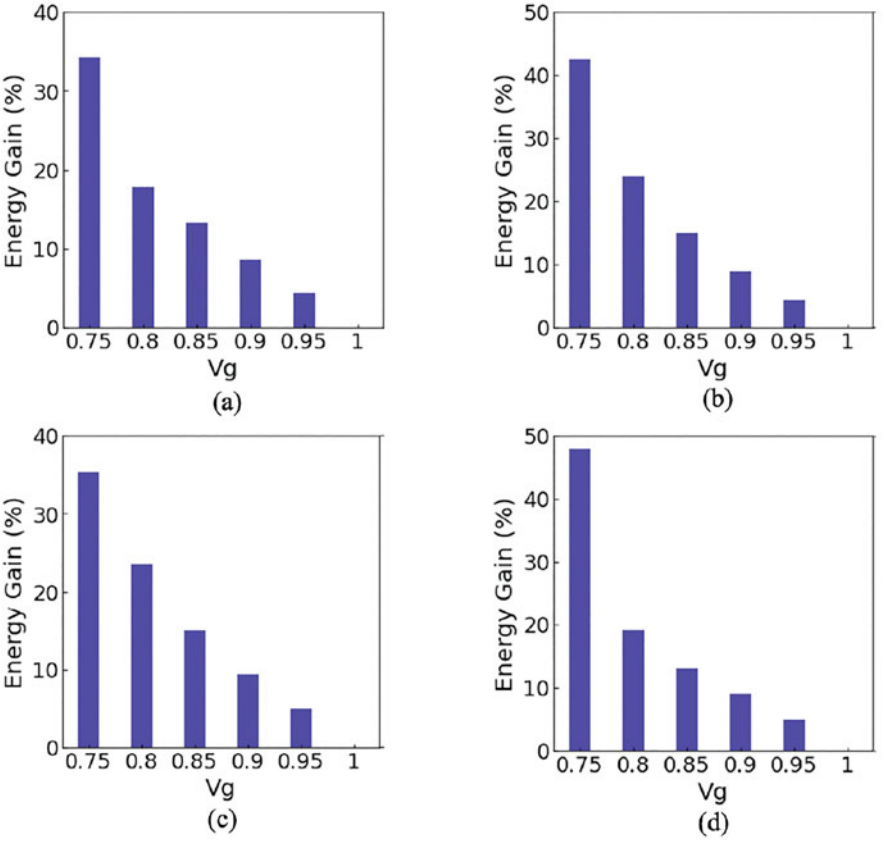


Fig. 7 Normalized energy gain for various NEAT-based DNN models with V_g varied from 0.75 V to 1.0 V. (**a**) VGG11/CIFAR10. (**b**) VGG11/CIFAR100. (**c**) ResNet18/CIFAR10. (**d**) ResNet18/CIFAR100

minimise the power dissipated in the crossbar arrays by reducing crossbar-column currents. In Fig. 7, we present the energy-efficiency of various DNN configurations with NEAT. The energy computed for $V_g = 1.0\,V$ is taken as *baseline* against which energy gains (%) for other values of V_g are shown. NEAT achieves high energy gain by simply reducing V_g. Especially, we can achieve ∼23% energy gain at $V_g = 0.8\,V$ on ResNet18 architecture with CIFAR10 while suffering minimal accuracy loss (∼1.5% in Fig. 6). However, selecting a very low value for V_g such as $V_g = 0.75\,V$ induces huge performance degradation.

Having mapped DNNs via NEAT in an energy-efficient manner onto 1T-1R crossbars and mitigating the impact of synaptic non-linearities, we now study the impact of crossbar non-idealities (enlisted in Sect. 2.2) on the robustness of NEAT-DNNs. Note, we would use the term "NEAT-DNN" to denote a DNN trained using NEAT and mapped onto 1T-1R crossbars, while the term 'Normal-DNN' would

refer to standard DNNs mapped directly onto 1R crossbars with non-idealities (the *baseline*). Henceforth, all experiments in Sect. 3 involving crossbar arrays would include the crossbar non-idealities.

It has been shown that the value of NF in crossbars decreases with increase in the effective resistance of a crossbar array, which minimizes the effect of interconnect resistive non-idealities [26, 33, 34, 44]. By increasing the proportion of lower conductance synapses in a crossbar array, one can reduce the impact of crossbar non-idealities and hence, the non-ideality factor. When NEAT technique is applied at low V_g values, we have a greater proportion of low conductance synapses post mapping of DNN weights onto the 1T-1R crossbars. Hence, NF is expected to be lower in the case of low V_g operation of crossbars. Furthermore, the value of NF for DNNs mapped onto 1T-1R crossbars via NEAT would be lesser than the value of NF for standard DNNs mapped onto 1R crossbars (the *baseline*). Since, NEAT boosts the feasibility of low conductance synapses and reduces the impact of crossbar non-idealities, NEAT-DNNs are more robust both in terms of clean and adversarial accuracies than the *baseline* Normal-DNNs.

Modes of adversarial attack: For unleashing adversarial attacks (FGSM/PGD) on the crossbar-mapped models of the DNNs, consider two modes:

1. **Software-inputs-on-hardware (SH) mode** where, the adversarial perturbations for each attack are created using the loss function of the software DNN model normally trained without applying the NEAT technique, and then added to the clean input that yields the adversarial input. The generated adversaries are then fed to the crossbar-mapped DNN. This a case of Black-Box adversarial attack on hardware.
2. **Hardware-inputs-on-hardware (HH) mode** where, the adversarial inputs are generated for each attack using the loss from the crossbar-based hardware models. It is evident that HH perturbations will incorporate the effect of intrinsic hardware non-idealities and thus will cast stronger attacks than SH. This is a case of White-Box adversarial attack on hardware.

Results for robustness in terms of clean and adversarial accuracies: Here, we evaluate robustness of the DNNs on non-ideal crossbars graphically as shown in Fig. 8 by plotting *'robustness maps'* as has been proposed in [26, 44]. Note, the NEAT-DNNs are shown for $V_g = 0.8\,V$. This approach to assess robustness of a network has been shown to be comprehensive and accurate since, it takes into account the cumulative impact of both clean accuracy and adversarial accuracy (which is a strong function of the clean accuracy). For a specific mode of attack (SH or HH) and a given crossbar size, we plot $\Delta\ Clean\ Accuracy$, the difference between clean accuracy of the crossbar-mapped DNN in question and the corresponding clean accuracy of its software model, on the x-axis. $\Delta\ Adversarial\ Accuracy$ (for a particular ϵ value) which is the difference between the adversarial accuracy of the mapped network in question and the corresponding adversarial accuracy of the software model is plotted on the y-axis. The value of $\Delta\ Clean\ Accuracy$ is always negative since DNNs when mapped on hardware suffer accuracy loss owing to non-idealities. The region bounded by the line $y = -x$

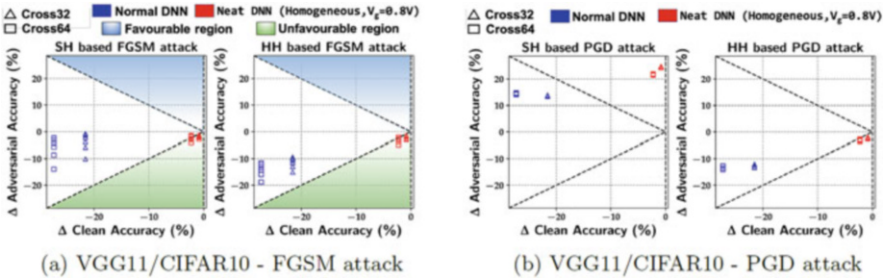


Fig. 8 (**a**), (**b**) Robustness maps for VGG11 DNN using CIFAR10 dataset for SH and HH modes of FGSM and PGD attacks, respectively

and the y-axis denotes the *favorable region* and the closer a point is towards this region, the better is the robustness of the network in question. Likewise, the region bounded by the line $y = x$ and the y-axis is the *unfavorable region*, where the mapped network is highly vulnerable to adversarial attacks. The favorable and unfavorable regions have been demarcated Fig. 8a.

Figure 8 shows the robustness maps for DNNs based on VGG11 network with CIFAR10 dataset for both SH and HH modes of attack. Figure 8a pertains to FGSM attack with ϵ varying from 0.05 to 0.3 with step size of 0.05. We find that NEAT-DNNs have significantly greater clean accuracy ($\sim$13% and $\sim$17% higher for 32×32 and 64×64 crossbars, respectively) as well as better adversarial accuracies on hardware for both modes of attack. The points corresponding to NEAT-DNNs are situated closer to the favorable region than the corresponding points for Normal-DNNs. This is a consequence of the reduction in non-ideality factor in case of iterative training with NEAT algorithm. Note, the points for 64×64 crossbars are situated farther from the favorable region than the corresponding points for 32×32 crossbars. This gap is owing to greater non-idealities that exist in case of a larger 64×64 crossbar than a 32×32 crossbar. However, this gap significantly decreases for NEAT-DNNs indicating that NEAT greatly reduces the impact of the crossbar non-idealities on the inference accuracy of the mapped DNNs. In other words, NEAT-DNNs do not suffer significant accuracy losses on larger crossbars. We further observe that the points for a NEAT-DNN, given a crossbar size, are more closely packed than the corresponding points for Normal-DNN. This implies that even on increasing the attack strength (ϵ), lesser adversarial loss is observed for DNN models on crossbars trained with NEAT algorithm.

Figure 8b also presents similar results but for a strong PGD attack with ϵ varying from 2/255 to 32/255 with step size of 2/255. In this case, the robustness is very high for SH mode of attack as compared to HH mode of attack, with points corresponding to NEAT-DNNs situated inside the favorable region for the SH mode. Similar to the case of FGSM attack, NEAT-DNNs outperform Normal-DNNs in terms of robustness for both modes of attack. Here, points for different ϵ values, given a style of mapping and crossbar size, are more closely packed than the corresponding

points of FGSM attack. This implies that hardware non-idealities interfere more with PGD attacks than FGSM attacks resulting in lesser accuracy loss.

From the above discussion, we find that NEAT-based DNNs are more immune to the impact of non-idealities and lead to robust implementations on non-ideal 1T-1R crossbars in addition to higher crossbar energy-efficiencies.

4 DetectX: Improving the Robustness of DNNs Using Hardware Signatures in Memristive Crossbar Arrays

In this section, we discuss how hardware signatures in memristive crossbar architectures can be used to detect adversarial attacks in an energy-efficient manner [27].

For detecting adversaries, we use a function called the Sum of Currents (SoI). It is defined as the absolute value summation of all the feature outputs of a particular layer as shown in Eq. (7).

$$SoI_l = \sum_{j=1}^{m} |Z_j|_l \tag{7}$$

Here, Z_j is the result of the weighted summation outputs of a particular layer l. This is proportional to the summation of the column current magnitudes in a memristive crossbar array. Interestingly, as shown in Fig. 9a, we find that the clean and adversarial SoI distributions of the first layer have an inherent separation between them. However, due to a significant overlap between the two distributions, the adversarial detection is low. To this end, we use a dual-phase training methodology to increase the distance between the SoI distributions and improve the adversarial detection.

In the first phase of training, we train the first layer of the DNN to increase the separation between the clean and adversarial SoIs. For this, we use a loss function

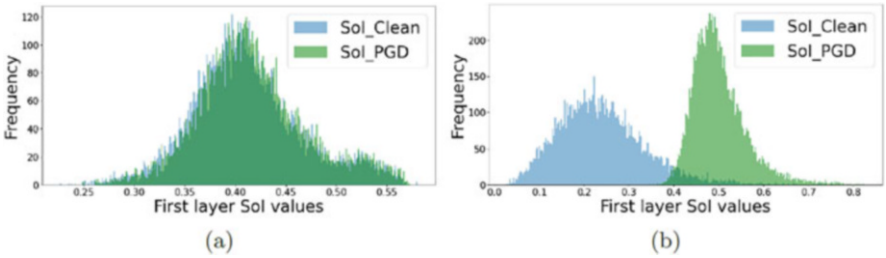


Fig. 9 (**a**) The clean and adversarial SoI distributions at the first layer have an inherent separation which motivates the use of SoI like hardware signature for adversarial detection. (**b**) After Phase1 training, the SoI distributions are separated. For both the figures, SoI PGD corresponds to first layer SoI values for PGD with $\epsilon = 16/255$

shown in Eq. (8). Here, we scale down the cross-entropy loss $\mathcal{L}_{CE}$ by a small value (of the order 10^{-3}). The loss function minimizes the distance between the desired SoI values (λ_c and λ_a) and the calculated mean of the SoI distributions (SoI_c and SoI_a).

The Phase1 training effectively increases the distance between the clean and adversarial SoI distributions as seen in Fig. 9b. At this stage, strong adversarial attacks are easily detected as a result of large SoI separation. However, weak attacks are not sufficiently detected as they have small SoI separation with the clean samples. Note, here strong attacks refer to adversarial attacks with high L_∞ distance (large ϵ value) and vice versa. Further, the DNN has very low accuracy on clean inputs as the cross-entropy loss was significantly scaled down during Phase1 training. To improve the DNN's accuracy on clean inputs and robustness against weak adversarial attacks, we employ Phase2 adversarial training.

$$\mathcal{L} = \beta \mathcal{L}_{CE} + y\mathcal{L}_{MSE}(SoI_a, \lambda_a) + (1-y)\mathcal{L}_{MSE}(SoI_c, \lambda_c) \tag{8}$$

In the Phase2 training, we freeze the first layer of the DNN and perform adversarial training [8, 48] with weak adversarial attacks. Freezing the first layer weights preserves the SoI separation at the first layer obtained after Phase1 training. Finally, after the dual-phase training, a high clean accuracy is obtained. Additionally, weak adversarial attacks are suitably classified while strong attacks are detected.

After the Phase2 training, we create a SoI-Probability Look-Up Table (LUT) that classifies a given SoI value as a clean or an adversarial sample. As seen in Fig. 10, we randomly sample a set of clean images from the training set and create their adversarial counterparts using PGD $\epsilon = 8/255$ attack. Then, we compute the clean and adversarial SoI distributions ($\mathcal{D}_c$ and $\mathcal{D}_a$). Using, $\mathcal{D}_c$ and $\mathcal{D}_a$, we compute the P(clean) values using Eq. (9). Here, n_c and n_a are the number of clean and adversarial samples, respectively, at a particular SoI value. A high *P(clean)* value

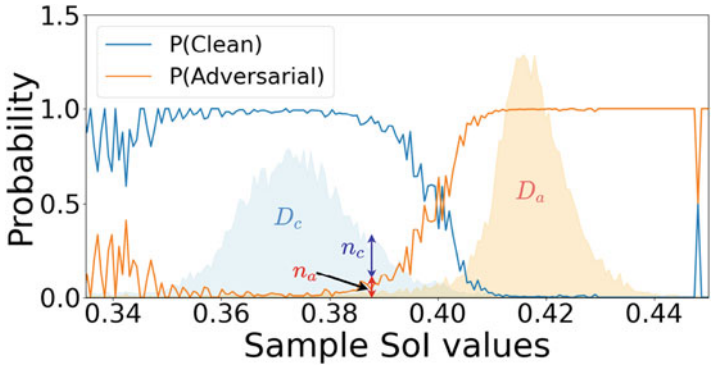


Fig. 10 The SoI-Probability LUT contains sample SoI values and their corresponding P(clean) values. It classifies a given SoI sample as clean or adversarial

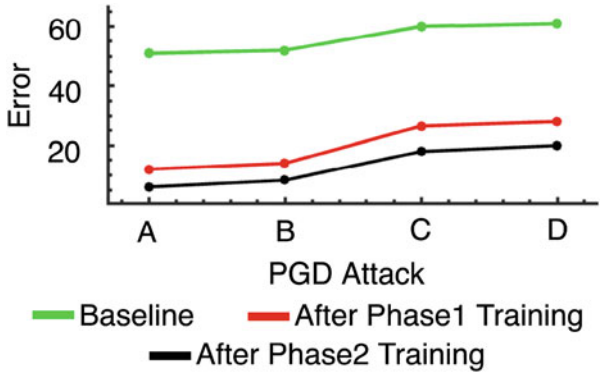


Fig. 11 After Phase1 training, some weak adversarial attacks might go undetected. Phase2 adversarial training helps classify the weak adversarial samples that further brings down the error

signifies that a given SoI value corresponds to a clean sample and vice versa. The SoI-Probability LUT contains sample SoI values and their corresponding *P(clean)* values.

$$P(Clean) = \frac{n_c}{n_c + n_a} \tag{9}$$

In Fig. 11, we show the efficacy of the Phase2 training in defending against weak PGD attacks. For this we plot the error values of a baseline DNN (without DetectX), a DNN with the first layer subjected to Phase1 training followed by Phase2 training. Here, error is defined as the amount of adversarial attacks that are undetected and are misclassified by the DNN. Clearly, most of the weak attacks are suitably detected after Phase1 training leading to a large drop in the error value. However, Phase 2 adversarial training helps classify the undetected weak attacks correctly leading to a further drop in error.

We implement the dual-phase trained DNN on an analog crossbar-based end-to-end DNN evaluation platform called Neurosim [49]. Neurosim [49] is a Python-based platform that performs a holistic energy-latency-accuracy evaluation of analog crossbar-based DNN accelerators. The Neurosim platform supports both SRAM and memristive computing devices (ReRAM and FeFET). For adversarial detection, we design a fully digital DetectX module (shown in Fig. 12) on 32 nm CMOS that contains digital circuits to compute the SoI value and the SoI-Probability LUT that is used to classify a given SoI value as clean or adversarial. For hardware evaluation, a dual-phase trained VGG16 model (trained on CIFAR100) is implemented on a 128×128 memristive crossbar with device on-off ratio of 10 and $R_{on} = 10\,\text{k}\Omega$. The DetectX module is appended at the end of the first layer crossbar. The energy evaluation of the DetectX module is performed using SPICE simulations. Based on the 128×128 crossbar Neurosim implementation, we find that the DetectX module only adds 2.6 nJ to the hardware cost for adversarial

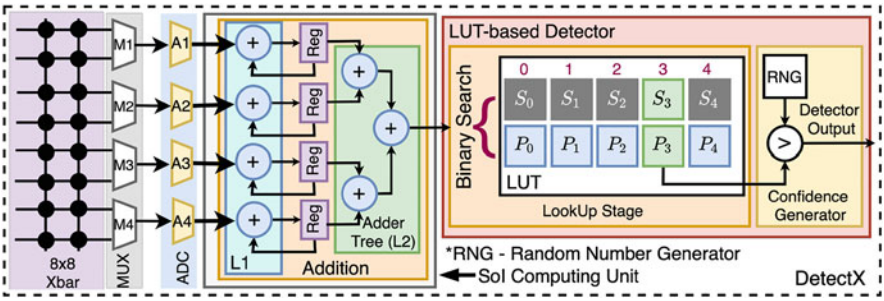


Fig. 12 The DetectX module is implemented on a fully digital 32 nm CMOS technology. It can directly interface with an analog crossbar (8×8 crossbar shown for illustration). The module contains circuits for computing the SoI signature and classifying the SoI value as clean or adversarial

CIFAR10 VGG8 Phase1- PGD (ϵ = 32/255), Phase2 – PGD (ϵ = 4/255)				
	FGSM (ϵ = 16/255)	**PGD (ϵ = 8/255)**	**PGD (ϵ = 16/255)**	**PGD (ϵ = 32/255)**
ROC-AUC Score	0.97	0.895	1	1
Accuracy (D/B)	80.1 / 87			
Error (D/B)	1.7 / 35.3	9.1 / 53.2	0 / 66.7	0 / 67.18
CIFAR100 VGG16 Phase1- PGD (ϵ = 32/255), Phase2 – PGD (ϵ = 4/255)				
ROC-AUC Score	0.98	0.99	1	1
Accuracy (D/B)	50.1 / 60.2			
Error (D/B)	5.1 / 59.4	0 / 96.6	0 / 97.7	0 / 97.7
TinyImagenet ResNet18 Phase1- PGD (ϵ = 32/255), Phase2 – PGD (ϵ = 4/255)				
ROC-AUC Score	0.84	0.86	0.98	1
Accuracy (D/B)	42.2 / 53.3			
Error (D/B)	13.2 / 56.5	12.36 / 90.7	10.1 / 91.2	0.1 / 93

Fig. 13 ROC-AUC scores, Error and Accuracy values for the DNN + DetectX system with different image datasets and adversarial attacks. We mention the adversarial attacks used for Phase1 and Phase2 training corresponding to each dataset. *D* and *B* denote the DNN + DetectX system and baseline model, respectively. The baseline model is a DNN trained on clean inputs using standard stochastic gradient descent and does not contain the DetectX module

detection. Compared to prior adversarial detection works that use large neural network-based detector modules [16–18], DetectX consumes about 25x less energy for adversarial detection.

DetectX significantly improves the adversarial robustness of the DNN. In Fig. 13, we show the ROC-AUC score of the DetectX module under different FGSM and PGD attacks across CIFAR10, CIFAR100 and TinyImagenet datasets. A high ROC-AUC score greater than 0.5 denotes reliable adversarial detection. Due to the high ROC-AUC score, the error of the DNN + DetectX system is significantly lower compared to the baseline DNN without the DetectX module. Further, due to the introduction of the DetectX module, the accuracy on clean inputs slightly drops. However, the drop is marginally low.

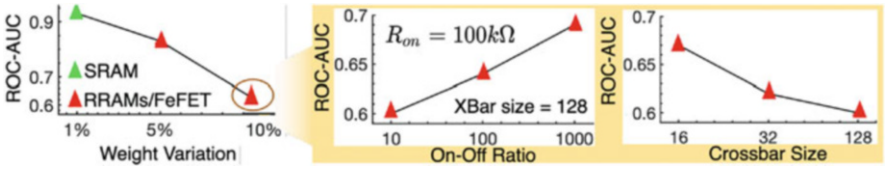


Fig. 14 With increasing device-device variations in a memristive crossbar (left), the adversarial detection performance decreases slightly. Further, with increasing device on-off ratio (middle), and decreasing crossbar sizes (right), the detection performance increases. Non-idealities negatively impact the detection performance of DetectX

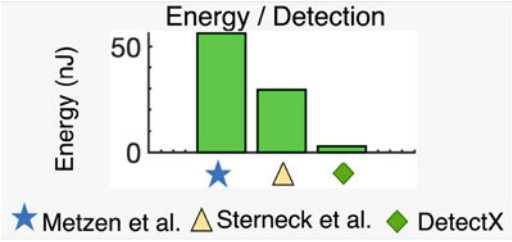


Fig. 15 Energy required per detection operation for different works [16, 18]. DetectX consumes more than 25x less energy for detection compared to prior works

As DetectX is integrated in a crossbar platform like Neurosim, it is important to understand the effects of crossbar non-idealities on the detection performance of DetectX. Figure 14(left) shows that the detection performance decreases with increasing device-device variations in the memristive crossbars [50]. The device variations introduce variations in the SoI value computation which ultimately negatively affect the detection performance. However, even at large weight variations, the ROC-AUC score is still greater than 0.5 which suggests reliable detection. Next, we show the effects of different memristive device on-off ratios (Fig. 14(middle)) and crossbar sizes (Fig. 14(right)) on DetectX's detection performance. These results are shown for the memristive device with 10% weight variations (shown with a circle). Evidently, the detection performance increases with higher on-off ratios and lower crossbar sizes. This is because with higher on-off ratios and lower crossbar sizes, the non-ideal effects in crossbars decrease.

We further show how the DetectX method consumes significantly low energy for adversarial detection compared to prior detection works [16–18]. Prior works use large neural networks to perform adversarial detection. For a fair comparison, these neural network-based detectors are implemented on the Neurosim platform [49] and the energies are evaluated. As seen in Fig. 15 DetectX consumes about 50x less energy compared to Metzen et al. [16] and 25x less energy compared to Sterneck et al. [18]. Note, here the energy values represent the energy required for a single detection operation.

5 Unleashing Robustness to Structure-Pruned DNNs Implemented on Crossbars with Non-idealities

In the recent years, several crossbar-aware pruning techniques have been devised that yield sparse DNN models. Owing to their high sparsity, these models require significantly lower number of crossbars to be mapped, thereby introducing hardware resource-efficiency not only in terms of crossbars but also peripheral circuits interfacing the crossbars. Pruning algorithms such as, [28–31], produce structured sparsity in DNNs that fit into crossbars as dense weight matrices [32]. These structured pruning algorithms claim to preserve the accuracy of the pruned DNNs, after implementation on crossbars, with minimal or no noticeable loss, while bringing in high energy- and area-efficiencies. However, none of these works has included the impact of the inexorable non-idealities (see Sect. 2.2) during inference on crossbars. For a realistic hardware evaluation of the performance of increased structured sparsity in DNNs mapped on crossbars, the inclusion of hardware non-idealities is critical. In this section, we introduce a recent work [33] that draws the focus of the research community towards a non-ideality aware evaluation of various existing structured pruning algorithms and shows how increased sparsity can degrade the robustness of DNNs on non-ideal crossbars. It also introduces two hardware-centric non-ideality mitigation strategies, namely crossbar-column rearrangement and *Weight-Constrained-Training* (WCT), to help improve the performance or robustness of the sparse DNNs on crossbars with little or no training overheads.

Crossbar-aware structured pruning of DNNs: There have been numerous works on structured pruning of DNNs, such as *channel/filter pruning* or C/F pruning (see Fig. 16(top)) wherein the unimportant filters and channels in a DNN (corresponding to rows and columns in the weight matrix of the DNN) are pruned to obtain a sparse 2D weight matrix [28, 29]. These pruned models result in significant hardware savings in terms of reduced number of crossbars for mapping, thereby bringing in energy- and area-efficiency for DNN implementation. Likewise, other crossbar-aware pruning strategies include *Crossbar-Column Sparsity* (XCS) [30] or *Crossbar-Row Sparsity* (XRS) [31] (XCS shown in Fig. 16(bottom)) that exploit fine-grained sparsity by, respectively, pruning columns or rows of weights within a crossbar [32]. Additionally, these works have claimed to preserve the inference accuracy of the structure-pruned networks on crossbars with minimal or no discernible performance loss with respect to the unpruned ones. However, none of the previous works has accounted for the non-idealities inherent in crossbar arrays which raises concerns about the claimed performance of the highly pruned models in the real scenario.

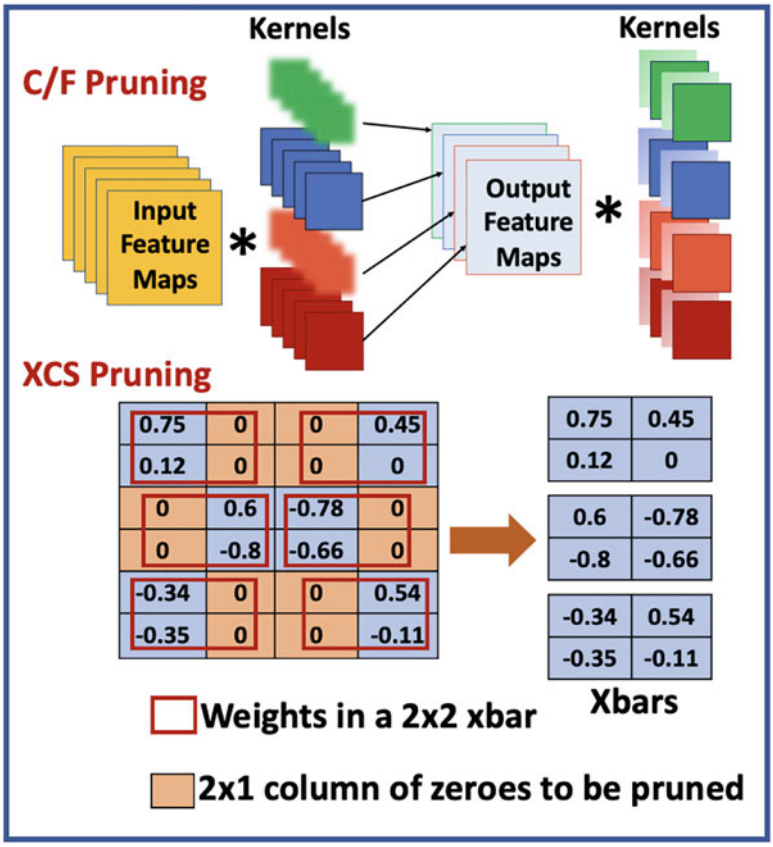


Fig. 16 **Top**: A representation of channel/filter pruning (C/F pruning). The blurred channels/filters correspond to DNN weights pruned in a structured manner. **Bottom**: A representation of XCS pruning (shown for a 4×4 weight matrix mapped onto 2×2 crossbars) that generates fine-grained sparsity along crossbar columns

5.1 Hardware Evaluation Framework for Non-ideality Integration During Inference

To map pretrained DNNs onto non-ideal memristive crossbars and investigate the cumulative impact of the circuit and device-level non-idealities on their performance during inference, a simulation framework in PyTorch is used by Bhattacharjee et al. [33] as shown in Fig. 17). In the platform, a Python wrapper is built that unrolls each and every convolution operation in the software DNN into MAC operations. This yields 2D weight matrices for each DNN layer which are to be partitioned into numerous crossbar instances. Before partitioning, based on the structured pruning approach, the following transformations T on the sparse weight matrices W are applied:

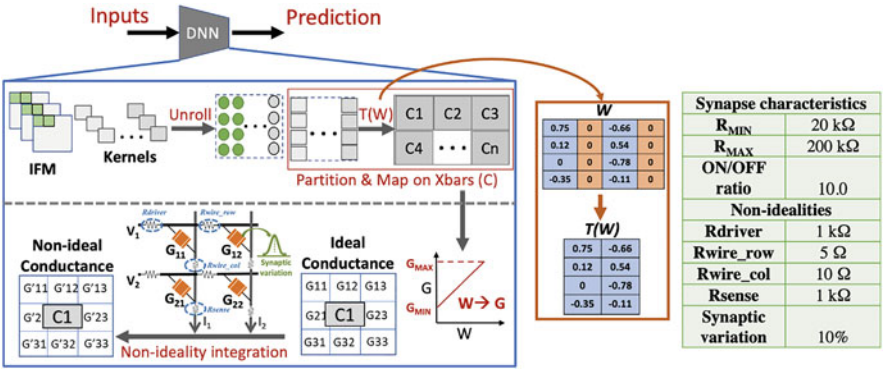


Fig. 17 Python-based hardware evaluation framework for non-ideality aware DNN inference

1. $T(W)$ **for C/F pruning:** Here, for a given 2D weight matrix of a DNN layer, all the columns bearing zero values are eliminated. Further, we also eliminate rows of the weight matrix of the next DNN layer that interact with the output feature maps corresponding to the columns of zero values in the previous layer.
2. $T(W)$ **for XCS (or XRS):** Here, within a given 2D weight matrix of a DNN layer, there are chunks of successive zero weight vectors of the size of crossbar-column (or crossbar-row) (see Fig. 16-Bottom) which are eliminated.

Note, for standard unpruned DNNs, the $T(W)$ is not required. The resulting transformed weight matrices are then partitioned into multiple crossbar instances. The subsequent stage of the platform converts the weights in the crossbars to suitable conductances G (between G_{MIN} and G_{MAX}). Thereafter, the circuit-level non-idealities (interconnect parasitics) and synaptic device variations are integrated with the conductances. The various synapse parameters (e.g. R_{MIN}, R_{MAX}, device ON/OFF ratio) and values of the non-idealities used for the subsequent experiments are listed in the table shown in Fig. 17.

5.2 Are Structure-Pruned DNNs Also Robust on Hardware?

In [33], VGG11 and VGG16 DNNs area trained with structured sparsity (via C/F pruning, XCS or XRS) using benchmark datasets such as, CIFAR10 and CIFAR100. For the experiments with CIFAR10 dataset, the sparsity is set as $s = 0.8$, while with CIFAR100 dataset, the sparsity is $s = 0.6$. The unpruned and pruned DNN models are trained to have nearly equal software accuracies to conduct a fair comparison of the impact of non-idealities when the models are mapped onto non-ideal crossbars (see Table 1). The crossbar-compression-rates for the structure-pruned DNNs on 32×32 crossbars are also shown in Table 1.

Table 1 Table showing software accuracies and crossbar-compression-rates (with 32×32 crossbars) for the various DNN models with CIFAR10 and CIFAR100 datasets

Dataset: CIFAR10	Software accuracy (%) ‖ Crossbar-compression-rate			
Network	Unpruned	C/F ($s = 0.8$)	XCS ($s = 0.8$)	XRS ($s = 0.8$)
VGG11	83.6 ‖ –	83.5 ‖ 19.69×	83.28 ‖ 4.26×	82.67 ‖ 4.88×
VGG16	84.48 ‖ –	83.65 ‖ 19.60×	82.06 ‖ 5.57×	83.47 ‖ 4.89×
Dataset: CIFAR100	Software accuracy (%)			
Network	Unpruned	C/F ($s = 0.6$)		
VGG11	53.29 ‖ –	52.72 ‖ 5.64×		
VGG16	51.83 ‖ –	50.55 ‖ 4.20×		

We find that the sparse DNNs have greatly reduced number of parameters than their unpruned counterparts which results in significantly lesser number of crossbars on hardware. However, the fewer parameters remaining in the sparse DNNs are crucial for the model's performance. Thus, any non-ideality interfering with the fewer parameters of the sparse DNNs would have huge impact on the DNN accuracy and hence, robustness on hardware. In Fig. 18a, we find that for the VGG11/CIFAR10 model, the DNNs with structured sparsity (via C/F pruning, XCS, XRS with $s = 0.8$) suffer greater accuracy degradation than their unpruned counterparts for crossbar sizes ranging from 16×16 to 64×64. Further, as we increase the crossbar size, both the accuracies of unpruned and pruned networks decline owing to increase in crossbar non-idealities [34, 39]. Specifically, on 64×64 crossbars, the inference accuracy of the unpruned model reduces by $\sim$21% with respect to the software baseline while, for the sparse DNNs pruned via C/F pruning, XCS and XRS, the decline is $\sim$39%, $\sim$24% and $\sim$30%, respectively. Also, in Fig. 18b, we find that on reducing the extent of sparsity in the C/F pruned DNNs from $s = 0.8$ to $s = 0.5$, the performance degradation suffered by the pruned DNNs is reduced. This validates the fact that greater sparsity, although leads to energy- and area-efficient mappings on crossbars, increases the interference of crossbar non-idealities, thereby hampering the performance and hence, the robustness of the pruned networks.

In Fig. 18c, for the VGG16 DNN with CIFAR10 dataset, the trends are similar to the case of the VGG11 DNN for XCS, XRS, and C/F pruning ($s = 0.8$) in case of 16×16 and 32×32 crossbars. However, in case of a larger 64×64 crossbar, we find that the performance of the network pruned by C/F pruning exceeds that of the unpruned network. This is because unpruned DNNs require a larger absolute number of crossbars for mapping than pruned ones. As a result, the value of NF is expected to increase at a higher rate for unpruned DNN on moving from 32×32 to 64×64 crossbars (see Fig. 18d). So, for larger crossbars, the accuracy degradation for structure-pruned DNNs would decelerate compared to their unpruned counterparts, which can even lead to better absolute accuracy of the pruned networks than the unpruned ones.

Next, we discuss two crossbar-aware non-ideality mitigation strategies that can help improve the robustness of structure-pruned DNNs on non-ideal crossbars.

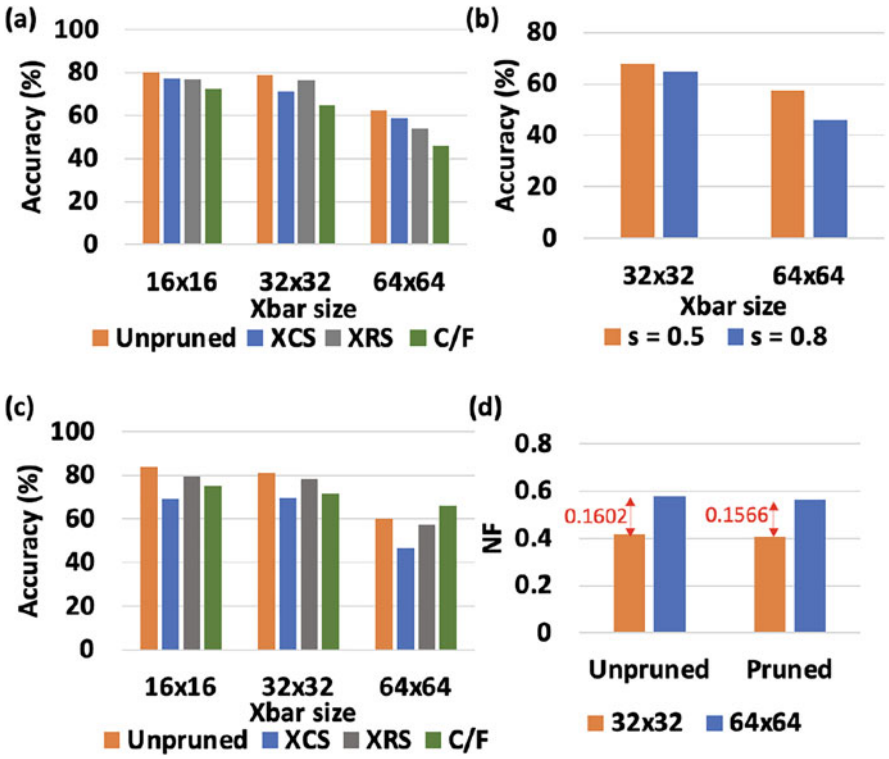


Fig. 18 Plot of inference accuracy versus crossbar size for (**a**) unpruned and structure-pruned ($s = 0.8$) VGG11/CIFAR10 DNN. (**b**) Different values of sparsity (s) of a C/F pruned VGG11/CIFAR10 DNN. (**c**) Unpruned and structure-pruned ($s = 0.8$) VGG16/CIFAR10 DNN. (**d**) Plot showing the variation in average NF for unpruned and C/F pruned weight matrices on increasing the crossbar size from 32×32 to 64×64

5.3 Non-ideality Mitigation Strategies for Increased Robustness of Structure-Pruned DNNs

1. **Crossbar-Column rearrangement** (R)**:** For the sparse DNNs obtained via C/F pruning, a simple hardware-friendly transformation of column rearrangement R has been proposed mapping weights onto non-ideal crossbars. This transformation is inspired from the fact that the impact of non-idealities (or non-ideality factor NF) reduces for crossbars with higher proportion of low conductance synapses [26, 44]. Additionally, this approach of column rearrangement does not have any training overhead and is applied before the mapping of the DNNs onto crossbars.
   To understand column rearrangement R, consider a 4×6 weight matrix W, after applying the transformation T, to be mapped onto 2×2 crossbars (see Fig. 19a). During the R transformation, we first compute the value of $(\mu \times \sigma)^{\frac{1}{2}}$

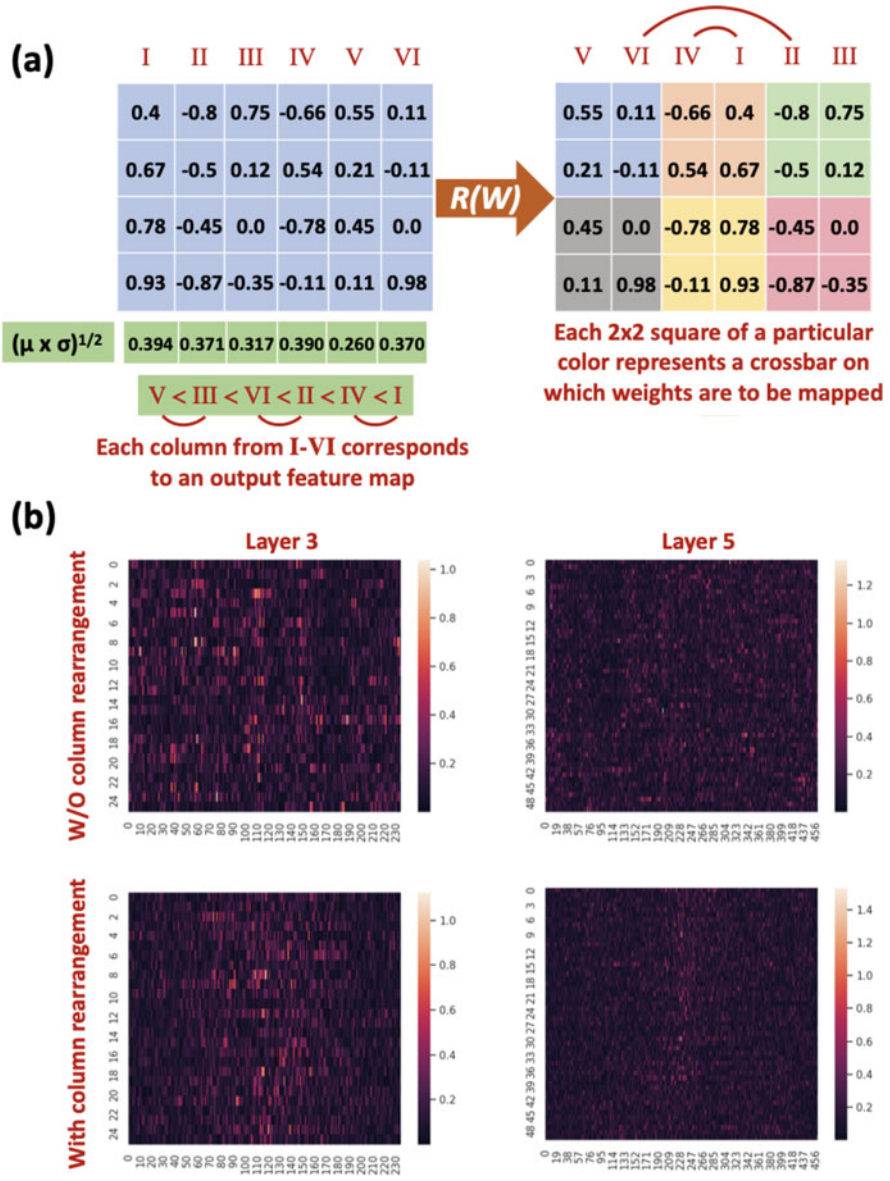


Fig. 19 (**a**) Pictorial representation of R transformation. (**b**) Heatmaps to visualize the impact of R transformation on the weight matrices of 3rd and 5th layers of the VGG16/CIFAR10 DNN trained with C/F pruning with $s = 0.8$

for each column from I–VI, where μ and σ, respectively, denote the mean and standard deviation of the absolute values of weights in each column. Thereafter, based on the increasing order of $(\mu \times \sigma)^{\frac{1}{2}}$, we rearrange columns I–VI in

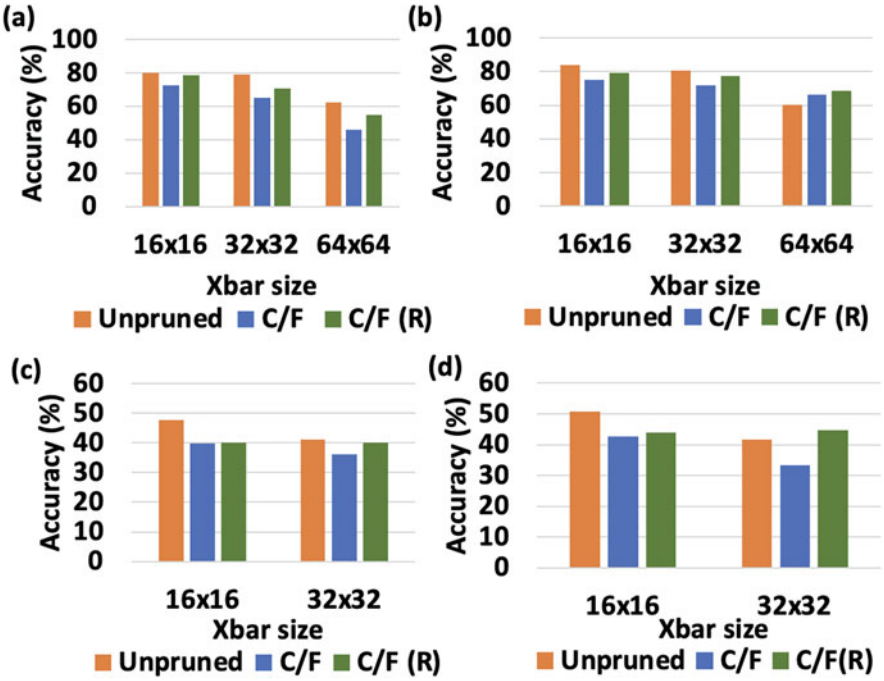


Fig. 20 A plot of inference accuracy versus crossbar size for unpruned, C/F pruned and (**a**) C/F pruned with transformation R ($s = 0.8$) VGG11/CIFAR10 DNNs. (**b**) C/F pruned with transformation R ($s = 0.8$) VGG16/CIFAR10 DNNs. (**c**) C/F pruned with transformation R ($s = 0.6$) VGG11/CIFAR100 DNNs. (**d**) C/F pruned with transformation R ($s = 0.6$) VGG16/CIFAR100 DNNs

the manner shown. Now, in Fig. 19b, the impact of R transformation can be visualized on the weight matrices of the 3rd and 5th convolutional layers of the VGG16/CIFAR10 DNN (C/F pruned with $s = 0.8$) using heatmaps. Before applying the transformation, the lighter (low conductance synapse) and darker (high conductance synapse) points in the heatmaps are intermixed. Post transformation, the lighter points are concentrated at the center of the heatmaps and darker points are mostly near the peripheries. Thus, post R transformation, when the DNN weight matrices are partitioned into multiple crossbar instances, majority of the crossbars have greater proportions of low conductance synapses, thereby mitigating the impact of crossbar non-idealities.

Figure 20a,b and c,d show that R transformation improves the performance of the C/F pruned VGG11 and VGG16 DNNs. Specifically, ∼9% (∼6%) improvement in accuracy is observed for VGG11 (VGG16) DNN on 64×64 (32×32) crossbars with CIFAR10 dataset. We also find that on 32×32 crossbars, the accuracy of the pruned VGG16/CIFAR100 DNN post R transformation is ∼3% greater than the unpruned counterpart.

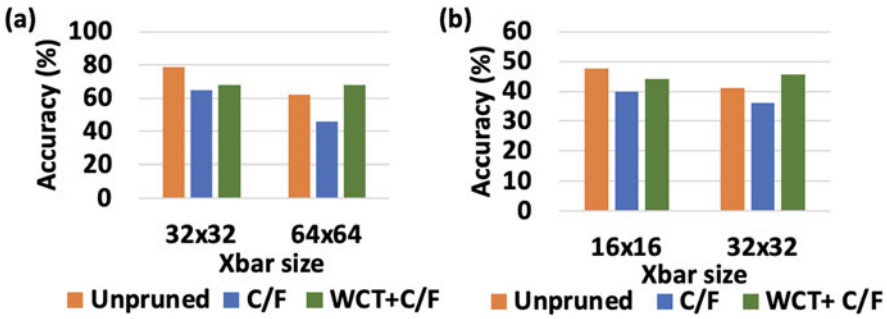


Fig. 21 A plot of inference accuracy versus crossbar size for unpruned, C/F pruned and (**a**) WCT + C/F pruned ($s = 0.8$) VGG11/CIFAR10 DNNs. (**b**) WCT + C/F pruned ($s = 0.6$) VGG11/CIFAR100 DNNs

2. **Weight-Constrained-Training (WCT):** WCT is another non-ideality mitigation technique for the structure-pruned DNNs that is motivated by the NEAT method described in Sect. 3. In WCT, based on the weight distribution of all the layers of a trained DNN, a cut-off value W_{cut} is heuristically determined, and the following transformation is then applied on the weights of the DNN: $W = min\{|W|, W_{cut})\} * sign(W)$. This transformation constrains the DNN weights in the interval $[-W_{cut}, W_{cut}]$. With the above transformation, the DNN is iteratively trained for 1–2 epochs, to maintain nearly iso-accuracy with baseline. Note, the iterative training via WCT does not add any computational overhead to the overall training time, thereby making it a viable choice. Similar to NEAT, a WCT-DNN also results in greater proportion of low conductance states on the crossbars, thus, reducing the impact of non-idealities. The resultant sparse WCT-DNNs are then mapped onto crossbars. In Fig. 21a,b, we find that the WCT-DNNs maintain their performance even on increasing the crossbar size, making them robust against crossbar non-idealities. Further, WCT-DNNs have better accuracy than the C/F pruned DNNs on crossbars. Specifically, with CIFAR10 (CIFAR100) dataset, the WCT-DNN has ∼6% (∼7%) higher accuracy than the unpruned model on 64×64 (32×32) crossbars.

6 Related Works

Recently, several crossbar-based In-Memory Computation (IMC) architectures and frameworks have been proposed for efficiency-driven acceleration of DNNs [50–54]. CONV-SRAM [50] proposes an energy-efficient static random access memory (SRAM) with embedded dot-product computation capability, for the inference of convolutional neural networks with binary weights. On the other hand, Kim et al. [52] and Gokmen et al. [53] have proposed an architecture based on CMOS-based

resistive processing unit (RPU) devices to achieve significant acceleration in DNN training.

In ISAAC [51], Shafiee et al. designed and characterized a pipelined memristive crossbar architecture and proposed a weight encoding scheme that reduces the analog-to-digital conversion overheads. Additionally, Marinella et al. [54] implement an ReRAM crossbar-based DNN acceleration platform and characterize the energy, latency, and area of the peripheral and crossbar components across different technology nodes. Besides crossbar-based DNN acceleration platforms, end-to-end hardware evaluation platforms such as Neurosim and PUMA [37, 49] provide software-based scalable solutions to perform hardware evaluation of crossbar implementations. While Neurosim [49] considers only NVM device variations during DNN evaluation, PUMA [37] models other circuit-level and data-dependent non-idealities by incorporating GenieX [34]. Few recent works such as RxNN [23] and GenieX [34] have delved deeper into modelling the characteristics of non-ideal crossbars. These non-idealities include crossbar-interconnect parasitics and data-dependent selector non-linearities. While RxNN can suitably compute data-independent non-idealities, GenieX incorporates both data-dependent and independent crossbar non-idealities. With this, they provide accurate hardware-realistic inference performance of crossbar-mapped DNNs.

However, none of these works has explored non-ideality aware crossbar-mapping of DNN models for adversarial robustness. Furthermore, these works have not delved into the correlation existing between sparsity in network weights and crossbar non-idealities to highlight the vulnerabilities of sparse DNNs. Additionally in prior works, the possibility of using inherent hardware signatures in the detection of adversarial attacks and building adversarial robustness for crossbar-mapped DNN models has not been well explored. This motivates us to present recent works involving non-ideality aware mapping of DNNs onto crossbars for improving their classification accuracy (robustness) in normal and/or adversarial scenarios [26, 33]. In addition, examining hardware signatures in crossbars for energy-efficient adversarial detection [27] is a key facet of this chapter. We present a summary table (Table 2) for the convenience of the readers to qualitatively compare the scope of different works based on memristive crossbar arrays pertaining to DNN inference acceleration.

7 Conclusion

This chapter elucidates recent advances in the energy-efficient and robust implementation of DNNs on memristive crossbar array platforms. Specifically, we come across works that use hardware-driven methods to improve the adversarial security of DNNs on noisy crossbars without additional overhead of retraining or reduced energy-efficiency. The first work (NEAT) improves the adversarial classification capabilities on DNNs on crossbars, while the other work (DetectX) is an adversarial detection method to guarantee robustness on crossbar platforms. Additionally, this

Table 2 Table comparing the scope of different memristive crossbar-based works for DNNs. The works discussed in this chapter—DetectX [27], NEAT [26], and Bhattacharjee et al. [33] have specifically added a new dimension of adversarial and sparsity-aware robustness which have not been looked into in prior works

	Xbar acceleration		End-to-end H/W	Robustness		
Work	Efficiency-driven	Novel weight mapping	Evaluation	Sparsity-aware	Non-ideality	Adversarial attacks
CONV-SRAM [50]	✓	×	×	×	×	×
ISAAC [51]						
Kim et al. [52]						
Gokmen et al. [53]						
Marinella et al. [54]						
Neurosim [49]	✓	×	✓	×	✓	×
PUMA [37]						
RxNN [23]	✓	×	×	×	✓	×
GenieX [34]						
DetectX [27]	✓	×	×	×	✓	✓
NEAT [26]	✓	✓	×	×	✓	✓
Bhattacharjee et al. [33]	✓	✓	×	✓	✓	×

chapter highlights a study which corroborates that although increased structured sparsity in weights is beneficial for resource-efficient implementation of DNNs on crossbars, it compromises their classification accuracy (robustness) in a non-ideal scenario. To this end, various hardware-based non-ideality mitigation approaches have been proposed to improve the performance and hence, the robustness of sparse DNNs on crossbars.

References

1. Reagen, B., et al.: Minerva: enabling low-power, highly-accurate deep neural network accelerators. In: 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), pp. 267–278. IEEE, Piscataway (2016)
2. Hadidi, R., et al.: Characterizing the deployment of deep neural networks on commercial edge devices. In: 2019 IEEE International Symposium on Workload Characterization (IISWC), pp. 35–48. IEEE, Piscataway (2019)
3. Chakraborty, I., et al.: Pathways to efficient neuromorphic computing with non-volatile memory technologies. Appl. Phys. Rev. (2020)
4. Wong, H.-S.P., et al.: Metal–oxide RRAM. Proc. IEEE **100**(6), 1951–1970 (2012)
5. Dodge, S., Karam, L.: A study and comparison of human and deep learning recognition performance under visual distortions. In: 2017 26th International Conference on Computer Communication and Networks (ICCCN), pp. 1–7. IEEE, Piscataway (2017)

6. Kurakin, A., Goodfellow, I., Bengio, S.: Adversarial machine learning at scale (2016). Preprint. arXiv:1611.01236
7. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples (2014). Preprint. arXiv:1412.6572
8. Madry, A., et al.: Towards deep learning models resistant to adversarial attacks (2017). Preprint. arXiv:1706.06083
9. Carlini, N., et al.: On evaluating adversarial robustness (2019). Preprint. arXiv:1902.06705
10. Lin, J., Gan, C., Han, S.: Defensive quantization: when efficiency meets robustness (2019). Preprint. arXiv:1904.08444
11. Qiu, H., et al.: Mitigating advanced adversarial attacks with more advanced gradient obfuscation techniques (2020). Preprint. arXiv:2005.13712
12. Guo, C., et al.: Countering adversarial images using input transformations (2017). Preprint. arXiv:1711.00117
13. Prakash, A., et al.: Deflecting adversarial attacks with pixel deflection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 8571–8580 (2018)
14. Xie, C., et al.: Mitigating adversarial effects through randomization (2017). Preprint. arXiv:1711.01991
15. Buckman, J., et al.: Thermometer encoding: one hot way to resist adversarial examples. In: International Conference on Learning Representations (2018)
16. Metzen, J.H., et al.: On detecting adversarial perturbations (2017). Preprint. arXiv:1702.04267
17. Yin, X., Kolouri, S., Rohde, G.K.: Gat: generative adversarial training for adversarial example detection and robust classification. In: International Conference on Learning Representations (2019)
18. Sterneck, R., Moitra, A., Panda, P.: Noise sensitivity-based energy efficient and robust adversary detection in neural networks. In: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (2021)
19. Panda, P., Chakraborty, I., Roy, K.: Discretization based solutions for secure machine learning against adversarial attacks. IEEE Access **7**, 70157–70168 (2019)
20. Gui, S., et al.: Model compression with adversarial robustness: a unified optimization framework. In: Wallach, H., et al. (eds.) Advances in Neural Information Processing Systems, vol. 32. Curran Associates Inc., Red Hook (2019)
21. Sehwag, V., et al.: Hydra: pruning adversarially robust neural networks. Adv. Neural Inf. Proces. Syst. **33**, 19655–19666 (2020)
22. Panda, P.: QUANOS-adversarial noise sensitivity driven hybrid quantization of neural networks. Preprint. arXiv:2004.11233 (2020)
23. Jain, S., et al.: RxNN: a framework for evaluating deep neural networks on resistive crossbars. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. (2020)
24. Bhattacharjee, A., Panda, P.: Rethinking non-idealities in memristive crossbars for adversarial robustness in neural networks (2020). Preprint. arXiv:2008.11298
25. Roy, D., et al.: Robustness hidden in plain sight: can analog computing defend against adversarial attacks? (2020). arXiv: 2008.1201
26. Bhattacharjee, A., et al.: NEAT: non-linearity aware training for accurate, energy-efficient and robust implementation of neural networks on 1T-1R crossbars. In: IEEE Transactions on Computer-Aided Design (2021)
27. Moitra, A., et al.: DetectX – adversarial input detection using current signatures in memristive XBar arrays. In: IEEE Transactions on Circuits and Systems I (2021)
28. Wen, W., et al.: Learning structured sparsity in deep neural networks (2016). Preprint. arXiv:1608.03665
29. Wang, P., et al.: SNrram: an efficient sparse neural network computation architecture based on resistive random-access memory. In: 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC) (2018)
30. Liang, L., et al.: Crossbar-aware neural network pruning. IEEE Access (2018)

31. Lin, J., et al.: Learning the sparsity for ReRAM: mapping and pruning sparse neural network for ReRAM based accelerator. In: ASPDAC '19: Proceedings of the 24th Asia and South Pacific Design Automation Conference (2019)
32. Chu, C., et al.: PIM-prune: fine-grain DCNN pruning for crossbar-based process-in-memory architecture. In: 2020 57th ACM/IEEE Design Automation Conference (DAC) (2020)
33. Bhattacharjee, A., Bhatnagar, L., Panda, P.: Examining and mitigating the impact of crossbar non-idealities for accurate implementation of sparse deep neural networks. In: 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE) (2022)
34. Chakraborty, I., et al.: GENIEx: a generalized approach to emulating non-ideality in memristive Xbars using neural networks (2020). Preprint. arXiv:2003.06902
35. Liu, B., et al.: Vortex: variation-aware training for memristor x-bar. In: Proceedings of the 52nd Annual Design Automation Conference, pp. 1–6 (2015)
36. Lee, S., et al.: Learning to predict IR drop with effective training for ReRAM-based neural network hardware. In: 2020 57th ACM/IEEE Design Automation Conference (DAC), pp. 1–6. IEEE, Piscataway (2020)
37. Ankit, A., et al.: PUMA: a programmable ultra-efficient memristor-based accelerator for machine learning inference. In: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 715–731 (2019)
38. Ansari, M., et al.: PHAX: physical characteristics aware ex-situ training framework for inverter-based memristive neuromorphic circuits. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **37**(8), 1602–1613 (2017)
39. Bhattacharjee, A., Moitra, A., Panda, P., Efficiency-driven hardware optimization for adversarially robust neural networks. In: Design, Automation and Test in Europe Conference (DATE) (2021)
40. Chen, P.-Y., et al.: Mitigating effects of non-ideal synaptic device characteristics for on-chip learning. In: 2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 194–199. IEEE, Piscataway (2015)
41. Agrawal, A., Lee, C., Roy, K.: X-CHANGR: changing memristive crossbar mapping for mitigating line-resistance induced accuracy degradation in deep neural networks (2019). Preprint. arXiv:1907.00285
42. Liu, B., et al.: Reduction and IR-drop compensations techniques for reliable neuromorphic computing systems. In: 2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 63–70. IEEE, Piscataway (2014)
43. He, Z., et al.: Noise injection adaption: end-to-end ReRAM crossbar nonideal effect adaption for neural network mapping. In: Proceedings of the 56th Annual Design Automation Conference 2019, pp. 1–6 (2019)
44. Bhattacharjee, A., et al.: SwitchX: gmin-gmax Switching for energy-efficient and robust implementation of binary neural networks on ReRAM Xbars (2021). Preprint. arXiv:2011.14498
45. Sun, X., Yu, S.: Impact of non-ideal characteristics of resistive synaptic devices on implementing convolutional neural networks. IEEE J. Emerging Sel. Top. Circuits Syst. **9**(3), 570–579 (2019)
46. Li, T., et al.: Sneak-path based test and diagnosis for 1r RRAM crossbar using voltage bias technique. In: Proceedings of the 54th Annual Design Automation Conference 2017, pp. 1–6 (2017)
47. Wang, Z., et al.: Ferroelectric tunnel memristor-based neuromorphic network with 1T1R crossbar architecture. In: 2014 International Joint Conference on Neural Networks (IJCNN), pp. 29–34. IEEE, Piscataway (2014)
48. He, Z., Rakin, A.S., Fan, D.: Parametric noise injection: trainable randomness to improve deep neural network robustness against adversarial attack. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 588–597 (2019)
49. Chen, P.-Y., Peng, X., Yu, S.: NeuroSim: a circuit-level macro model for benchmarking neuro-inspired architectures in online learning. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **37**(12), 3067–3080 (2018)

50. Biswas, A., Chandrakasan, A.P.: CONV-SRAM: an energy-efficient SRAM with in-memory dot-product computation for low-power convolutional neural networks. IEEE J. Solid State Circuits **54**(1), 217–230 (2018)
51. Shafiee, A., et al.: ISAAC: a convolutional neural network accelerator with in-situ analog arithmetic in crossbars. ACM SIGARCH Comput. Archit. News **44**(3), 14–26 (2016)
52. Kim, S., et al.: Analog CMOS-based resistive processing unit for deep neural network training. In: 2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), pp. 422–425. IEEE, Piscataway (2017)
53. Gokmen, T., Vlasov, Y., Acceleration of deep neural network training with resistive cross-point devices: design considerations. Front. Neurosci. **10**, 333 (2016)
54. Marinella, M.J., et al.: Multiscale co-design analysis of energy, latency, area, and accuracy of a ReRAM analog neural training accelerator. IEEE J. Emerging Sel. Top. Circuits Syst. **8**(1), 86–101 (2018)

Adversarial ML for DNNs, CapsNets, and SNNs at the Edge

Alberto Marchisio, Muhammad Abdullah Hanif, and Muhammad Shafique

1 Introduction

The use-cases of Machine Learning (ML) applications have been significantly growing in recent years. Among the ML models, Deep Neural Networks (DNNs), which stack several layers of neurons, have demonstrated to solve complex tasks with high accuracy. Capsule Networks (CapsNets) have established as prominent ML models due to their high learning capabilities. Moreover, Spiking Neural Networks (SNNs) emerged as biologically plausible models, in which their spike event-based communication provides energy-efficient capabilities to be employed in low-power and resource-constrained devices [9, 10].

On the other hand, ML systems are expected to be reliable against multiple security threats. Several studies highlighted that one of the most critical issues is represented by the adversarial attacks, i.e., small and imperceptible input perturbations that cause misclassifications. Moreover, as highlighted in Fig. 1, also other ML vulnerabilities cause serious concerns questioning the deployment of ML models in safety-critical applications. Therefore, the ML community analyzed and proposed several attack methodologies and defensive countermeasures [77]. While the attacks and defenses for DNNs have been extensively studied, the security of advanced ML models such as CapsNets and SNNs is still in its emerging phase and needs more thorough investigations.

After discussing the security challenges for ML systems and the taxonomy of adversarial ML, this chapter provides an overview of the security threats for DNNs,

A. Marchisio (✉)
Technische Universität Wien (TU Wien), Vienna, Austria
e-mail: alberto.marchisio@tuwien.ac.at

M. A. Hanif · M. Shafique
eBrain Lab, Division of Engineering, New York University Abu Dhabi, Abu Dhabi, UAE
e-mail: mh6117@nyu.edu; muhammad.shafique@nyu.edu

S. Pasricha, M. Shafique (eds.), *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing*, https://doi.org/10.1007/978-3-031-40677-5_18

CapsNets, and SNNs, focusing on recent advancements, current trends, and unique possibilities for specific ML models to enhance their robustness.

2 Security Challenges for ML

Recent works [14, 77, 78, 96] have shown that ML-based systems are vulnerable to different types of security and reliability threats (see Fig. 1), which can span from maliciously injected perturbations, such as adversarial attacks, hardware Trojans, or injected faults, to natural misfunctioning of the system, like permanent faults generated during chip fabrication, aging, and process variations. Moreover, the leakage of sensitive and confidential data, including the intellectual property of the ML model (e.g., architecture and parameters) and training dataset, have raised several privacy issues. While the adversarial ML issues will be extensively discussed in the rest of the chapter, this section briefly introduces the other types of vulnerabilities.

2.1 ML Privacy

Due to the massive performance and computational power of high-end GPU-HPC workstations, it is possible to conduct ML tasks using a massive amount of data on a large scale. If such data is collected from users' private information, such as private images, interests, web searches, and clinical records, the ML deployment toolchain will have access to sensitive information that could potentially be mishandled. The privacy attacks for ML can be classified into two categories, namely Model Extraction Attacks and Model Inversion Attacks. While the former category aim

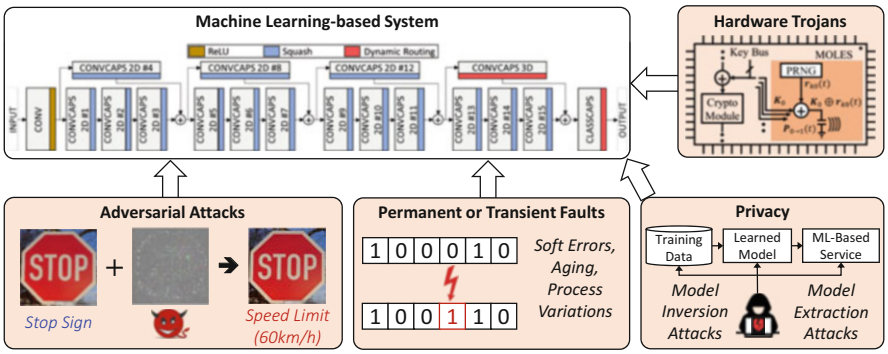


Fig. 1 Vulnerability threats for ML-based systems, their manifestation and impact on their functionality

at extracting private information of the ML model (e.g., model parameter, model architecture), the latter threatens the sensitive features of the training data.

- *Model Extraction Attacks:* The goal of the adversary is to duplicate the parameters and hyperparameters of the model to provide ML services, and to compromise the ML algorithms' confidentiality and intellectual property of the service provider [87, 92].
- *Model Inversion Attacks:* The adversary aims to infer sensitive information from the training data. Membership inference attacks [81] can infer whether a sensitive record belongs to the training set when the ML model is overfitted, While Property Inference Attacks [19] infer specific properties that only hold for a fraction of the training data.

There are currently four possible categories of techniques that can be applied to avoid these leakages of sensitive information:

- *Differential Privacy:* The goal is to prevent the adversary from inferring whether a specific data was used to train the target model, such that the ML algorithm learns to extract features of the training data without disclosing sensitive information about individuals. The privacy is guaranteed through a randomization mechanism, which could be based on injecting noise into the stochastic gradient descent process (Noisy SGD [1]) or through the Private Aggregation of Teacher Ensembles (PATE) method [65], in which a "student" model receives the knowledge transferred from an ensemble of "teacher" models.
- *Homomorphic Encryption:* It is an encryption scheme $x \rightarrow y$, in which the ML computations are conducted on ciphertexts y, and the decrypted output in plaintext x matches the result that would have been computed without encryption. As long as the decryption key is unknown to the adversary, the data remains confidential. Since the Fully Homomorphic Encryption (FHE) system [20] dramatically increases the computational complexity of the ML algorithm, a partial homomorphic encryption system [63] which supports only certain operations in the ciphertext domain, such as additions or multiplications, is more suited for complex computations. In the context of ML, CryptoNets [21] performs DNN inference on encrypted data, while Nandakumar et al. [60] extends the encryption support to the whole training process.
- *Secure Multi-Party Computation:* The basic idea consists of distributing the training/testing data across multiple servers and training/inferring the ML model together, while each server does not have access to the training/testing data of the other servers. Different privacy-preserving ML protocols have been proposed, including SecureML [59], MiniONN [44], DeepSecure [74], Gazelle [31], and SecureNN [91].
- *Trusted Execution Environment:* Additional hardware is used to create a secure and isolated computation environment in which the ML algorithms are executed [47]. In this way, the e integrity and confidentiality of the data and codes loaded inside the protected regions are guaranteed.

However, these privacy-preserving methods significantly increase the computational overhead and require customization for specific ML models at the software and hardware levels to improve the efficiency of computations.

2.2 Fault Injection and Hardware Trojans on ML Systems

Hardware-level security vulnerabilities for ML systems include fault injection techniques (e.g., bit-flips) and the injected hardware Trojans into ML accelerators. Generically speaking, an adversary can flip the bits of data stored into the SRAM and DRAM memory cells through laser injection [2] or Row-Hammer attacks [35].

- *Fault Injection Attack Methodologies* aim at finding the most sensitive locations in which to inject faults [45, 89]. The Bit-Flip Attack [72] finds the most vulnerable bits of the ML model parameters using a progressive bit search method, while the Practical Fault Attack [7] injects faults into ML activations.
- *Hardware Trojans* are maliciously introduced hardware injected during chip fabrication that only activate when triggered. They represent serious threats when the hardware devices are manufactured in off-shore fabrication facilities, thus increasing the risk of facing untrusted supply chains. In the context of ML accelerators, Clements et al. [12] designed hardware Trojans for the ML activation function, and in NeuroAttack, the Trojan consists of flipping the values of certain bits of ML models. In both methods, the hardware Trojan is triggered by a carefully designed input pattern.

The defensive countermeasures to mitigate against the above-discussed vulnerabilities are based on improving the resiliency of ML accelerators and memory systems and detecting Trojans.

- *Fault tolerance methods*, similarly to the soft error mitigation methodologies, aim at improving the resiliency of ML applications. Such defensive techniques are based on hardware redundancy [62], range restriction [11], or weight reconstruction [40]. More specifically, the algorithm-based fault tolerance (ABFT) method [97] detects and corrects errors in the convolutional layers.
- *Trojan detection methods* are based on runtime monitoring [18] of the ML accelerator. The operations executed in the hardware device are constantly monitored, and any eventual functionality violation due to an inserted hardware Trojan or other reasons can be immediately detected and notified.

2.3 ML Systems Reliability Threats

Unlike the vulnerability threats that are intentionally injected by malicious adversaries, ML systems are subjected to reliability threats that undermine their correct

functionality. The continuous underscaling of the technology nodes in which the chips are fabricated has significantly increased the probability that hardware circuits are affected by permanent or transient faults and has accelerated the aging process.

- *Permanent Faults:* These process variations represent imperfections that are generated during the fabrication of integrated circuits [71]. High rates of such process variations result in permanent faults, which dramatically decrease the yield of the wafer fabrication.
- *Transient Faults:* Soft errors are bit-flips caused by high-energy particle strikes or induced by other radiation events [6]. They are categorized as transient errors since the faulty cells are not permanently damaged, but these faults vanish once new data is written into the same locations.
- *Aging:* The electronic circuits gradually degrade over time [32], due to various physical phenomena, like Hot Carrier Injection (HCI), Bias Temperature Instability (BTI), and Electromigration (EM). These effects can manifest as transistors' threshold voltage increase, which causes timing errors and permanent faults over time.

Conventional fault mitigation techniques such as Dual Modular Redundancy (DMR) [88], Triple Modular Redundancy (TMR) [48], and Error-Correcting Codes (ECC) [69] can be applied, but they incur huge overheads, which makes them impractical for ML applications. Therefore, ad-hoc cost-effective mitigation techniques need to be applied.

- *Permanent faults mitigation:* To mitigate permanent faults due to process variations in ML accelerators, different techniques have been proposed. Fault-Aware Training (FAT) and Fault-Aware Pruning (FAP) [95] incorporate the information of faults into the training process and bypass the faulty components. To avoid the re-training overhead, Fault-Aware Mapping techniques such as SalvageDNN [28] are based on mapping the least significant weights on the faulty units.
- *Soft error mitigation:* To mitigate transient faults, generic fault-tolerant methods like Ranger [11] and ABFT [97] can be applied. Moreover, FT-ClipAct [30] uses clipped activation functions that are mapped into pre-specified values within a range that has the lowest impact on the output, and Sanity-Check [61] protects fully connected and convolutional layers of ML models employing spatial and temporal checksums that exploit the linearity property.
- *Aging mitigation:* The effects of timing errors that occur in the computational units of ML accelerators can be mitigated with ThUnderVolt [94] and GreenTPU [64]. The NBTI aging of on-chip SRAM-based memory cells in ML accelerators is mitigated with the DNN-Life framework [29] that employs read and write transducers to balance the duty-cycle in each SRAM cell.

3 Taxonomy of Adversarial ML

Given an ML model M, an input x, and its output prediction label y_{true}, the goal of classical ML is to make a correct prediction, i.e., the predicted output $y = M(x)$ is equal to y_{true}. On the contrary, an adversarial attack method aims at generating a misclassification by introducing a small noise ε to the input, such that the adversarial example $x' = x + \varepsilon$ is incorrectly classified ($M(x') \neq y_{true}$). Due to the wide variety of adversarial attack typologies and threat models, it is important to define a common taxonomy for their categorization. Towards this, we discuss four different features of adversarial attacks and their possible types. An overview of the taxonomy is shown in Fig. 2.

- *Attacker Knowledge:* It refers to what is the threat model in which the adversary operates and what are the accessible data and features. In white-box attacks, the adversary has full knowledge about the ML model, its parameters, the training algorithm, and the training data. On the contrary, black-box attacks assume no knowledge about the ML model. Hence the adversary can only craft an adversarial example by sending a series of queries and analyzing the vulnerability based on the corresponding outputs. Moreover, in the literature, there exist different attacker knowledge assumption models referred to as grey-box attacks, in which the adversary knows more features than for black-box attacks, but does not have full access like under the white-box assumption.

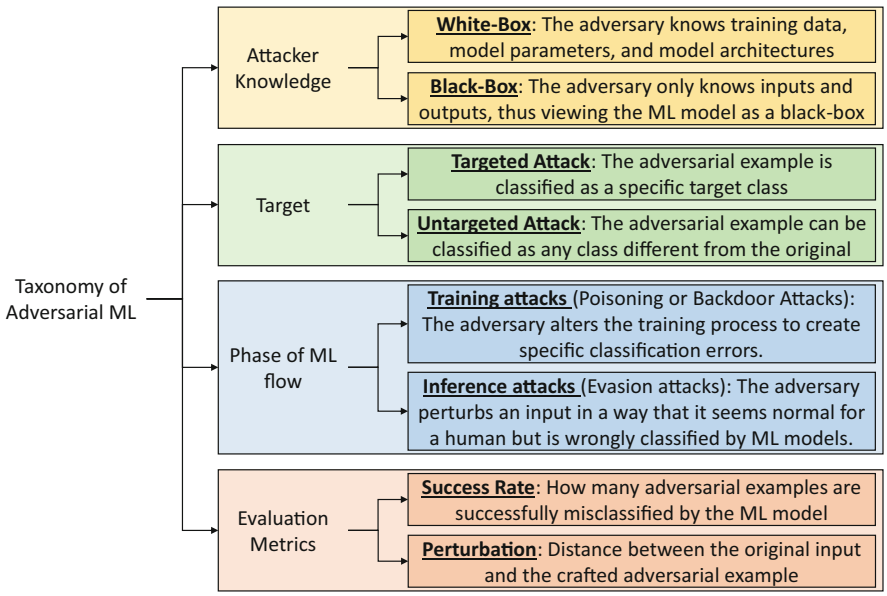


Fig. 2 Categorization of different types of adversarial attack methods and their taxonomy

- *Adversarial Goal:* It refers to the scope of the attack algorithm. If the goal is simply a misclassification, the attack is untargeted since any class different from the correct one can be the prediction of the adversarial example. On the other hand, in a targeted attack, the adversary produces adversarial examples that force the output of the ML model to predict a specific class.
- *Phase of ML Flow:* It refers to the stage of the ML development in which the adversary operates. In training attacks, the adversary poisons the training data by injecting carefully designed samples to force the ML model to learn wrong features that can later be used to generate specific misclassifications. On the contrary, in evasion attacks that operate at the inference stage, the adversary tries to evade the system by crafting malicious samples that force the ML model to make false predictions.
- *Evaluation Metrics:* It refers to the quantitative methods for measuring the strengths of the attacks, and easily accessible comparison metrics. To evaluate the robustness of the attack, the success rate measures the number of adversarial examples that are misclassified by the ML model. Since a well-designed attack needs to be imperceptible, i.e., hardly distinguishable from the original input by a human eye, the perturbation measures the distance between the adversarial example and the original (clean) input.

4 Security for DNNs

Due to their high accuracy on many tasks, DNNs are prime candidate algorithms to be applied to safety-critical applications. However, due to the security vulnerabilities that undermine their correct functionality, several defensive countermeasures need to be applied. An overview of adversarial attacks and defenses applied to the DNN design flow is shown in Fig. 3.

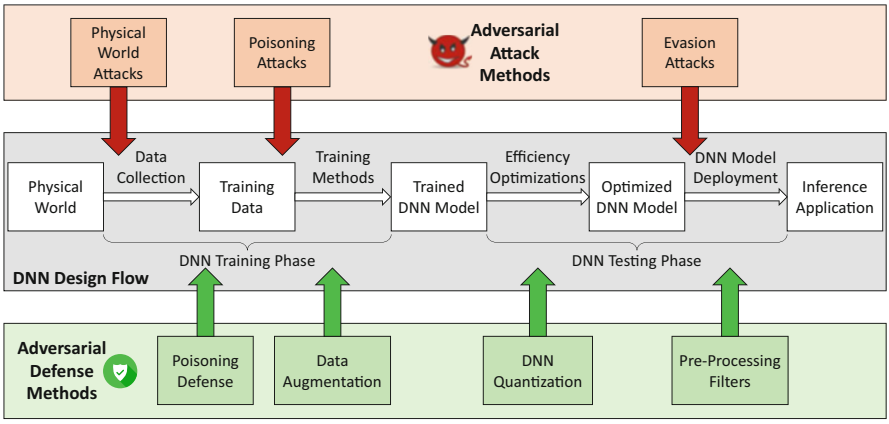


Fig. 3 Adversarial attacks and defenses applied in different stages of the DNN design flow

4.1 Adversarial Attacks

As previously discussed, the adversarial attacks can be categorized into different types based on the adversary's knowledge, goal, and phase of the ML flow. Due to the mainstream usage of DNNs, several attack methodologies have been proposed. The following list discusses the most prominent ones:

- *Poisoning Attacks:* At the training stage, the training data can be poisoned with contaminated inputs. Based on the principles of Genetic Adversarial Networks (GANs), Goodfellow et al. [22] devised a procedure to generate samples similar to the training set, having almost identical distribution. This method inspired many of the successive adversarial attack methodologies. Poisoning Attacks [76] alter the training dataset to modify the decision boundaries of the DNN classifiers. Backdoor Attacks [24] aim at training the DNN for a carefully crafted noise pattern (acting as a backdoor) while maintaining high accuracy on its intended task. However, when such a backdoor trigger is present at the input of the DNN, a targeted misclassification is achieved.
- *Evasion Attacks:* Different evasion attack methodologies were proposed. In white-box settings, gradient-based attacks like the Fast Gradient Sign Method (FGSM) [23] and its iterative version, the Projected Gradient Descent (PGD) [50] exploit the gradient of the DNN output predictions w.r.t. the inputs to craft the adversarial perturbations as imperceptible noise that make the DNN classifier cross the decision boundary. In black-box settings, the One Pixel Attack [85] demonstrated to misclassify DNN models by changing only one pixel intensity. fakeWeather attacks [55] emulate the effect of atmospheric conditions to fool DNNs. Decision-based attacks [8] are a subset of evasion attacks in which the adversary does not have access to the output probabilities but only to the prediction. For instance, the FaDec attack [34] jointly optimizes the number of queries and the perturbation distance between the adversarial example and the clean example to fool DNNs.
- *Attacks in the Physical World:* While the aforementioned attacks mainly make modifications in the experimental settings, the adversarial attacks can also be applied in real life by introducing physical modifications [38]. Examples of physical world attacks have been showcased in the context of road sign classification by adding stickers [17], in the context of object detection by adding adversarial patches [86], or in face detection using eyeglasses with special frames [79].

4.2 Adversarial Defenses

The large variety of adversarial attacks led to the design of several types of defenses, which can be summarized and grouped into the following categories:

- *Poisoning Defenses:* To mitigate against poisoning attacks, several defensive countermeasures have been proposed. Outlier detection-based defenses [67] filter out training sample outliers, which most likely correspond to poisoned samples. Since typically backdoor attacks exploit the sparsity of DNNs, the Fine-Pruning method [46] defends against backdoor attacks by eliminating the neurons that are dormant for clean inputs in the backdoor network.
- *Data Augmentation:* The basic principle of Adversarial Training [50] is to extend the training example with the adversarial examples, for instance, generated with the PGD attack. In this way, the DNN models achieve higher robustness against such perturbations. This method is considered very effective to defend against adversarial attacks, but its high computation overhead pushes the community to search for efficient optimizations of this procedure.
- *Quantization:* The optimization techniques employed to improve the energy - efficiency of DNNs can also achieve higher robustness against adversarial attacks. The Defensive Quantization method [42] demonstrated that the adversarial noise magnitude remains contained in quantized DNNs. The QuSecNets method [33] selects the quantization levels based on the DNN resilience and computes the appropriate quantization threshold values based on an optimization function. Other approaches, such as Defensive Approximation [27] are promising, but the work of Siddique et al. [83] demonstrated that approximate computing cannot be referred to as a universal defense technique against adversarial attacks.
- *Pre-Processing Filters:* Another common technique to improve the DNN robustness against adversarial attacks is to employ pre-processing filters. The basic idea of this approach is to view the adversarial perturbation as a noise added to the input, which can be filtered out at runtime. Methods based on Sobel filters [3] and randomized smoothing [13] demonstrated that the pre-processing filters have a smoothing effect and significantly reduce the adversarial success rate.

5 Security for Capsule Networks

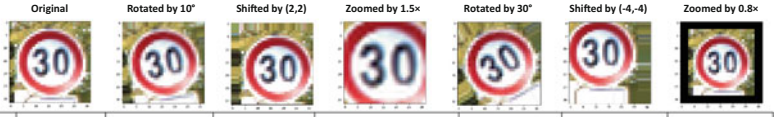
CapsNets have emerged as efficient ML models which encode hierarchical information of the features through multi-dimensional capsules [75]. Based on the principle of inverse graphics, the CapsNets from the image pixels encode the pose of low-level features, and from these low-level features encode the higher-level entities. Moreover, to overcome the translation-invariance issue that affects traditional DNNs, the max-pooling layers are replaced by the iterative dynamic routing-by-agreement algorithm, which determines the values of the coupling coefficients between low-level capsules and higher-level capsules at runtime. Therefore, it is key to analyze the security vulnerabilities of CapsNets and compare their robustness to the traditional DNNs.

5.1 Robustness Against Affine Transformations

Before studying the vulnerability of CapsNets under adversarial examples, their robustness against affine transformation is studied [51]. This analysis is key to determining how affine transformations, which are perceptible yet plausible perturbations appearing in the real world, can or cannot fool the networks under investigation. We apply three different types of transformations, which are rotation, shift, and zoom, on the images of the GTSRB dataset [84]. For the evaluation, we compare the CapsNet model [36] with a 9-layer VGGNet [93] and a 5-layer LeNet [39].

Figure 4 shows some examples of affine transformations applied to the images of the GTSRB dataset. Both the CapsNet and the VGGNet can be fooled by some affine transformations, like zoom or shift, while the prediction confidence of the CapsNet is lower. Moreover, as expected, the LeNet is more vulnerable to this kind of transformations due to its lower number of layers and parameters compared to the VGGNet. The CapsNet, on the other hand, is able to overcome a lower complexity than the VGGNet in terms of the number of layers and parameters. Indeed, as noticed in the example of the STOP image rotated by 30°, the confidence is lower, but both the CapsNet and the VGGNet are able to classify it correctly, while the LeNet is fooled.

CapsNet	Prediction	STOP	STOP	STOP	ROAD WORK	STOP	YELD	STOP
	Probability	0.057	0.056	0.050	0.026	0.032	0.037	0.054
VGGNet	Prediction	STOP	STOP	STOP	GENERAL CAUTION	STOP	YIELD	STOP
	Probability	1.000	0.999	0.999	0.735	0.747	0.900	0.999
LeNet	Prediction	STOP	STOP	STOP	YIELD	GO STRAIGHT OR LEFT	YIELD	STOP
	Probability	0.999	0.999	0.984	0.574	0.668	0.999	0.989



CapsNet	Prediction	SPEED LIMIT 30	SPEED LIMIT 30	SPEED LIMIT 30	BYCICLES CROSSING	KEEP RIGHT	TRAFFIC SIGNALS	SPEED LIMIT 30
	Probability	0.056	0.055	0.045	0.032	0.027	0.024	0.058
VGGNet	Prediction	SPEED LIMIT 30	SPEED LIMIT 30	SPEED LIMIT 30	ROUNDABOUT MANDATORY	SPEED LIMIT 50	SPEED LIMIT 30	SPEED LIMIT 30
	Probability	1.000	1.000	1.000	0.629	0.433	0.870	1.000
LeNet	Prediction	SPEED LIMIT 30	SPEED LIMIT 30	SPEED LIMIT 30	ROUNDABOUT MANDATORY	SPEED LIMIT 50	SPEED LIMIT 50	SPEED LIMIT 30
	Probability	1.000	1.000	0.999	0.830	0.999	0.440	0.999

Fig. 4 Predicted classes and their probability associated with the prediction confidence, comparing the CapsNet, VGGNet, and LeNet, under different affine transformations applied to two examples of the GTSRB dataset [51]

5.2 *Robustness Against Adversarial Attacks*

Besides the vulnerability against affine transformations, the robustness against adversarial attacks is a key metric to analyze when evaluating the security. The CapsAttack methodology [51] evaluates the adversarial robustness of CapsNets and other DNNs under a novel adversarial attack generation algorithm (see Fig. 5) and analyzes in detail the output probability variations of single images under attack.

5.2.1 Adversarial Attack Methodology

The goal of an efficient adversarial attack is to generate imperceptible and robust examples to fool the network. An adversarial example can be defined *imperceptible* if the modifications of the original sample are so small that humans cannot notice them. Therefore, the perturbations added in high variance zones are less evident and more difficult to be detected, compared to the perturbations applied in low variance pixels. To measure the imperceptibility, we measure the distance D between the original sample X and the adversarial sample X*. This value indicates the total amount of perturbation added to all the pixels in the image. We also define D_{MAX} as the maximum total perturbation tolerated by the human eye.

Moreover, an adversarial example can be defined *robust* if the gap between the probability of the target class and the probability of the highest class is maximized. A gap increase makes the adversarial example more robust, since the modifications of the probabilities caused by the image transformations (e.g., resizing or compression) tend to be less effective. Indeed, if the gap is high, a small variation of the probabilities may not be sufficient to change the prediction.

As shown in Fig. 5, the CapsAttacks methodology is based on an iterative procedure that automatically produces targeted imperceptible and robust adversarial examples in a black-box setting [51]. The input image is modified to maximize

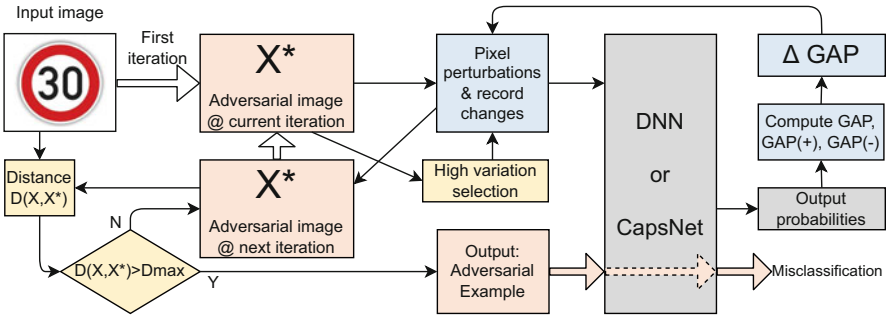


Fig. 5 The CapsAttacks methodology [51] to generate adversarial examples. The blue-colored boxes work towards fooling the network, while the yellow-colored boxes control the imperceptibility of the adversarial example

the gap (*imperceptibility*) until the distance between the original image and the adversarial example is lower than D_{MAX} (*robustness*). The perturbations are applied to a set of pixels in the highest variation regions at every iteration to create imperceptible perturbations. Moreover, the algorithm automatically decides whether it is more effective to add or subtract the noise to maximize the gap according to the values of the two parameters $GAP(+)$ and $GAP(-)$. These mechanisms increase the imperceptibility and the robustness of the attack.

5.2.2 Evaluation Results

The CapsAttacks methodology is applied to the previously described CapsNet [36], LeNet [39] and VGGNet [93], tested on different examples of the GTSRB dataset [84].

The CapsNet is tested on two different examples, shown in Fig. 6a (Example 1) and Fig. 6e (Example 2). For the first one, we analyze two cases to test the dependence on the target class:

- **Case I:** the target class is the class relative to the second-highest probability between all the initial output probabilities.
- **Case II:** the target class is the class relative to the fifth-highest probability between all the initial output probabilities.

The analyses of the examples in Case I and Case II lead to the following observations:

1. The CapsNet classifies the input image shown in Fig. 6a as the "120 km/h speed limit" (S8) class with a probability equal to 0.0370.
   The target class for Case I is "Double curve" (S21) with a probability equal to 0.0297. After 13 iterations, the image (in Fig. 6b) is classified as "Double curve" with a probability equal to 0.0339. Hence, *the probability of the target class has overcome the initial one*, as shown in Fig. 7a. At this iteration, the distance $D(X^*, X)$ is equal to 434.20. If we increase the number of iterations, the robustness of the attack will increase as well since the gap between the two probabilities increases. However, the adversarial noise becomes more perceptible. Indeed, the distance at the iteration 16 overcomes $D_{MAX} = 520$ (see Fig. 6c).

(a) (b) (c) (d) (e) (f) (g)

Fig. 6 Images for the attack applied to the CapsNet: (**a**) Original input image of Example 1. (**b**) Image misclassified by the CapsNet at iteration 13 for Case I. (**c**) Image misclassified by the CapsNet at iteration 16 for Case I. (**d**) Image at iteration 12 for Case II. (**e**) Original input image of Example 2. (**f**) Image at iteration 5, applied to the CapsNet. (**g**) Image misclassified by the CapsNet at iteration 21

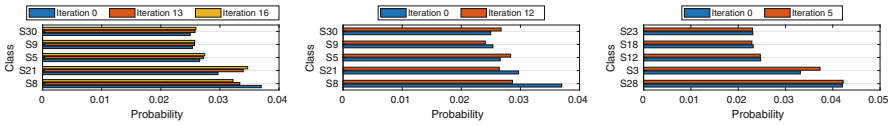


Fig. 7 CapsNet results: (**a**) Output probabilities of Example 1—Case I: the blue bars represent the starting probabilities, the orange bars the probabilities at the point of misclassification, and the yellow bars at the D_{MAX}. (**b**) Output probabilities of Example 1—Case II: the blue bars represent the starting probabilities, and the orange bars the probabilities at the D_{MAX}. (**c**) Output probabilities of Example 2: the blue bars represent the starting probabilities, and the orange bars the probabilities at the D_{MAX}

For the Case II, the probability of the target class "Beware of ice/snow" (S30) is equal to 0.0249, as shown in Fig. 7b. The gap between the highest probability and the probability of the target class is larger than the gap in Case I. After 12 iterations, the CapsNet still correctly classifies the image (see Fig. 6d). Indeed, Fig. 7b shows that the gap between the two classes is lower, but not enough for a misclassification. However, the distance at this iteration overcomes $D_{MAX} = 520$. This experiment shows that the algorithm would need more iterations to misclassify, at the cost of more perceivable perturbations.

2. The CapsNet classifies the input image shown in Fig. 6e as the "Children crossing" (S28) class with a probability equal to 0.042. The target class, which is "60 km/h speed limit" (S3), has a probability equal to 0.0331. After 5 iterations, the distance overcomes $D_{MAX} = 250$, while the network has not misclassified the image yet (see Fig. 6f), because the probability of the target class has not overcome the initial highest probability, as shown in Fig. 7c. The misclassification is noticed at the iteration 21 (see Fig. 6g). However, the perturbation is very perceivable.

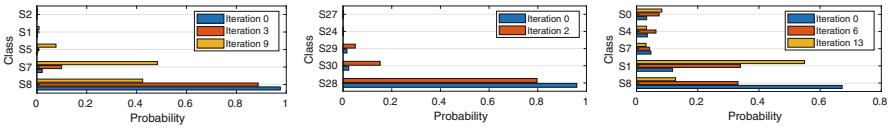
The same two examples are evaluated to compare the robustness of the CapsNet and the 9-layer VGGNet. For the Example 1, only Case I is analyzed as benchmark. Since the VGGNet classifies the input images with different output probabilities compared to the ones obtained by the CapsNet, the evaluation of how much the VGGNet is resistant to the attack is based on the gap measured at the same distance. To compare the robustness of the CapsNet and the 5-layer LeNet, we only analyze the Example 1, since the original Example 2 is incorrectly classified by the LeNet.

From the results in Figs. 8 and 9, we can make the following observations:

1. The VGGNet classifies the input image (in Fig. 8a) as the "120 km/h speed limit" (S8) class with a probability equal to 0.976. The target class, which is "100 km/h speed limit" (S7), has a probability equal to 0.021. After 3 iterations, the distance overcomes $D_{MAX} = 520$, while the VGGNet has not misclassified the image yet (see Fig. 8b) yet, since the two initial probabilities were very distant, as shown in Fig. 9a. The algorithm would need to perform 9 iterations (see Fig. 8c) to fool the VGGNet, where the probability of the target class is 0.483.

(a) (b) (c) (d) (e) (f) (g)

Fig. 8 Images for the attack applied to the DNNs: (**a**) Original input image of Example 1. (**b**) Image at iteration 3, applied to the VGGNet. (**c**) Image at iteration 9, misclassified by the VGGNet. (**d**) Original input image of Example 2. (**e**) Image at iteration 2, applied to the VGGNet. (**a**) Image at iteration 6, misclassified by the LeNet. (**b**) Image at iteration 13, misclassified by the LeNet



(a) (b) (c)

Fig. 9 DNNs results: (**a**) Output probabilities for the Example 1 on the VGGNet: the blue bars represent the starting probabilities, the orange bars the probabilities at the point of misclassification, and the yellow bars at the D_{MAX}. (**b**) Output probabilities for the Example 2 on the VGGNet: the blue bars represent the starting probabilities, and the orange bars the probabilities at the D_{MAX}. (**c**) Output probabilities for the Example 1 on the LeNet: the blue bars represent the starting probabilities, the orange bars the probabilities at the point of misclassification, and the yellow bars at the D_{MAX}

2. The VGGNet classifies the input image (in Fig. 8d) as the "Children crossing" (S28) class with a probability equal to 0.96. The target class, which is "Beware of ice/snow" (S30), has a probability equal to 0.023. After 2 iterations, the distance overcomes $D_{MAX} = 250$, while the VGGNet has not misclassified the image yet (see Fig. 8e). As in the previous example, this behavior is due to the high distance between the initial probabilities, as shown in Fig. 9b. Note that the VGGNet reaches D_{MAX} in a lower number of iterations compared to the CapsNet.
3. The LeNet classifies the input image (in Fig. 8a) as the "120 km/h speed limit" (S8) class with a probability equal to 0.672. The target class, which is "30 km/h speed limit" (S1), has a probability equal to 0.178. After 6 iterations, the LeNet is fooled, because the image (in Fig. 8f) is recognized as the target class with a probability equal to 0.339. The noise becomes perceptible after 13 iterations (Fig. 8g), where the distance overcomes $D_{MAX} = 520$.

5.3 Discussion

While it is highly complex to formalize generic conclusions, a common trend is that the CapsNets are more robust against adversarial attacks and affine transformation than DNNs with similar depth and number of parameters. These observations are aligned with similar works of Michels et al. [58] and Gu et al. [25].

Concurrently, the CapsNets security has been analyzed from different perspectives. The Vote Attack [26] is a method that directly perturbs the CapsNets by manipulating the votes from primary capsules. Qin et al. [70] proposed a method to detect adversarial examples using the CapsNet reconstruction network.

These analyses and findings open several directions and strategies for deploying robust CapsNets in safety-critical applications.

6 Security for Spiking Neural Networks

SNNs are considered the third generation of neural networks [49] due to their high biological plausibility and similarities to the human brain. Compared to traditional DNNs, which are based on the computation of continuous values, SNNs process discrete spike trains in an event-based fashion. Hence, they exhibit great potential for deploying high-performance and energy-efficient ML algorithms [56, 90]. In terms of security, the different computational principles of SNNs offer unique vulnerabilities and potential optimizations for improving their robustness. In contrast to the well-established knowledge about DNN security, the robustness of SNNs is an ongoing research topic of high interest in the ML community.

6.1 Comparison DNNs vs. SNNs

The robustness evaluation of SNNs can be conducted by analyzing the comparison between an SNN and a (non-spiking) DNN having the same architectural model, i.e., the same number of layers, neurons per layer, and connections. While the DNN has traditional neurons with ReLU activation function, the SNN has LIF neurons with threshold voltage $V_{th} = 1\,V$. For these experiments, we use a 5-layer network with 3 convolutional layers and 2 fully connected layers on the MNIST dataset [39]. The DNN is trained using the PyTorch framework [66], while the SNN has been implemented and trained with the Norse framework [68]. The PGD attack [50] is applied to both networks using the Foolbox library [73]. Figure 10 shows the accuracy results of both networks when varying the value of adversarial noise budget ε. While for low noise magnitude the DNN has slightly higher accuracy than the SNN, after the turnaround point of $0.5 \leq \varepsilon \leq 0.6$, the opposite behavior is noticed. While the accuracy curve of the DNN decreases sharply, the SNN curve has a lower slope. For instance, when $\varepsilon = 1$, the SNN accuracy is more than 50% higher than the DNN accuracy [16]. Such an outcome indicates that SNNs have the potential to be applied in security contexts due to their higher inherent robustness compared to traditional DNNs. These findings are aligned with recent works [5, 37, 52, 80] that demonstrate the SNNs' higher robustness against security threats, and motivate deeper analyses on this topic.

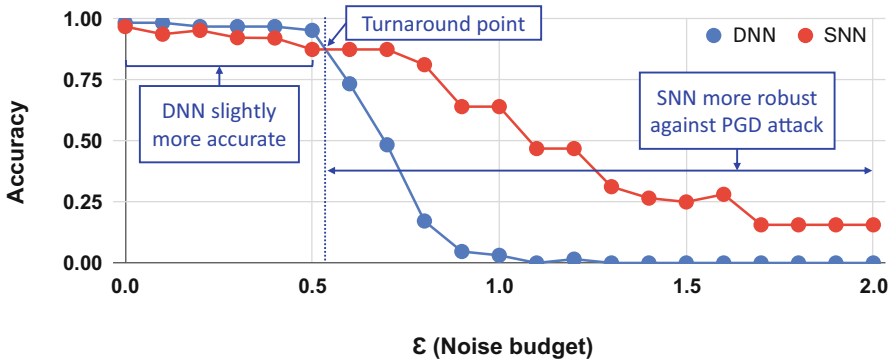


Fig. 10 Comparison between a DNN and an SNN with the same structure under the PGD attack with different values of the adversarial perturbation ε (adapted from [16])

6.2 *Improving the SNN Robustness Through Inherent Structural Parameters*

The previous analyses can be extended not only by exploring the SNN robustness for different adversarial perturbations but also by studying the impact of the SNN structural parameters, i.e., spiking threshold voltage V_{th} and time window T. V_{th} represents the threshold to be compared with the spiking neuron's membrane potential to decide whether or not to emit an output spike. T represents the observation period in which an SNN implemented with the rate-coding mechanism receives spike sequences associated with the same intensity value, which is associated with the firing rate.

6.2.1 SNN Robustness Exploration Methodology

Figure 11 shows the robustness exploration methodology, mainly composed of two steps:

1. *Learnability Analysis:* Given the SNN architecture, the threshold voltage V_i, and the time window T_j, the training in the spiking domain is conducted. This step excludes the configurations of parameters that have low accuracy, by setting a minimum baseline accuracy level below which the SNN learning process is considered inefficient, since there is no interest in continuing the study on SNNs that do not converge.
2. *Security Analysis:* For all the (V_i, T_j) tuples for which the SNN achieves high baseline accuracy, the security study is conducted. The adversarial examples are also generated based on the adversarial noise ε, and the SNN robustness is evaluated. The parameter ε models the strength of the attack, where a high value tends to reduce the SNN accuracy due to the higher perturbation budget given

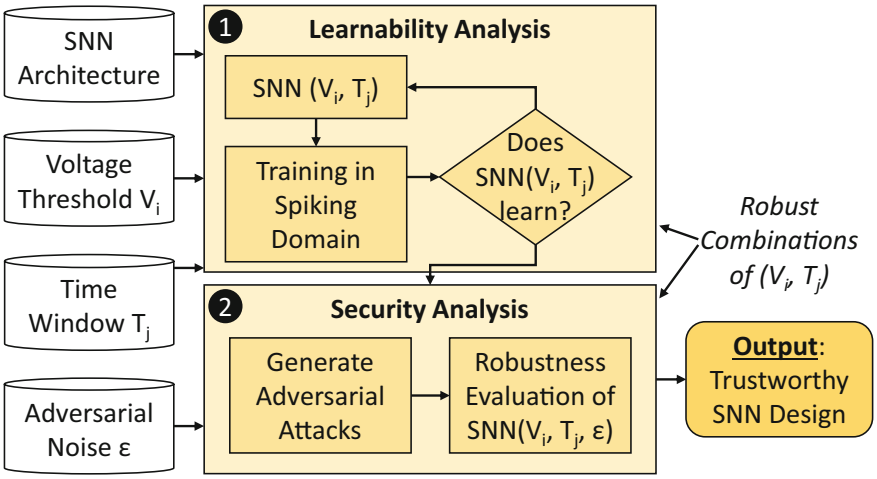


Fig. 11 Methodology for exploring the SNN robustness, varying the threshold voltage V_{th}, the time window T, and the adversarial perturbation ε [16]

to the adversary. For every value of ε, the robustness is computed as the inverse of the attack's success rate, i.e., how many adversarial examples are correctly classified by the SNN.

By observing the robust combinations of (V_i, T_j) during both the learnability and security analyses, a trustworthy SNN design is obtained at the output.

6.2.2 SNN Robustness Evaluation

The experiments were conducted using a 5-layer SNN similar to the LeNet-5 architecture adapted for the spiking domain. It is trained for the MNIST dataset [39] with the Norse framework [68], and the PGD adversarial attacks are implemented using the Foolbox library [73]. Figure 12a shows the heat map relative to the learnability analysis. The variations of V_{th} and T appear on the horizontal and vertical axes, respectively, while the color denotes the SNN accuracy. Compared to the default values, which are $(V_i, T_j) = (1, 64)$, other combinations of parameters are explored and evaluated. While a high-accuracy region can be identified in the top-left corner (low V_i, high T_j), the accuracy is not monotonic w.r.t. both parameters, since the SNN with $(V_i, T_j) = (1.25, 56)$ has lower accuracy than the surrounding points.

Figure 12b show the security analysis heat map for $\varepsilon = 1$. A comparison between the two graphs indicates that high learnability (i.e., without adversarial attacks) does not guarantee high robustness. Indeed, different responses of the SNNs under adversarial attacks based on their respective structural parameters can be noticed. Two SNNs that have a comparable baseline accuracy may have different robustness.

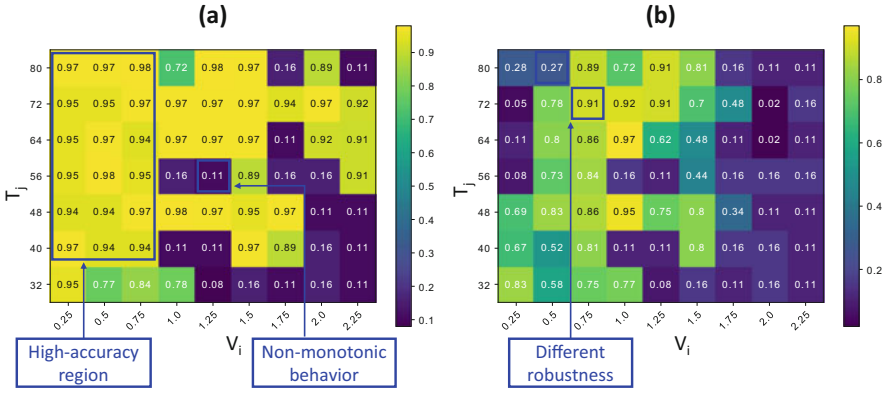


Fig. 12 Heat maps showing the SNN accuracy for the MNIST dataset using different combinations of (V_i, T_j), based on the results in [16]. (**a**) Learnability analysis, equivalent of having $\varepsilon = 0$. (**b**) Security analysis, for $\varepsilon = 1$

For example, the SNN with $(V_i, T_j) = (0.75, 72)$ has 91% accuracy under attack, and the SNN with $(V_i, T_j) = (0.5, 80)$ has only 27% accuracy, while their baseline accuracy is equal to 97% for both combinations.

Hence, studying the SNN security under different values of adversarial perturbations is crucial to identifying robust combinations of threshold voltage and time windows, which contribute to enabling the deployment of SNNs for safety-critical applications.

6.3 Adversarial Attacks and Defenses on Event-Based Data

Along with the efficient implementation of SNNs on neuromorphic architectures (e.g., Intel Loihi [15] and IBM TrueNorth [57]), other advancements in the vision field have come from the event-based camera sensor, such as the dynamic vision sensors (DVS) [41]. Unlike classical frame-based cameras, the DVS cameras emulate the behavior of the human retina by recording the information in the form of spike event sequences, which are generated each time a change of light intensity is detected. As a consequence, SNNs processing event-based data are affected by different types of security vulnerabilities compared to frame-based data processing.

Figure 13 provides an overview of the adversarial threat model used in this section. The frames of events recorded by a DVS camera are subjected to adversarial attacks, while DVS noise filters placed at the input of the neuromorphic hardware that executes SNN inference can mitigate the adversary perturbations.

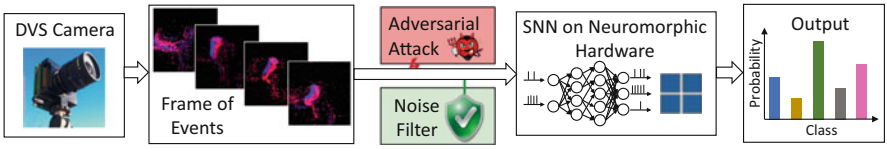


Fig. 13 Adversarial threat model for applying attack algorithms and noise filters on event-based SNNs. Figure adapted from [54]

Algorithm 1: Gradient-based adversarial attack methodology for event-based SNNs [54]

```
1  Define M as a mask able to select only certain frames;
2  Define D as a dataset composed of DVS images;
3  Define P as a perturbation to be added to the images;
4  Define prob as the output probability of a certain class;
5  for d in D do
6      for i in max_iteration do
7          Add P to d only for the frames selected by M;
8          Calculate the prevision on the perturbed input;
9          Extract prob for the correspondent class of d;
10         Update the loss as loss = −log(1 − prob);
11         Calculate the gradients and update P;
```

6.3.1 Gradient-Based Attack for Event Sequences

There exist different types of adversarial attacks and noise filters specific to event-based data. A gradient-based attack [54], described in Algorithm 1, is an iterative algorithm that progressively updates the injected perturbations into the event sequences based on the loss function (lines 7–11 of Algorithm 1) for each frame series of the dataset. After defining a mask in which the perturbation should be added (line 7), the output probability and its respective loss, obtained in the presence of the perturbation, are computed in lines 9 and 10, respectively. Afterward, the perturbation values are updated based on the gradients of the inputs with respect to the loss.

6.3.2 Background Activity Filter for Event Cameras

DVS sensors are mainly affected by noise caused by thermal noise and junction leakage current, which can be classified as a background activity. Since similar events are typically generated in a neighborhood of pixels, the real events have a higher spatio-temporal correlation than the noise events. Such empirical observation is exploited for generating the Background Activity Filter (BAF) [43, 53]. The

Algorithm 2: Background activity filter for event sequences [53]

```
1  Define E as a list of events of the form (x, y, p, t);
2  Define (x_e, y_e, p_e, t_e) as the x coordinate, the y coordinate, the polarity, and the timestamp
     of the event e, respectively;
3  Define M as a N × N matrix, where N is the size of the frames;
4  Define S and T as the spatial and temporal filter's parameters;
5  Initialize M to zero;
6  Sort E from the oldest to the newest event;
7  for e in E do
8      for i in (x_e − S,x_e + S) do
9          for j in (y_e − S, y_e + S) do
10             if not (i == x_e and j == y_e) then
11                 M[i][j] = t_e;
12     if t_e − M[x_e][y_e] > T then
13         Remove e from E;
```

spatio-temporal correlation between events is computed. If such correlation is lower than a certain threshold, the events are filtered out since they are likely due to noise, while the events with higher correlations are kept. The methodology is reported in Algorithm 2, where S and T are the parameters of the filter that set the dimensions of the spatio-temporal neighborhood. Large S and T values imply that few events are filtered out. The filter's decision is based on the comparison between $t_e - M[x_e][y_e]$ and T (lines 12–13 of Algorithm 2). The event is filtered out if the first term is lower.

6.3.3 Evaluation of Gradient-Based Attack and Background Activity Filter

The experiments are conducted by training the 4-layer SNN described in [82], with two convolutional layers and two fully connected layers, for the DvsGesture dataset [4] using the SLAYER backpropagation method [82]. Figure 14 shows the results for the gradient-based attack applied to the SNN. When it is not protected by the BAF, the attack is successful since the SNN accuracy drops to 15.15%. However, the BAF plays the role of a suitable defense since the accuracy remains higher than 90% for a wide range of values for the parameters s and t. At the extremes, for $t = 1$, the accuracy is strongly affected by the parameter s, while for $t = 500$ the SNN accuracy drops to less than 48%.

The results relative to a case study in which the gradient-based attack is applied to the sequence of events of a sample of the DvsGesture dataset are shown in Fig. 15. The first row (Fig. 15a) shows the results for the clean event sequence, i.e., without attack and without filter. The SNN correctly classifies the frame as the class 2, which corresponds to the "left hand wave" label. The second row

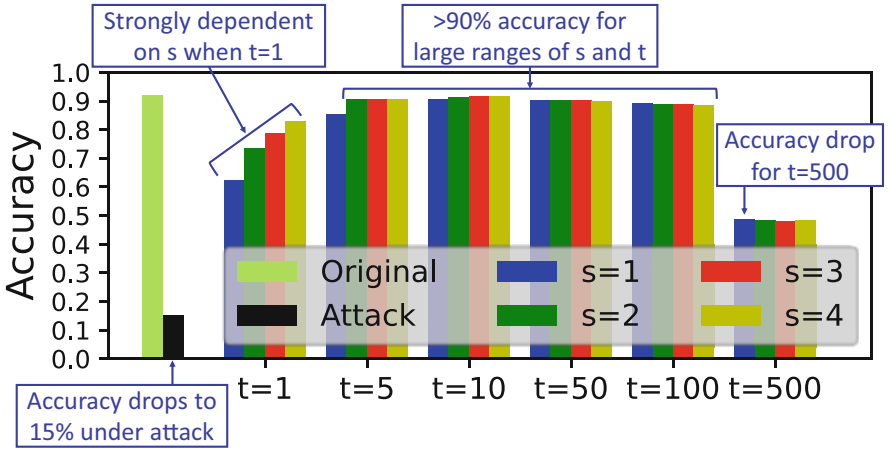


Fig. 14 Robustness evaluation for the SNN on the DvsGesture dataset, under the gradient-based attack and BAF filter. Based on the results in [54]

(Fig. 15b) shows the outcome when the gradient-based adversarial attack is applied. The visible modifications in the event sequences are minimal, but the sample is misclassified by the SNN as the class 0, which corresponds to "hand clap." The last row (Fig. 15c), relative to the scenario in which both the gradient-based attack and the BAF filter (with $s = 2$ and $t = 5$) are present, shows that the sequence is again correctly classified as the class 2 ("left hand wave"). It is worth noticing that several spurious events have been filtered out by the BAF, resulting in high SNN prediction confidence.

6.3.4 Dash Attack for Event Sequences

While the BAF filter is successful against the gradient-based attack, more sophisticated adversarial attack algorithms can evade this protection. For instance, the Dash Attack [53] injects events in the form of a dash. Only two pixels are perturbed for each time step. It starts by targeting the top-left corner (lines 11–13 of Algorithm 3). Afterward, the x and y coordinates are updated to hit only two consecutive pixels (see lines 17–25 of Algorithm 3). Hence, this attack results difficult to spot since the injected spikes do not cause a large overhead on the whole sample.

6.3.5 Mask Filter for Event Cameras

Another type of filtering methodology for event sequences is represented by the Mask Filter (MF) [43, 53]. Algorithm 4 shows the MF technique, whose basic functionality is to filter out the noise on the pixels which have low temporal contrast.

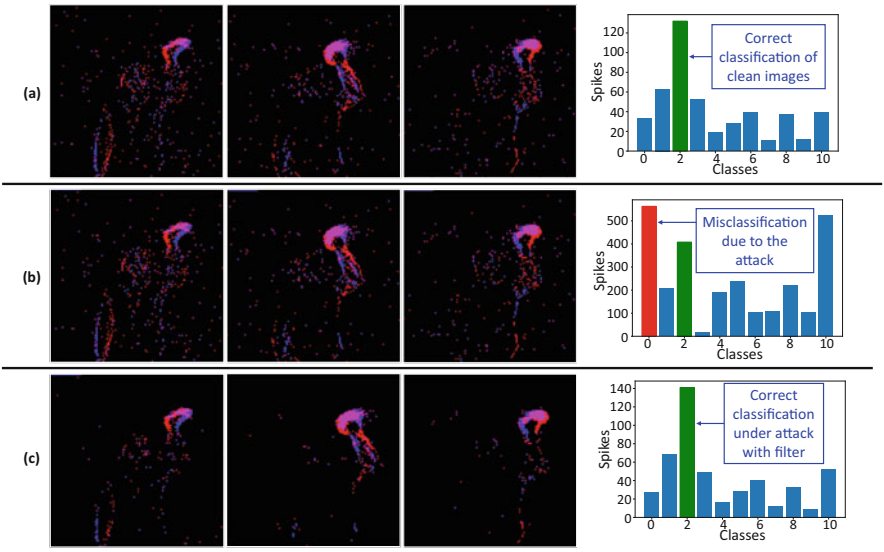


Fig. 15 Detailed results of an event sequence of the DVSGesture dataset labeled as "left hand wave." (**a**) Clean event sequence. (**b**) Event sequence under the gradient-based adversarial attack, unfiltered. (**d**) Event sequence under the gradient-based adversarial attack and protected by the BAF filter with $s = 2$ and $t = 5$. Based on the results in [54]

The activity of each pixel coordinate is monitored (lines 10–11 of Algorithm 4). If such activity exceeds the temporal parameter T, the mask is activated (lines 14–15 of Algorithm 4). After setting all the pixel coordinates of the mask, each event corresponding to a coordinate in which the mask is active is filtered out (lines 15–16 of Algorithm 4).

6.3.6 Evaluation of the Dash Attack Against Background Activity Filter and Mask Filter

The Dash attack introduces perturbations that look very similar to the inherent background noise generated by the DVS camera recording the events. Therefore, they result difficult to be spotted. As shown in Fig. 16a, the accuracy of the SNN without filter under the Dash Attack drops to 0% for the DvsGesture dataset, while the BAF defense produces a slightly higher SNN accuracy. However, the accuracy peak of 28.41% achieved with the BAF with $s = 1$ and $t = 10$ is too low to consider the BAF as a good defense method against the Dash Attack. However, the MF represents a successful defense because the SNN accuracy is high for large values of T.

Algorithm 3: Dash attack methodology [53]

Define D as an event-based dataset made of $(C \times N \times N \times T)$ tensors, where C is the number of channels, N is the size, and T is the duration of the sample;
S is the list of the samples that compose D;
$x_{min} = 0$;
$x = 0$;
$y = 2$;
$left = True$;
while *S is not empty* **do**
  for *s in S* **do**
    for *i in $(0, N-1)$* **do**
      for *j in $(0, N-1)$* **do**
        if $i == x \wedge (left \wedge (j == y \vee j == y-1) \vee \overline{left}$
        $\wedge (j == N-y \vee j == N-y+1))$ **then**
          $s[:, i, j, :] = 1$;
    The perturbed sample s is fed into the SNN, which produces a prediction P;
    if *P is incorrect* **then**
      Remove s from S;
  if $x == x_{min}$ **then**
    $x = N - x_{min} - 1$;
  else
    $left = left \oplus 1$;
    $x = x_{min}$;
    if $\overline{left}$ **then**
      $y = y + 1$;
  if $y > N/2$ **then**
    $x_{min} = x_{min} + 1$;

6.3.7 Mask Filter-Aware Dash Attack for Event Sequences

The main drawback of the Dash attack is its intrinsic weakness against the MF. In fact, it targets the same pixels for the complete sample duration. This highlights which pixels are targeted by the attack. Indeed, the number of events produced by the affected pixels is significantly higher than the events associated with the other pixel coordinates not hit by the attack. In addition, it mainly injects events on the boundaries of the images, which do not tend to overlap with useful information that is typically centered. Hence, by hitting the perimeter of the frames, there is a low risk of superimposing adversarial noise to the main subject. These observations explain the success of the MF in restoring the original SNN accuracy. The perturbed pixels are easily identifiable due to their high number of events, and the filter does not remove useful information, since the modifications are mainly conducted at the edge of the image. Based on these premises, the Mask Filter-Aware Dash Attack

Algorithm 4: Mask filter for event sequences [53]

```
1  Define E as a list of events of the form (x, y, p, t);
2  Define (x_e, y_e, p_e, t_e) as the x coordinate, the y coordinate, the polarity, and the timestamp
     of the event e, respectively;
3  Define M as a N × N matrix, where N is the size of the frames;
4  Define activity as a N × N matrix, representing the number of event produced by each
     pixel;
5  Define T as the temporal threshold passed to the filter as a parameter;
6  Initialize activity to zero;
7  for x in (0, N − 1) do
8      for y in (0, N − 1) do
9          for e in E do
10             if (x, y) == (x_e, y_e) then
11                 activity[x][y]+ = 1;
12         if activity[x][y] > T then
13             M[x][y] = 1;
14 for e in E do
15     if M[x_e][y_e] == 1 then
16         Remove e from E;
```

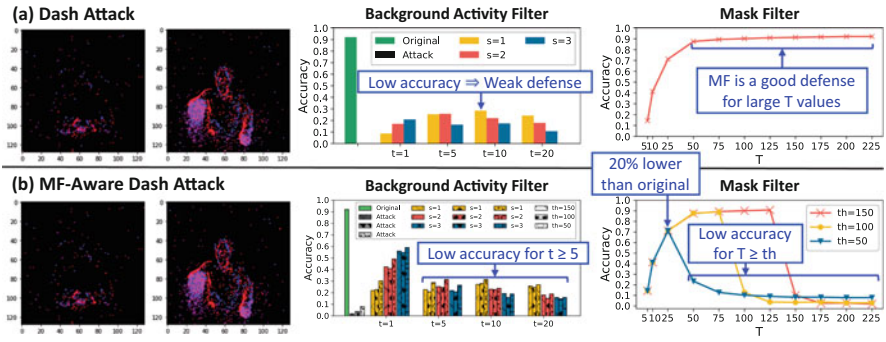


Fig. 16 Evaluation of DVS attacks for the SNN on the DvsGesture dataset, under the BAF and Mask filters, based on the results in [53]. (**a**) Results for the Dash Attack. (**b**) Results for the MF-Aware Dash Attack

has been designed, aiming at being resistant to the MF. It receives as a parameter th that sets a limit on the number of frames that can be changed for each pixel (line 14 of Algorithm 5). Therefore, the algorithm hits a couple of pixels, as in the case of the Dash Attack. However, after injecting events into th frames, it moves to the following pixel coordinates (lines 17–19 of Algorithm 5). The visual effect created by the MF-Aware Dash Attack is that of a dash moving along a line. A smaller th implies a faster movement of the dash across the image.

Algorithm 5: Mask filter-aware dash attack methodology [53]

Define D as an event-based Dataset made of $(2 \times N \times N \times T)$ tensors, where N is the frame dimensions, and T is the sample duration;
Define S as the list of the samples that compose D;
Define th as the parameter associated the activity threshold of the Mask Filter;
Initialize $x = 0$;
Initialize $y_0 = 2$;
Initialize $left = True$;
while *S is not empty* **do**
  for *s in S* **do**
    $th = th_0$;
    $y = y_0$;
    for *t in T* **do**
      for *i in* $(0, N-1)$ **do**
        for *j in* $(0, N-1)$ **do**
          if $i == x \wedge t < th \wedge (left \wedge (j == y \vee j == y-1)$
          $\vee \overline{left} \wedge (j == N-y \vee j == N-y+1))$ **then**
            $s[0, i, j, t] = 1$;
      if $t == th$ **then**
        $th = th + th_0$;
        $y = y + 2$
    The perturbed sample s is fed into the SNN, which produces a prediction P;
    if *P is incorrect* **then**
      Remove s from S;
  if $x == 0$ **then**
    $x = N - 1$;
  else
    $left = left \oplus 1$ $x = 0$ **if** $\overline{left}$ **then**
      $y_0 = y_0 + 1$;

6.3.8 Evaluation of the Mask Filter-Aware Dash Attack Against Background Activity Filter and Mask Filter

Figure 16b shows the results relative to the experiments conducted for the MF-Aware Dash Attack, with different values of the parameter th. While the visibility of the injected noise on the DvsGesture dataset, reported for $th = 150$, is similar to the Dash Attacks, the behavior of the MF-Aware Dash Attack in the presence of noise filters is much different. The accuracy of the SNN under attack without filter is very low (up to 7.95% for $th = 50$. The SNN defended by the BAF shows discrete robustness, in particular, when $s = 3$ and $t = 1$. In such a scenario, the accuracy reaches 59.09% against the MF-Aware Dash Attack with $th = 50$. However, when $t \geq 5$, the SNN accuracy is lower than 31.44%. The key advantage compared to the Dash Attack resides in the behavior of the MF-Aware Dash Attack in the presence of the MF. If $T \geq th$, the SNN accuracy becomes lower than 23.5%. On

the contrary, the behavior for $T < th$ is similar to the results achieved for the Dash Attack. For example, the MF-Aware Dash Attack with $th = 50$ achieves 71.21% accuracy for $T = 25$, which is 20.83% lower than the original SNN accuracy. These results demonstrate that noise event filters such as the BAF and the MF significantly improve the SNN robustness against adversarial attacks. However, an adversarial attack algorithm specifically designed for being resistant to the MF, such as the MF-Aware Dash Attack, has the potential to break the noise filter defense for a good choice of its parameter th.

7 Conclusion

Despite being employed at a large scale, ML models are vulnerable to security threats. Therefore, several defensive mechanisms have been explored to increase their robustness. This chapter presented an overview of ML security, focusing on emerging architectures, such as DNNs, CapsNets, and SNNs. The high complexity of these models requires dedicated methodologies to investigate their trustworthiness. The analyses conducted in this chapter demonstrated that CapsNets are more robust than traditional DNNs against affine transformations and adversarial attacks. SNNs are inherently more robust than non-spiking DNNs, and such inherent robustness can be enhanced by fine-tuning their structural parameters, like the spiking voltage threshold and the time window. Moreover, event-based SNNs can be protected through noise filters for event sensors, like the Background Activity Filter and the Mask Filter. However, when properly tuned, advanced event-based adversarial attack methodologies, such as the Mask Filter-Aware Dash Attack, can cause significant accuracy drops in SNNs.

Acknowledgments This work has been supported in part by Intel Corporation through Gift funding for the project "Cost-Effective Dependability for Deep Neural Networks and Spiking Neural Networks," and in part by the Doctoral College Resilient Embedded Systems, which is run jointly by the TU Wien's Faculty of Informatics and the UAS Technikum Wien. This work was also supported in parts by the NYUAD Center for Interacting Urban Networks (CITIES), funded by Tamkeen under the NYUAD Research Institute Award CG001, Center for CyberSecurity (CCS), funded by Tamkeen under the NYUAD Research Institute Award G1104, and Center for Artificial Intelligence and Robotics (CAIR), funded by Tamkeen under the NYUAD Research Institute Award CG010.

References

1. Abadi, M., Chu, A., Goodfellow, I.J., McMahan, H.B., Mironov, I., Talwar, K., Zhang, L.: Deep learning with differential privacy. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, October 24–28, 2016, pp. 308–318. ACM, New York (2016). https://doi.org/10.1145/2976749.2978318

2. Agoyan, M., Dutertre, J., Mirbaha, A., Naccache, D., Ribotta, A., Tria, A.: How to flip a bit? In: 16th IEEE International On-line Testing Symposium (IOLTS 2010, 5–7 July, 2010, Corfu, pp. 235–239. IEEE Computer Society, Washington (2010). https://doi.org/10.1109/IOLTS.2010.5560194
3. Ali, H., Khalid, F., Tariq, H., Hanif, M.A., Ahmed, R., Rehman, S.: SSCNets: robustifying DNNs using secure selective convolutional filters. IEEE Des. Test **37**(2), 58–65 (2020). https://doi.org/10.1109/MDAT.2019.2961325
4. Amir, A., Taba, B., Berg, D.J., Melano, T., McKinstry, J.L., di Nolfo, C., Nayak, T.K., Andreopoulos, A., Garreau, G., Mendoza, M., Kusnitz, J., DeBole, M., Esser, S.K., Delbrück, T., Flickner, M., Modha, D.S.: A low power, fully event-based gesture recognition system. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, July 21–26, 2017, pp. 7388–7397. IEEE Computer Society, Washington (2017). https://doi.org/10.1109/CVPR.2017.781
5. Bagheri, A., Simeone, O., Rajendran, B.: Adversarial training for probabilistic spiking neural networks. In: 19th IEEE International Workshop on Signal Processing Advances in Wireless Communications, SPAWC 2018, Kalamata, June 25–28, 2018, pp. 1–5. IEEE, Piscataway (2018). https://doi.org/10.1109/SPAWC.2018.8446003
6. Baumann, R.: Radiation-induced soft errors in advanced semiconductor technologies. IEEE Trans. Device Mater. Reliab. **5**(3), 305–316 (2005). https://doi.org/10.1109/TDMR.2005.853449
7. Breier, J., Hou, X., Jap, D., Ma, L., Bhasin, S., Liu, Y.: Practical fault attack on deep neural networks. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, October 15–19, 2018, pp. 2204–2206. ACM, New York (2018). https://doi.org/10.1145/3243734.3278519
8. Brendel, W., Rauber, J., Bethge, M.: Decision-based adversarial attacks: reliable attacks against black-box machine learning models. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, April 30–May 3, 2018. Conference Track Proceedings. OpenReview.net (2018). https://openreview.net/forum?id=SyZI0GWCZ
9. Capra, M., Bussolino, B., Marchisio, A., Masera, G., Martina, M., Shafique, M.: Hardware and software optimizations for accelerating deep neural networks: survey of current trends, challenges, and the road ahead. IEEE Access **8**, 225134–225180 (2020). https://doi.org/10.1109/ACCESS.2020.3039858
10. Capra, M., Bussolino, B., Marchisio, A., Shafique, M., Masera, G., Martina, M.: An updated survey of efficient hardware architectures for accelerating deep convolutional neural networks. Fut. Int. **12**(7), 113 (2020). https://doi.org/10.3390/fi12070113
11. Chen, Z., Li, G., Pattabiraman, K.: Ranger: boosting error resilience of deep neural networks through range restriction. CoRR abs/2003.13874 (2020). https://arxiv.org/abs/2003.13874
12. Clements, J., Lao, Y.: Hardware trojan design on neural networks. In: IEEE International Symposium on Circuits and Systems, ISCAS 2019, Sapporo, May 26–29, 2019, pp. 1–5. IEEE, Piscataway (2019). https://doi.org/10.1109/ISCAS.2019.8702493
13. Cohen, J.M., Rosenfeld, E., Kolter, J.Z.: Certified adversarial robustness via randomized smoothing. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9–15 June 2019, Long Beach. Proceedings of Machine Learning Research, vol. 97, pp. 1310–1320. PMLR (2019). http://proceedings.mlr.press/v97/cohen19c.html
14. Dave, S., Marchisio, A., Hanif, M.A., Guesmi, A., Shrivastava, A., Alouani, I., Shafique, M.: Special session: towards an agile design methodology for efficient, reliable, and secure ML systems. In: 40th IEEE VLSI Test Symposium, VTS 2022, San Diego, April 25–27, 2022, pp. 1–14. IEEE, Piscataway (2022). https://doi.org/10.1109/VTS52500.2021.9794253
15. Davies, M., Srinivasa, N., Lin, T., Chinya, G.N., Cao, Y., Choday, S.H., Dimou, G.D., Joshi, P., Imam, N., Jain, S., Liao, Y., Lin, C., Lines, A., Liu, R., Mathaikutty, D., McCoy, S., Paul, A., Tse, J., Venkataramanan, G., Weng, Y., Wild, A., Yang, Y., Wang, H.: Loihi: A neuromorphic manycore processor with on-chip learning. IEEE Micro **38**(1), 82–99 (2018). https://doi.org/10.1109/MM.2018.112130359

16. El-Allami, R., Marchisio, A., Shafique, M., Alouani, I.: Securing deep spiking neural networks against adversarial attacks through inherent structural parameters. In: Design, Automation & Test in Europe Conference & Exhibition, DATE 2021, Grenoble, February 1–5, 2021, pp. 774–779. IEEE, Piscataway (2021). https://doi.org/10.23919/DATE51398.2021.9473981
17. Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T., Song, D.: Robust physical-world attacks on deep learning visual classification. In: 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, June 18–22, 2018, pp. 1625–1634. Computer Vision Foundation/IEEE Computer Society, Washington (2018). https://doi.org/10.1109/CVPR.2018.00175, http://openaccess.thecvf.com/content_cvpr_2018/html/Eykholt_Robust_Physical-World_Attacks_CVPR_2018_paper.html
18. Fani, R., Zamani, M.S.: Runtime hardware trojan detection by reconfigurable monitoring circuits. J. Supercomput. (2022). https://doi.org/10.1007/s11227-022-04362-1
19. Ganju, K., Wang, Q., Yang, W., Gunter, C.A., Borisov, N.: Property inference attacks on fully connected neural networks using permutation invariant representations. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, October 15–19, 2018, pp. 619–633. ACM, New York (2018). https://doi.org/10.1145/3243734.3243834
20. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, May 31–June 2, 2009, pp. 169–178. ACM, New York (2009). https://doi.org/10.1145/1536414.1536440
21. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K.E., Naehrig, M., Wernsing, J.: Cryptonets: applying neural networks to encrypted data with high throughput and accuracy. In: Balcan, M., Weinberger, K.Q. (eds.) Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, June 19–24, 2016, JMLR Workshop and Conference Proceedings, vol. 48, pp. 201–210. JMLR.org (2016). http://proceedings.mlr.press/v48/gilad-bachrach16.html
22. Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A.C., Bengio, Y.: Generative adversarial networks. CoRR abs/1406.2661 (2014). http://arxiv.org/abs/1406.2661
23. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, May 7–9, 2015. Conference Track Proceedings (2015). http://arxiv.org/abs/1412.6572
24. Gu, T., Liu, K., Dolan-Gavitt, B., Garg, S.: BadNets: evaluating backdooring attacks on deep neural networks. IEEE Access **7**, 47230–47244 (2019). https://doi.org/10.1109/ACCESS.2019.2909068
25. Gu, J., Tresp, V.: Improving the robustness of capsule networks to image affine transformations. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, June 13–19, 2020, pp. 7283–7291. Computer Vision Foundation/IEEE, Piscataway (2020). https://doi.org/10.1109/CVPR42600.2020.00731, https://openaccess.thecvf.com/content_CVPR_2020/html/Gu_Improving_the_Robustness_of_Capsule_Networks_to_Image_Affine_Transformations_CVPR_2020_paper.html
26. Gu, J., Wu, B., Tresp, V.: Effective and efficient vote attack on capsule networks. In: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, May 3–7, 2021. OpenReview.net (2021). https://openreview.net/forum?id=33rtZ4Sjwjn
27. Guesmi, A., Alouani, I., Khasawneh, K.N., Baklouti, M., Frikha, T., Abid, M., Abu-Ghazaleh, N.B.: Defensive approximation: securing CNNs using approximate computing. In: Sherwood, T., Berger, E.D., Kozyrakis, C. (eds.) ASPLOS '21: 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Virtual Event, April 19–23, 2021, pp. 990–1003. ACM, New York (2021). https://doi.org/10.1145/3445814.3446747
28. Hanif, M.A., Shafique, M.: Salvagednn: salvaging deep neural network accelerators with permanent faults through saliency-driven fault-aware mapping. Phil. Trans. R. Soc. A. **378**(2164) (2020). https://doi.org/10.1098/rsta.2019.0164

29. Hanif, M.A., Shafique, M.: DNN-life: an energy-efficient aging mitigation framework for improving the lifetime of on-chip weight memories in deep neural network hardware architectures. In: Design, Automation & Test in Europe Conference & Exhibition, DATE 2021, Grenoble, February 1–5, 2021, pp. 729–734. IEEE, Piscataway (2021). https://doi.org/10.23919/DATE51398.2021.9473943
30. Hoang, L.H., Hanif, M.A., Shafique, M.: FT-ClipAct: resilience analysis of deep neural networks and improving their fault tolerance using clipped activation. In: 2020 Design, Automation & Test in Europe Conference & Exhibition, DATE 2020, Grenoble, March 9–13, 2020, pp. 1241–1246. IEEE, Piscataway (2020). https://doi.org/10.23919/DATE48585.2020.9116571
31. Juvekar, C., Vaikuntanathan, V., Chandrakasan, A.P.: GAZELLE: a low latency framework for secure neural network inference. In: Enck, W., Felt, A.P. (eds.) 27th USENIX Security Symposium, USENIX Security 2018, Baltimore, August 15–17, 2018, pp. 1651–1669. USENIX Association, Berkeley (2018). https://www.usenix.org/conference/usenixsecurity18/presentation/juvekar
32. Kang, K., Gangwal, S., Park, S.P., Roy, K.: NBTI induced performance degradation in logic and memory circuits: how effectively can we approach a reliability solution? In: Kyung, C., Choi, K., Ha, S. (eds.) Proceedings of the 13th Asia South Pacific Design Automation Conference, ASP-DAC 2008, Seoul, January 21–24, 2008, pp. 726–731. IEEE, Piscataway (2008). https://doi.org/10.1109/ASPDAC.2008.4484047
33. Khalid, F., Ali, H., Tariq, H., Hanif, M.A., Rehman, S., Ahmed, R., Shafique, M.: QuSecNets: quantization-based defense mechanism for securing deep neural network against adversarial attacks. In: Gizopoulos, D., Alexandrescu, D., Papavramidou, P., Maniatakos, M. (eds.) 25th IEEE International Symposium on On-Line Testing and Robust System Design, IOLTS 2019, Rhodes, July 1–3, 2019, pp. 182–187. IEEE, Piscataway (2019). https://doi.org/10.1109/IOLTS.2019.8854377
34. Khalid, F., Ali, H., Hanif, M.A., Rehman, S., Ahmed, R., Shafique, M.: FaDec: a fast decision-based attack for adversarial machine learning. In: 2020 International Joint Conference on Neural Networks, IJCNN 2020, Glasgow, July 19–24, 2020, pp. 1–8. IEEE, Piscataway (2020). https://doi.org/10.1109/IJCNN48605.2020.9207635
35. Kim, Y., Daly, R., Kim, J.S., Fallin, C., Lee, J., Lee, D., Wilkerson, C., Lai, K., Mutlu, O.: Flipping bits in memory without accessing them: an experimental study of DRAM disturbance errors. In: ACM/IEEE 41st International Symposium on Computer Architecture, ISCA 2014, Minneapolis, June 14–18, 2014, pp. 361–372. IEEE Computer Society, Washington (2014). https://doi.org/10.1109/ISCA.2014.6853210
36. Kumar, A.D.: Novel deep learning model for traffic sign detection using capsule networks. CoRR abs/1805.04424 (2018). http://arxiv.org/abs/1805.04424
37. Kundu, S., Pedram, M., Beerel, P.A.: HIRE-SNN: harnessing the inherent robustness of energy-efficient deep spiking neural networks by training with crafted input noise. In: 2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, October 10–17, 2021, pp. 5189–5198. IEEE, Piscataway (2021). https://doi.org/10.1109/ICCV48922.2021.00516
38. Kurakin, A., Goodfellow, I.J., Bengio, S.: Adversarial examples in the physical world. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, April 24–26, 2017, Workshop Track Proceedings. OpenReview.net (2017). https://openreview.net/forum?id=HJGU3Rodl
39. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998). https://doi.org/10.1109/5.726791
40. Li, J., Rakin, A.S., Xiong, Y., Chang, L., He, Z., Fan, D., Chakrabarti, C.: Defending bit-flip attack through DNN weight reconstruction. In: 57th ACM/IEEE Design Automation Conference, DAC 2020, San Francisco, July 20–24, 2020, pp. 1–6. IEEE, Piscataway (2020). https://doi.org/10.1109/DAC18072.2020.9218665
41. Lichtsteiner, P., Posch, C., Delbrück, T.: A 128×128 120 dB 15 μs latency asynchronous temporal contrast vision sensor. IEEE J. Solid State Circuits **43**(2), 566–576 (2008). https://doi.org/10.1109/JSSC.2007.914337

42. Lin, J., Gan, C., Han, S.: Defensive quantization: When efficiency meets robustness. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, May 6–9, 2019. OpenReview.net (2019). https://openreview.net/forum?id=ryetZ20ctX
43. Linares-Barranco, A., Perez-Peña, F., Moeys, D.P., Gomez-Rodriguez, F., Jiménez-Moreno, G., Liu, S., Delbrück, T.: Low latency event-based filtering and feature extraction for dynamic vision sensors in real-time FPGA applications. IEEE Access **7**, 134926–134942 (2019). https://doi.org/10.1109/ACCESS.2019.2941282
44. Liu, J., Juuti, M., Lu, Y., Asokan, N.: Oblivious neural network predictions via MiniONN transformations. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, October 30–November 03, 2017, pp. 619–631. ACM, New York (2017). https://doi.org/10.1145/3133956.3134056
45. Liu, Y., Wei, L., Luo, B., Xu, Q.: Fault injection attack on deep neural network. In: Parameswaran, S. (ed.) 2017 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2017, Irvine, November 13–16, 2017, pp. 131–138. IEEE, Piscataway (2017). https://doi.org/10.1109/ICCAD.2017.8203770
46. Liu, K., Dolan-Gavitt, B., Garg, S.: Fine-pruning: defending against backdooring attacks on deep neural networks. In: Bailey, M., Holz, T., Stamatogiannakis, M., Ioannidis, S. (eds.) Research in Attacks, Intrusions, and Defenses – 21st International Symposium, RAID 2018, Heraklion, Crete, September 10–12, 2018, Proceedings, Lecture Notes in Computer Science, vol. 11050, pp. 273–294. Springer, Berlin (2018). https://doi.org/10.1007/978-3-030-00470-5_13
47. Liu, X., Deng, R.H., Wu, P., Yang, Y.: Lightning-fast and privacy-preserving outsourced computation in the cloud. Cybersecur **3**(1), 17 (2020). https://doi.org/10.1186/s42400-020-00057-3
48. Lyons, R.E., Vanderkulk, W.: The use of triple-modular redundancy to improve computer reliability. IBM J. Res. Dev. **6**(2), 200–209 (1962). https://doi.org/10.1147/rd.62.0200
49. Maass, W.: Networks of spiking neurons: the third generation of neural network models. Neural Netw. **10**(9), 1659–1671 (1997). https://doi.org/10.1016/S0893-6080(97)00011-7
50. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, April 30–May 3, 2018, Conference Track Proceedings. OpenReview.net (2018). https://openreview.net/forum?id=rJzIBfZAb
51. Marchisio, A., Nanfa, G., Khalid, F., Hanif, M.A., Martina, M., Shafique, M.: CapsAttacks: robust and imperceptible adversarial attacks on capsule networks. CoRR abs/1901.09878 (2019). http://arxiv.org/abs/1901.09878
52. Marchisio, A., Nanfa, G., Khalid, F., Hanif, M.A., Martina, M., Shafique, M.: Is spiking secure? A comparative study on the security vulnerabilities of spiking and deep neural networks. In: 2020 International Joint Conference on Neural Networks, IJCNN 2020, Glasgow, July 19–24, 2020, pp. 1–8. IEEE, Piscataway (2020). https://doi.org/10.1109/IJCNN48605.2020.9207297
53. Marchisio, A., Pira, G., Martina, M., Masera, G., Shafique, M.: DVS-attacks: adversarial attacks on dynamic vision sensors for spiking neural networks. In: International Joint Conference on Neural Networks, IJCNN 2021, Shenzhen, July 18–22, 2021, pp. 1–9. IEEE, Piscataway (2021). https://doi.org/10.1109/IJCNN52387.2021.9534364
54. Marchisio, A., Pira, G., Martina, M., Masera, G., Shafique, M.: R-SNN: an analysis and design methodology for robustifying spiking neural networks against adversarial attacks through noise filters for dynamic vision sensors. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2021, Prague, September 27–Oct. 1, 2021, pp. 6315–6321. IEEE, Piscataway (2021). https://doi.org/10.1109/IROS51168.2021.9636718
55. Marchisio, A., Caramia, G., Martina, M., Shafique, M.: fakeWeather: adversarial attacks for deep neural networks emulating weather conditions on the camera lens of autonomous systems. In: 2022 International Joint Conference on Neural Networks, IJCNN 2022, Padua, July 18–23, 2022. IEEE, Piscataway (2022)

56. Massa, R., Marchisio, A., Martina, M., Shafique, M.: An efficient spiking neural network for recognizing gestures with a DVS camera on the Loihi neuromorphic processor. In: 2020 International Joint Conference on Neural Networks, IJCNN 2020, Glasgow, July 19–24, 2020, pp. 1–9. IEEE, Piscataway (2020). https://doi.org/10.1109/IJCNN48605.2020.9207109
57. Merolla, P.A., Arthur, J.V., Alvarez-Icaza, R., Cassidy, A.S., Sawada, J., Akopyan, F., Jackson, B.L., Imam, N., Guo, C., Nakamura, Y., Brezzo, B., Vo, I., Esser, S.K., Appuswamy, R., Taba, B., Amir, A., Flickner, M.D., Risk, W.P., Manohar, R., Modha, D.S.: A million spiking-neuron integrated circuit with a scalable communication network and interface. Science **345**(6197), 668–673 (2014). https://doi.org/10.1126/science.1254642, https://www.science.org/doi/abs/10.1126/science.1254642
58. Michels, F., Uelwer, T., Upschulte, E., Harmeling, S.: On the vulnerability of capsule networks to adversarial attacks. CoRR abs/1906.03612 (2019). http://arxiv.org/abs/1906.03612
59. Mohassel, P., Zhang, Y.: SecureML: a system for scalable privacy-preserving machine learning. In: 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, May 22–26, 2017, pp. 19–38. IEEE Computer Society, Washington (2017). https://doi.org/10.1109/SP.2017.12
60. Nandakumar, K., Ratha, N.K., Pankanti, S., Halevi, S.: Towards deep neural network training on encrypted data. In: IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2019, Long Beach, June 16–20, 2019, pp. 40–48. Computer Vision Foundation/IEEE, Piscataway (2019). https://doi.org/10.1109/CVPRW.2019.00011, http://openaccess.thecvf.com/content_CVPRW_2019/html/CV-COPS/Nandakumar_Towards_Deep_Neural_Network_Training_on_Encrypted_Data_CVPRW_2019_paper.html
61. Ozen, E., Orailoglu, A.: Sanity-check: Boosting the reliability of safety-critical deep neural network applications. In: 28th IEEE Asian Test Symposium, ATS 2019, Kolkata, December 10–13, 2019, pp. 7–12. IEEE, Piscataway (2019). https://doi.org/10.1109/ATS47505.2019.000-8
62. Ozen, E., Orailoglu, A.: Boosting bit-error resilience of DNN accelerators through median feature selection. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **39**(11), 3250–3262 (2020). https://doi.org/10.1109/TCAD.2020.3012209
63. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) Advances in Cryptology – EUROCRYPT '99, Proceeding of the International Conference on the Theory and Application of Cryptographic Techniques, Prague, May 2–6, 1999. Lecture Notes in Computer Science, vol. 1592, pp. 223–238. Springer, Berlin (1999). https://doi.org/10.1007/3-540-48910-X_16
64. Pandey, P., Basu, P., Chakraborty, K., Roy, S.: GreenTPU: predictive design paradigm for improving timing error resilience of a near-threshold tensor processing unit. IEEE Trans. Very Large Scale Integr. Syst. **28**(7), 1557–1566 (2020). https://doi.org/10.1109/TVLSI.2020.2985057
65. Papernot, N., Song, S., Mironov, I., Raghunathan, A., Talwar, K., Erlingsson, Ú.: Scalable private learning with PATE. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, April 30–May 3, 2018. Conference Track Proceedings. OpenReview.net (2018). https://openreview.net/forum?id=rkZB1XbRZ
66. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E.Z., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: PyTorch: An imperative style, high-performance deep learning library. In: Wallach, H.M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E.B., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8–14, 2019, Vancouver, pp. 8024–8035 (2019). https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html
67. Paudice, A., Muñoz-González, L., György, A., Lupu, E.C.: Detection of adversarial training examples in poisoning attacks through anomaly detection. CoRR abs/1802.03041 (2018). http://arxiv.org/abs/1802.03041
68. Pehle, C., Pedersen, J.E.: Norse—a deep learning library for spiking neural networks (2021). https://doi.org/10.5281/zenodo.4422025. Documentation: https://norse.ai/docs/

69. Prasanth, V., Singh, V., Parekhji, R.A.: Reduced overhead soft error mitigation using error control coding techniques. In: 17th IEEE International On-line Testing Symposium (IOLTS 2011), 13–15 July, 2011, Athens, pp. 163–168. IEEE Computer Society, Washington (2011). https://doi.org/10.1109/IOLTS.2011.5993831
70. Qin, Y., Frosst, N., Sabour, S., Raffel, C., Cottrell, G.W., Hinton, G.E.: Detecting and diagnosing adversarial images with class-conditional capsule reconstructions. In: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, April 26–30, 2020. OpenReview.net (2020). https://openreview.net/forum?id=Skgy464Kvr
71. Raghunathan, B., Turakhia, Y., Garg, S., Marculescu, D.: Cherry-picking: exploiting process variations in dark-silicon homogeneous chip multi-processors. In: Macii, E. (ed.) Design, Automation and Test in Europe, DATE 13, Grenoble, March 18–22, 2013, pp. 39–44. EDA Consortium San Jose/ACM DL, New York (2013). https://doi.org/10.7873/DATE.2013.023
72. Rakin, A.S., He, Z., Fan, D.: Bit-flip attack: Crushing neural network with progressive bit search. In: 2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, October 27–November 2, 2019, pp. 1211–1220. IEEE, Piscataway (2019). https://doi.org/10.1109/ICCV.2019.00130
73. Rauber, J., Brendel, W., Bethge, M.: Foolbox v0.8.0: a python toolbox to benchmark the robustness of machine learning models. CoRR abs/1707.04131 (2017). http://arxiv.org/abs/1707.04131
74. Rouhani, B.D., Riazi, M.S., Koushanfar, F.: DeepSecure: scalable provably-secure deep learning. In: Proceedings of the 55th Annual Design Automation Conference, DAC 2018, San Francisco, June 24–29, 2018, pp. 2:1–2:6. ACM, New York (2018). https://doi.org/10.1145/3195970.3196023
75. Sabour, S., Frosst, N., Hinton, G.E.: Dynamic routing between capsules. In: Guyon, I., von Luxburg, U., Bengio, S., Wallach, H.M., Fergus, R., Vishwanathan, S.V.N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4–9, 2017, Long Beach, pp. 3856–3866 (2017). https://proceedings.neurips.cc/paper/2017/hash/2cad8fa47bbef282badbb8de5374b894-Abstract.html
76. Shafahi, A., Huang, W.R., Najibi, M., Suciu, O., Studer, C., Dumitras, T., Goldstein, T.: Poison frogs! targeted clean-label poisoning attacks on neural networks. In: Bengio, S., Wallach, H.M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3–8, 2018, Montréal, pp. 6106–6116 (2018). https://proceedings.neurips.cc/paper/2018/hash/22722a343513ed45f14905eb07621686-Abstract.html
77. Shafique, M., Naseer, M., Theocharides, T., Kyrkou, C., Mutlu, O., Orosa, L., Choi, J.: Robust machine learning systems: challenges, current trends, perspectives, and the road ahead. IEEE Des. Test **37**(2), 30–57 (2020). https://doi.org/10.1109/MDAT.2020.2971217
78. Shafique, M., Marchisio, A., Putra, R.V.W., Hanif, M.A.: Towards energy-efficient and secure edge AI: a cross-layer framework ICCAD special session paper. In: IEEE/ACM International Conference On Computer Aided Design, ICCAD 2021, Munich, November 1–4, 2021, pp. 1–9. IEEE, Piscataway (2021). https://doi.org/10.1109/ICCAD51958.2021.9643539
79. Sharif, M., Bhagavatula, S., Bauer, L., Reiter, M.K.: Accessorize to a crime: real and stealthy attacks on state-of-the-art face recognition. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, October 24–28, 2016, pp. 1528–1540. ACM, New York (2016). https://doi.org/10.1145/2976749.2978392
80. Sharmin, S., Rathi, N., Panda, P., Roy, K.: Inherent adversarial robustness of deep spiking neural networks: Effects of discrete input encoding and non-linear activations. In: Vedaldi, A., Bischof, H., Brox, T., Frahm, J. (eds.) Proceedings of the Computer Vision – ECCV 2020 – 16th European Conference, Glasgow, August 23–28, 2020, Part XXIX. Lecture Notes in Computer Science, vol. 12374, pp. 399–414. Springer, Berlin (2020). https://doi.org/10.1007/978-3-030-58526-6_24

81. Shokri, R., Stronati, M., Song, C., Shmatikov, V.: Membership inference attacks against machine learning models. In: 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, May 22–26, 2017, pp. 3–18. IEEE Computer Society, Washington (2017). https://doi.org/10.1109/SP.2017.41
82. Shrestha, S.B., Orchard, G.: SLAYER: spike layer error reassignment in time. In: Bengio, S., Wallach, H.M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3–8, 2018, Montréal, pp. 1419–1428 (2018). https://proceedings.neurips.cc/paper/2018/hash/82f2b308c3b01637c607ce05f52a2fed-Abstract.html
83. Siddique, A., Hoque, K.A.: Is approximation universally defensive against adversarial attacks in deep neural networks? CoRR abs/2112.01555 (2021). https://arxiv.org/abs/2112.01555
84. Stallkamp, J., Schlipsing, M., Salmen, J., Igel, C.: The German traffic sign recognition benchmark: a multi-class classification competition. In: The 2011 International Joint Conference on Neural Networks, IJCNN 2011, San Jose, July 31–August 5, 2011, pp. 1453–1460. IEEE, Piscataway (2011). https://doi.org/10.1109/IJCNN.2011.6033395
85. Su, J., Vargas, D.V., Sakurai, K.: One pixel attack for fooling deep neural networks. IEEE Trans. Evol. Comput. **23**(5), 828–841 (2019). https://doi.org/10.1109/TEVC.2019.2890858
86. Thys, S., Ranst, W.V., Goedemé, T.: Fooling automated surveillance cameras: adversarial patches to attack person detection. In: IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2019, Long Beach, June 16–20, 2019, pp. 49–55. Computer Vision Foundation/IEEE, Piscataway (2019). https://doi.org/10.1109/CVPRW.2019.00012, http://openaccess.thecvf.com/content_CVPRW_2019/html/CV-COPS/Thys_Fooling_Automated_Surveillance_Cameras_Adversarial_Patches_to_Attack_Person_Detection_CVPRW_2019_paper.html
87. Tramèr, F., Zhang, F., Juels, A., Reiter, M.K., Ristenpart, T.: Stealing machine learning models via prediction APIs. In: Holz, T., Savage, S. (eds.) 25th USENIX Security Symposium, USENIX Security 16, Austin, August 10–12, 2016, pp. 601–618. USENIX Association, Berkeley (2016). https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/tramer
88. Vadlamani, R., Zhao, J., Burleson, W.P., Tessier, R.: Multicore soft error rate stabilization using adaptive dual modular redundancy. In: Micheli, G.D., Al-Hashimi, B.M., Müller, W., Macii, E. (eds.) Design, Automation and Test in Europe, DATE 2010, Dresden, March 8–12, 2010, pp. 27–32. IEEE Computer Society, Washington (2010). https://doi.org/10.1109/DATE.2010.5457242
89. Venceslai, V., Marchisio, A., Alouani, I., Martina, M., Shafique, M.: NeuroAttack: undermining spiking neural networks security through externally triggered bit-flips. In: 2020 International Joint Conference on Neural Networks, IJCNN 2020, Glasgow, July 19–24, 2020, pp. 1–8. IEEE, Piscataway (2020). https://doi.org/10.1109/IJCNN48605.2020.9207351
90. Viale, A., Marchisio, A., Martina, M., Masera, G., Shafique, M.: Carsnn: An efficient spiking neural network for event-based autonomous cars on the Loihi neuromorphic research processor. In: International Joint Conference on Neural Networks, IJCNN 2021, Shenzhen, July 18–22, 2021, pp. 1–10. IEEE, Piscataway (2021). https://doi.org/10.1109/IJCNN52387.2021.9533738
91. Wagh, S., Gupta, D., Chandran, N.: SecureNN: 3-party secure computation for neural network training. Proc. Priv. Enhancing Technol. **2019**(3), 26–49 (2019). https://doi.org/10.2478/popets-2019-0035
92. Wang, B., Gong, N.Z.: Stealing hyperparameters in machine learning. In: Proceedings of the 2018 IEEE Symposium on Security and Privacy, SP 2018, 21–23 May 2018, San Francisco, pp. 36–52. IEEE Computer Society, Washington (2018). https://doi.org/10.1109/SP.2018.00038
93. Wang, L., Guo, S., Huang, W., Qiao, Y.: Places205-vggnet models for scene recognition. CoRR abs/1508.01667 (2015). http://arxiv.org/abs/1508.01667
94. Zhang, J., Rangineni, K., Ghodsi, Z., Garg, S.: ThunderVolt: enabling aggressive voltage underscaling and timing error resilience for energy efficient deep learning accelerators. In: Proceedings of the 55th Annual Design Automation Conference, DAC 2018, San Francisco,

June 24–29, 2018, pp. 19:1–19:6. ACM, New York (2018). https://doi.org/10.1145/3195970.3196129

95. Zhang, J.J., Gu, T., Basu, K., Garg, S.: Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator. In: 36th IEEE VLSI Test Symposium, VTS 2018, San Francisco, April 22–25, 2018, pp. 1–6. IEEE Computer Society, Washington (2018). https://doi.org/10.1109/VTS.2018.8368656
96. Zhang, J.J., Liu, K., Khalid, F., Hanif, M.A., Rehman, S., Theocharides, T., Artussi, A., Shafique, M., Garg, S.: Building robust machine learning systems: current progress, research challenges, and opportunities. In: Proceedings of the 56th Annual Design Automation Conference 2019, DAC 2019, Las Vegas, June 02–06, 2019, p. 175. ACM, New York (2019). https://doi.org/10.1145/3316781.3323472
97. Zhao, K., Di, S., Li, S., Liang, X., Zhai, Y., Chen, J., Ouyang, K., Cappello, F., Chen, Z.: FT-CNN: algorithm-based fault tolerance for convolutional neural networks. IEEE Trans. Parallel Distrib. Syst. **32**(7), 1677–1689 (2021). https://doi.org/10.1109/TPDS.2020.3043449

On the Challenge of Hardware Errors, Adversarial Attacks and Privacy Leakage for Embedded Machine Learning

Ihsen Alouani

1 Introduction

Due to the recent breakthroughs in deep neural networks (DNNs) design and training, DL architectures are currently deployed to solving mainstream applications, along with industrial and critical applications: going from intelligent transportation systems [1–3], natural language processing [4], robotics [5], and healthcare [6]. This is in part owing to the VLSI technology progress, the new high-performance communication systems and the development of IoT devices. More specifically, this trend results in the generation of abundant amounts of data from different embedded sensors and IT systems, which are necessary for training accurate DNN models.

Given the computing-intensive aspect of DNNs, the by-default deployment of deep models is in Cloud data-centers or private data-centers. However, there are practical limits and drawbacks of such systems at least from 2 perspectives:

(i) First, from resource and power consumption and consequently environmental impact perspective, this scheme has considerable overheads.
(ii) Second, from a communication perspective, such a deployment scheme requires sending raw data from sensors to the servers all through wireless and wired communication platforms.

The downside of this scheme is that data-centers are power-hungry platforms; they are estimated to account for around 1% of worldwide electricity use with high environmental impact [7]. These trends motivate a ML computing paradigm that overcomes these issues. Specifically, a more distributed deployment of ML at the Edge emerged as a promising paradigm towards power-efficient near-sensor intelligent systems. While Embedded and Edge ML offers promising power/accuracy

I. Alouani (✉)
Centre for Secure Information Technologies (CSIT), Queen's University Belfast, Belfast, UK
e-mail: i.alouani@qub.ac.uk

S. Pasricha, M. Shafique (eds.), *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing*, https://doi.org/10.1007/978-3-031-40677-5_19

trade-off and enhances the mainstream development of ML models towards sustainable and smart systems and cities, several problems still limit ML trustworthiness.

In this chapter, we focus on three aspects of ML trustworthiness, namely Robustness to errors, security, and privacy.

2 ML Robustness to Errors

In a context of performance-driven design requirements, new hardware generations continuously shrink transistors dimensions, thereby increasing circuits sensitivity to external events which can negatively affect their reliability. There are two scenarios in which errors occur in modern embedded systems:

- Deliberate fault injection attacks such as Rowhammer [8]. Intentional attacks are another potential source of faults. The widespread usage of CNNs led to the development of sophisticated attacks. Malicious users could intentionally tamper with the parameters of the model [9].
- Reliability-related events such as soft errors either in memories, i.e., Single Event Upsets (SEU) or in combinatorial circuits, i.e., Single Event Transients (SET). These events are typically caused by high energy particles striking electronic devices.

These errors can propagate through the neural network to create accuracy loss, and potentially global system failures that can be safety-critical or security sensitive in some cases.

In this section, we provide an exploratory analysis of DNNs vulnerability to errors.

2.1 Methodology

In most embedded ML accelerators, the model parameters are stored on-board. A memory corruption has a persistent and, hence, cumulative aspect and will remain until a new model is trained and implemented.

To reproduce models behavior under this threat, we simulate memory corruptions by injecting a number of bit-flips in random parameters of a model at runtime (inference). We subsequently evaluate the model robustness for different error rates and locations (Fig. 1).

We consider two data representations:

- IEEE-754 single-precision 32-bit float: This is the standard representation format for real numbers. It is the dominant representation in CPU and GPU architectures. For simplicity we refer to this representation as $\mathcal{F}$ in the rest of the chapter. are composed of three parts: a sign, an exponent and a fraction part (see Fig. 2). The normalized format of IEEE-754 floating point is expressed as follows:

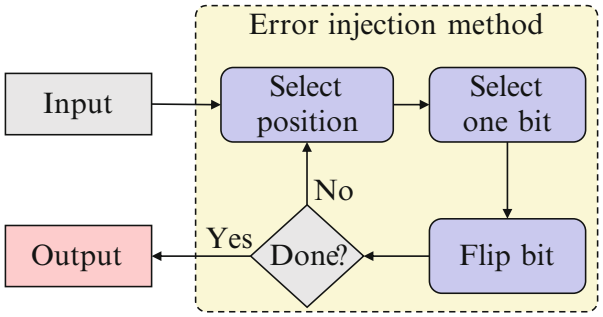


Fig. 1 Overview of the fault injection methodology

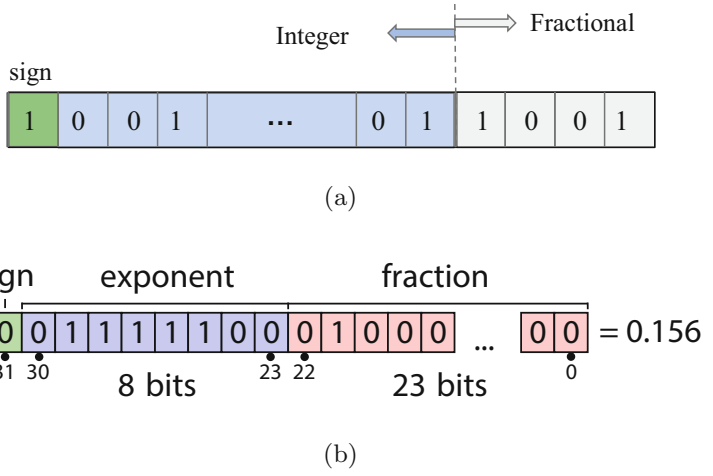


Fig. 2 Fixed-point representation in (**a**) with a bit-width of 8 and a fractional length of 2 (left) and −2 (right). On (**b**) the standard IEEE-754 representation of 32 bit floating-point values

$$val = (-1)^{sign} \times 2^{exp-bias} \times (1.fraction) \tag{1}$$

- Fixed-point representation: This representation uses two parameters: bit-width and fractional length. Negative fractional lengths can be used to represent powers of two. This representation is referred to as $\mathcal{Q}$ (for quantized) in the rest of the chapter.

To evaluate the robustness of a given model to faults, we create a fault injection framework that takes a trained network as an input. While testing the model at inference time, bit-flips are injected in the network's weights with a tunable injection rate. After each test, we report the overall accuracy under fault injection.

These tests are repeated 100 times for a statistically representative experiment. In each run, the engine generates a new set of errors and the injection of the generated errors is performed each run. We then report the accuracy distribution, i.e., the

average accuracy, the maximum, the minimum, and the standard deviation for the test.

2.2 *Results*

The results were obtained on weights in single-precision floating point comparatively with quantized weights in terms of classification accuracy of the different networks. The results of different runs are presented as the mean and the standard deviation of the top-1 accuracy.

Figure 3 illustrates the result of comparing the floating-point and quantized representations. The results show that quantized models are surprisingly more robust to fault injection than the full precision models, which has been consistently observed for 4 different CNNs with different fault injection rates. We believe that the reason behind this observation is the error distance after injection denoted by $\mathcal{A}$ in [10]. For instance, the $\mathcal{Q}$ representation with 7 decimal bits and 1 integer bit will differ from the original value by at most ± 1. However, for the full precision representation, the error distance on activation can reach 3×10^{38} as observed in [10]. Therefore, since floating-point numbers are more sensitive to bit-flips than fixed-point representation, quantized networks tend to show higher robustness to errors, in addition to the area and power consumption gains.

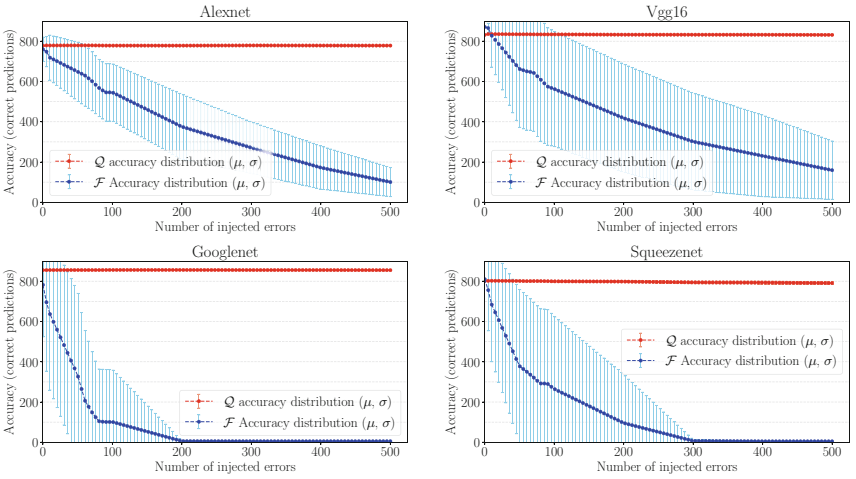


Fig. 3 Models accuracy under fault injection for weights representation with 8-bit fixed point ($\mathcal{Q}$) and 32-bit single-precision IEEE-754 ($\mathcal{F}$)

3 ML Security

ML systems have been deployed in a variety of application domains, including security-sensitive and safety-critical applications [11]. However, ML models have been shown vulnerable to several security threats, including adversarial examples, which consist of additive noise carefully crafted to fool ML models.

3.1 Adversarial Attacks

Adversarial examples are additive perturbations to an input that are carefully crafted by an adversary to deceive the model and force it to output a wrong label. If adversaries succeed in manipulating the decisions of a ML classifier to their advantage, this can tamper with the security and integrity of the system, and potentially threaten the safety of people in some applications like autonomous vehicles. For example, adding adversarial noise to a stop sign that leads an autonomous car to wrongly classify it as a speed limit sign can lead to crashes and loss of life. In fact, adversarial examples have been shown effective under real-world settings [12–14]: that when printed out, an adversarially crafted image can fool the classifiers even under different lighting conditions and orientations. Therefore, understanding and mitigating these attacks is essential to developing safe and trustworthy intelligent systems.

Attacker Knowledge When attacking a DNN-based model, we can distinguish two main attack scenarios based on *attacker knowledge*:

(i) Black-box setting: the adversary has partial or no access to the victim model's architecture and parameters. The adversary uses the results of querying the victim to reverse engineer the classifier and create a substitute model used to generate the adversarial examples. An illustration of this scenario is given by Fig. 4.
(ii) White-box setting: in which the adversary has complete knowledge of the training data of the victim model in addition to the target model's architecture and parameters. An illustration of this scenario is given by Fig. 5. (FGSM) [15] attack, Projected gradient descent (PGD) [16] attack, Carlini & Wagnar (C&W) [17] are the main white-box adversarial attacks.

The attacker intention is to slightly modify the source image so that it is classified incorrectly by the target model, without special preference towards any particular output which is known as *untargeted attack*. However, in a *targeted attack*, the attacker aims at a specified wrong target class.

Minimizing Injected Noise An adversary, using information learned about the classifier, generates perturbations to cause incorrect classification under the constraint of minimizing this perturbation magnitude to avoid detection. For illustration

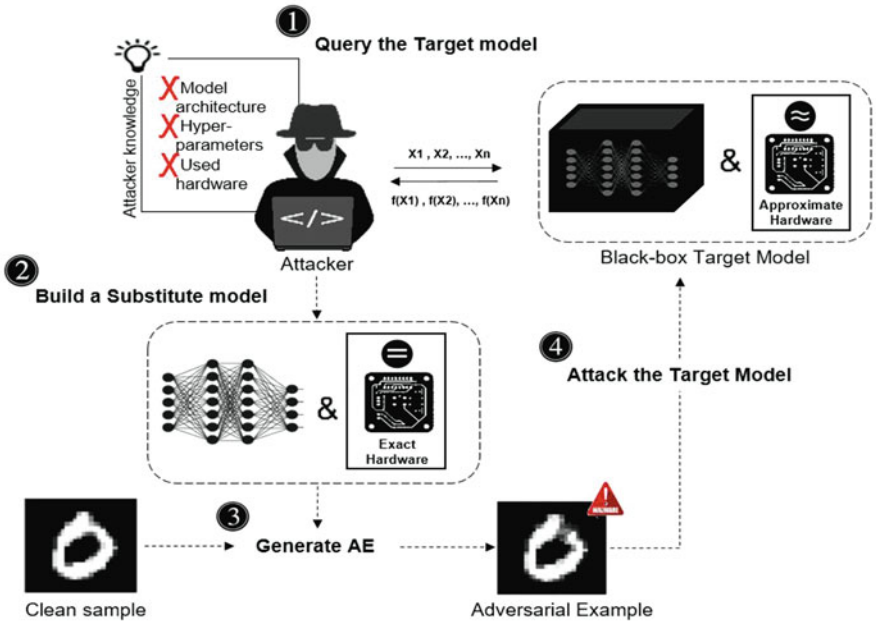


Fig. 4 Illustration of a black-box attack setting

purposes, consider a CNN used for image classification. More formally, given an original input image x and a target classification model $f()$ $s.t.$ $f(x) = l$, the problem of generating an adversarial example x^* can be formulated as a constrained optimization [17]:

$$x^* = \arg\min_{x^*} \mathcal{D}(x, x^*), s.t.\ f(x^*) = l^*,\ l \neq l^* \tag{2}$$

where $\mathcal{D}$ is the distance metric used to quantify the similarity between two images and the goal of the optimization is to minimize this added noise, typically to avoid detection of the adversarial perturbations. l and l^* are the two labels of x and x^*, respectively: x^* is considered as an adversarial example if and only if the label of the two images are different ($f(x) \neq f(x^*)$) and the added noise is bounded ($\mathcal{D}(x, x^*) < \epsilon$ where $\epsilon \geqslant 0$).

Distance Metrics The adversarial perturbations should be visually imperceptible by a human eye. Since it is hard to model humans' perception, three metrics have been practically used to measure the noise magnitude relatively to a given input, namely L_0, L_2, and L_∞ [17]. Notice that these three metrics are special cases of the L_p norm defined as follows:

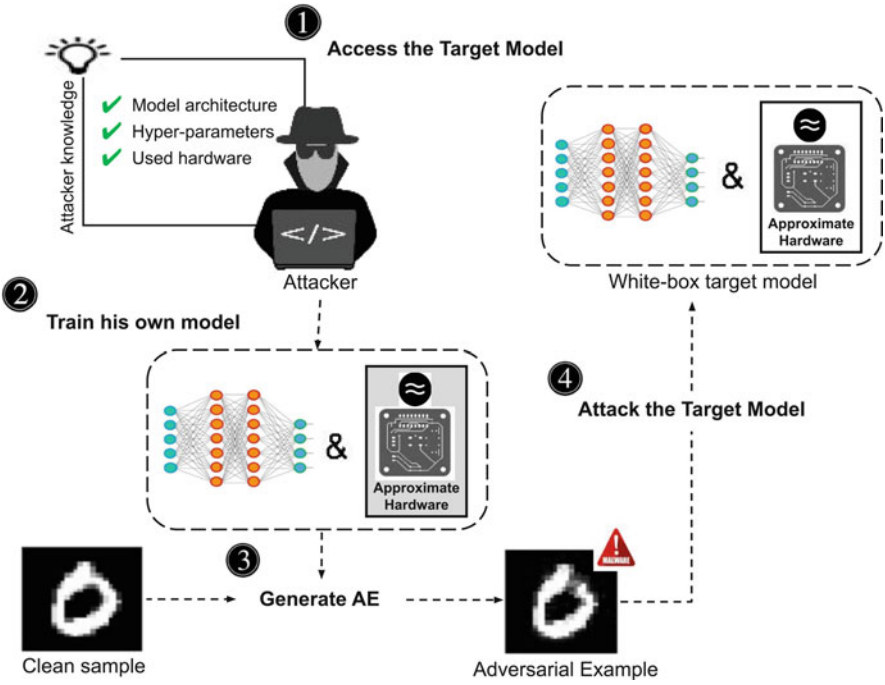


Fig. 5 Illustration of a white-box attack setting

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{\frac{1}{p}} \tag{3}$$

These metrics focus on different aspects of visual significance. For example, L_0 evaluates the number of pixels with different values at corresponding positions in two inputs. L_2 is the Euclidean distance between two images x and x^*, while L_∞ is the maximum difference for all pixels at corresponding positions in the two images.

Adversarial Attacks Generation Several methods have been proposed in the literature to generate adversarial examples. In the following we give a quick overview on the most popular ones:

Fast Gradient Sign Method (FGSM) FGSM is a single-step, gradient-based, attack. An adversarial example is generated by calculating a one-step gradient update following the direction of the sign of the loss gradient over the input, which is the direction that maximizes the target model's loss:

$$x_{adv} = x + \epsilon sign(\nabla_x J_\theta(x, y)) \tag{4}$$

where $\nabla J()$ is the gradient of the loss function J and θ is the set of model parameters and ϵ is the perturbation magnitude budget.

Projected Gradient Descent (PGD) PGD is a more efficient attack generation method; it is an iterative variant of FGSM where the adversarial noise is generated adaptively as follows:

$$x_{adv}^{t+1} = \mathcal{P}_{\mathcal{S}_x}(x_{adv}^{t} + \alpha \cdot sign(\nabla_x \mathcal{L}_\theta(x_{adv}^{t}, y))) \tag{5}$$

where $\mathcal{P}_{\mathcal{S}_x}()$ is a projection operator projecting the input into the feasible region $\mathcal{S}_x$ and α is the added noise at each iteration. PGD find the perturbation that maximizes the loss of a model on a particular input while keeping the size of the perturbation lower than the budget due to the projection operator.

Carlini & Wagner (C&W) This attack has 3 variants based on the used distance metric (l_0, l_2, l_∞). It generates adversarial examples by solving the following optimization problem:

$$\begin{aligned} &\underset{\delta}{\text{minimize}} \ \|\delta\|_2 + c \cdot l(x+\delta) \\ &\text{s.t. } x + \delta \in [0,1]^n \end{aligned} \tag{6}$$

where $\|\delta\|_2$ is the lowest noise that forces the model to misclassify. $l(\cdot)$ is the loss function defined as follows:

$$l(x) = max(max_{i \neq t}\{Z(x)_i\} - Z(x)_t - \kappa) \tag{7}$$

where $Z(x)$ is the output of the layer before the softmax called *logits*. t is the target label, and κ is the attack confidence. An adversarial example is considered as successful if $max_{i \neq t}\{Z(x)_i\} - Z(x)_t \leq 0$.

3.1.1 Defenses Against Adversarial Attacks

To protect ML models against adversarial attacks, several defense techniques can be found in the literature. We briefly introduce the different categories and provide insights from Embedded Systems perspective.

Adversarial Training (AT) AT is one of the most efficient state-of-the-art defense methods against adversarial attacks whose aim is to integrate the adversarial noise within the training process. It can be formulated as follows [16]:

$$\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\max_{\delta \in B(x,\varepsilon)} \mathcal{L}_{ce}(\theta, x+\delta, y) \right] \tag{8}$$

where θ indicates the parameters of the classifier, $\mathcal{L}_{ce}$ is the cross-entropy loss, $(x, y) \sim \mathcal{D}$ represents the training data sampled from a distribution $\mathcal{D}$, and $B(x, \varepsilon)$ is the allowed perturbation set. In this formulation, the inner maximization problem's objective is to explore the "adversarial surrounding" of a given training point and to take into account, not only the sample but also the worst-case noise from an adversarial perspective. The outer minimization problem is the conventional training aiming at minimizing the loss function (which includes adversarial noise) [16].

Nonetheless, the drawback of AT is its significant computational intensivity compared to the baseline training process. This is obviously due to the nested optimization problems in the formulation that need to be solved iteratively.

Input Pre-processing (IP) Input pre-processing is based on applying transformations to the input in order to remove adversarial perturbations [18, 19]. Transformations include the averaging, median, and Gaussian low-pass filters [19], as well as JPEG compression [20]. However, it has been shown that these defenses are vulnerable to white-box attacks [21]; in a white-box setting, where the adversary is aware of the defense, they can integrate the pre-processing function in the noise generation process. Furthermore, pre-processing requires computation overheads which is not suitable for resource-constrained devices such as Embedded Systems.

Gradient Masking (GM) GM leverages regularization to make the model's output less sensitive to input perturbations. Papernot et al. presented defensive distillation [22]. Nonetheless, this method is vulnerable to $C\&W$ attack [17]. Besides, GM techniques such as defensive distillation require a retraining process which results in time and energy overheads.

Randomization-Based Defenses These techniques leverage randomness to protect systems from adversarial noise. Lecuyer et al. [23] propose that random noise be added to the first layer of the DNN and the output be estimated via a Monte Carlo simulation. Raghunathan et al. [24] evaluate only a tiny neural network. Estimating the model output requires a heavy Monte Carlo simulation with a number of different model inference runs online, which cannot be afforded under resource constraints.

These defense strategies either require changing the DNN structure, modifying the training process or retrain the model only against known adversarial threats, which results in considerable overheads in time, resource utilization and energy consumption. In the following, we present defense strategies that take into account this aspect, which we call Embedded Systems-friendly defenses.

3.2 Embedded Systems-Friendly Defenses

Another set of defense techniques are inspired by hardware-efficiency techniques such quantization [25, 26]. The authors in [27] proposed Defensive approximation (DA), which leverages approximate computing (AC) to build robust models.

3.2.1 Defensive Approximation

The demand on high-performance embedded and mobile devices has been drastically increasing in the past decades. However, the technology is physically reaching the end of Moore's law, especially with the release of TSMC and Samsung 5 nm technology [28]. On the other hand, we observe that highly accurate computations might not be a must in all application domains. In fact, in a wide range of emerging applications, there is no specific accuracy requirements at the computing-element level, but rather a quality-of-service requirements on the system level. These application are inherently fault-tolerant by design and can relax the computational accuracy constraint. This observation has motivated the development of approximate computing (AC), a computing paradigm that trades power consumption with accuracy. The idea is to implement inexact/approximate elements that consume less energy, as far as the overall application tolerates the imprecision level in computation. This paradigm has been shown promising for inherently fault-tolerant applications such as deep learning, data analytics, and image/video/signal processing. Several AC techniques have been proposed in the literature and can be classified into three main categories based on the computing stack layer they target: software, architecture, and circuit level [29, 30].

Defensive approximation [27] tackles the problem of robustness to adversarial attacks from a new perspective, i.e., approximation in the underlying hardware, and leverages AC to secure DNNs. Specifically, at the lowest level, DA replaces exact conventional multipliers used in the convolution operations by an approximate multipliers. These approximate multiplier can generate inaccurate outputs, but the error distance needs to be under control. For this reason, the approximation occurs specifically in the mantissa multiplication, exclusively, to avoid high magnitude noise in the case of errors in the exponent or the sign bit of floating-point numbers. Subsequently, the convolution layers are built based on the approximate multipliers, which injects AC-induced noise within model layers. This noise is leveraged to protect DNNs against adversarial attacks. Moreover, in addition to the by-product gains in resources due to AC, this defense requires no retraining or fine-tuning of the protected model.

DA targets both robustness and energy/resource challenges. In fact, DA exploits the inherent fault tolerance of deep learning systems to provide resilience while also obtaining by-product gains of AC in terms of energy and resources. The AC-induced perturbations tend to help the classifier generalize and enhances its confidence and consequently enhance the classifier's robustness. Figure 6 gives an overview on DA

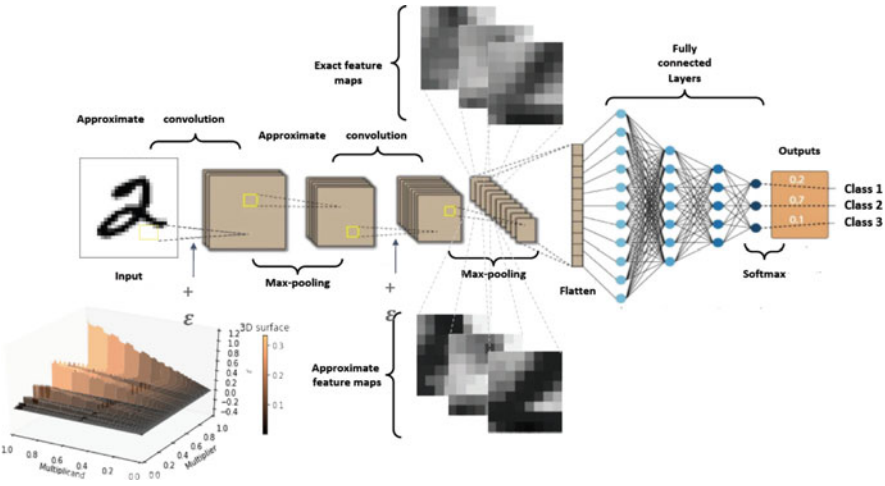


Fig. 6 Defensive approximation overview

mechanism within a CNN. It shows the distribution of the error distance due to the approximate multiplier. This noise distribution propagates within the model and impacts the features map, thereby defusing the adversarial noise mechanism. In the following, we discuss the exploration of approximation space with regards to the baseline accuracy of the models.

3.2.1.1 Baseline Accuracy

Before exploring the impact of AC on the security of DNNs, the protected model needs to maintain models' utility as a bottom line. For this reason, we explore the impact of the approximate multiplier on the accuracy for different levels of approximation, i.e., starting from approximating the full network, comparatively with having increasing exact layers along with the approximate model. Table 1 gives an overview on the utility as a function of the approximation level of the model for CIFAR-10 and ImageNet datasets.

3.2.1.2 Impact on Robustness

To evaluate the impact of AC on robustness, we consider a powerful adversary that has full access to the defense mechanism as well as the victim model architecture and parameters. Hence, we measure the model accuracy under adversarial examples created using different attacks for several noise budgets.

Table 1 Impact of approximation on model classification accuracy for a set of clean Inputs from CIFAR-10 and ImageNet

Model	Top 1 accuracy	
	CIFAR-10	ImageNet
Full exact model	100%	100%
Full approximate model	85.7%	73.23%
Exact: 1st conv layer	98.34%	97.18%
Exact: 2nd conv layer	93.4%	83.60%
Exact: 3rd conv layer	93.4%	83.60%
Exact: 1st FC layer	88.04%	75.4%
Exact: 2nd FC layer	88.04%	75.4%
Exact: 3rd FC layer	88.04%	75.4%
Exact: all FC layers	95%	78.87%

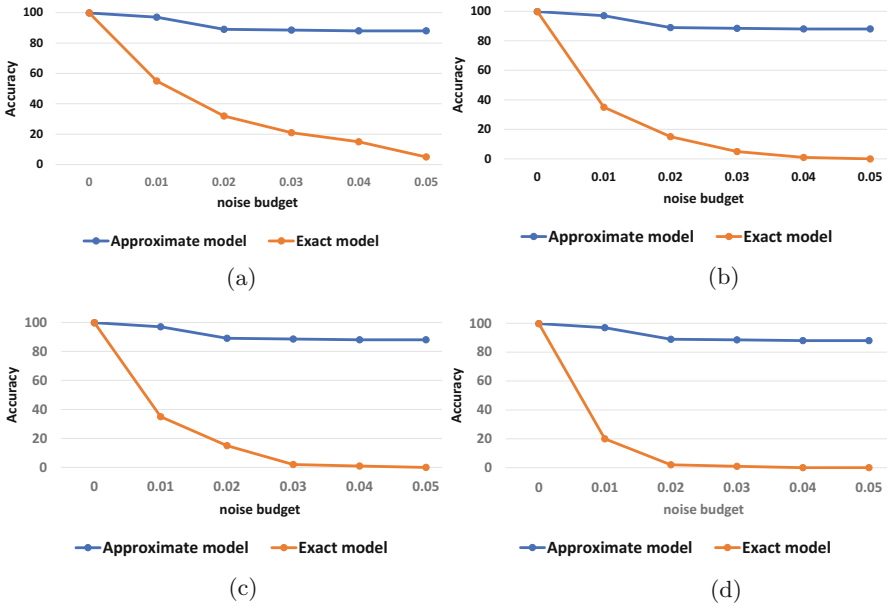


Fig. 7 Model accuracy for different noise budgets under white-box attack. (**a**) CIFAR-10 using FGSM. (**b**) CIFAR-10 using PGD. (**c**) ImageNet using FGSM. (**d**) ImageNet using PGD

Figure 7 summarizes the effectiveness of DA defense against FGSM and PGD attacks for different noise budgets (ε). The approximate hardware prevents the attacker from generating efficient AE for deeper networks and complex data distribution. Even with a high amount of injected noise ($\epsilon = 0.06$), DA model accuracy remains as high as 90% under PGD attack.

3.2.2 Undervolting as a Defense

3.2.2.1 Approach

This approach explores using voltage over-scaling (VOS) as a lightweight defense against adversarial attacks [31]. It consists of reducing supply voltage at runtime, i.e., inference, without accordingly scaling down the frequency (Fig. 8). This creates stochastic hardware-induced noise at computation circuitry that is leveraged to defend DNNs against adversarial attacks. The rationale behind choosing VOS is as follows:

(i) ***Stochastic noise:*** The impact of injecting random noise on DNNs robustness has been proven theoretically in [23, 32]. However, none of these works provides a practical implementation of the randomness source, especially one that does not require high overhead and considerable complexity to cope with Embedded ML requirements. This approach leverages a fundamental property of VOS, which is a stochastic behavior of the induced timing violations within the circuit.

(ii) ***Controllable noise magnitude:*** While injected random noise can be used to improve the robustness of DNNs [23], its magnitude should be under control. In fact, injecting high magnitude noise can have drastic impact on baseline accuracy. Nonetheless, VOS-induced noise magnitude is directly controllable by the supply voltage.

3.2.2.2 Setup

To match the fault rates with the voltage levels, we used a Xilinx Zynq Ultrascale+ ZCU104 FPGA platform that hosts a VGG-16 CNN. The device's Processing System (PS) includes a quad-core Arm Cortex-A53 applications processor (APU), as well as a dual-core Cortex-R5 real-time processor (RPU). We leveraged an external voltage controller, the Infineon USB005, to perform undervolting characterization on the FPGA device, which is connected to the board via an I2C wire. We can read and write the different voltage rail supplies to the board using PowerIRCenter GUI.

3.2.2.3 Impact on Robustness

Figure 9 shows the accuracy of the exact model and undervolted models for LeNet-5, AlexNet, and ResNet-18 CNNs under ℓ_∞ and ℓ_2 C&W attack. While the baseline exact model (h_{exact}) yields high classification accuracy, it drops drastically under $C\&W$ attack reaching near 0 for $\varepsilon = 0.4$. Most importantly, approximate model with a fault rate $fr = 10^{-4}$ maintains a high robustness (accuracy under attack) even for high magnitude ε. This observation holds for AlexNet and ResNet-18 as well.

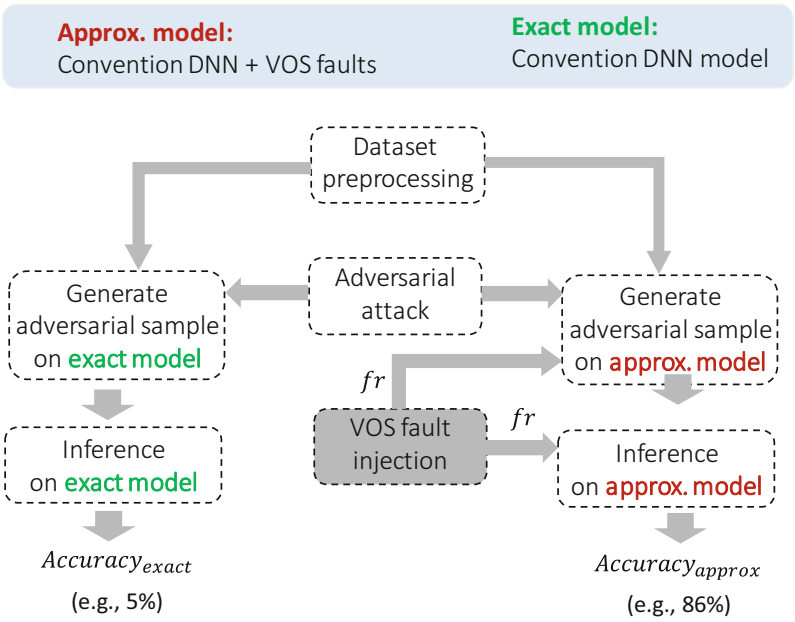


Fig. 8 Experimental setup for undervolted models robustness

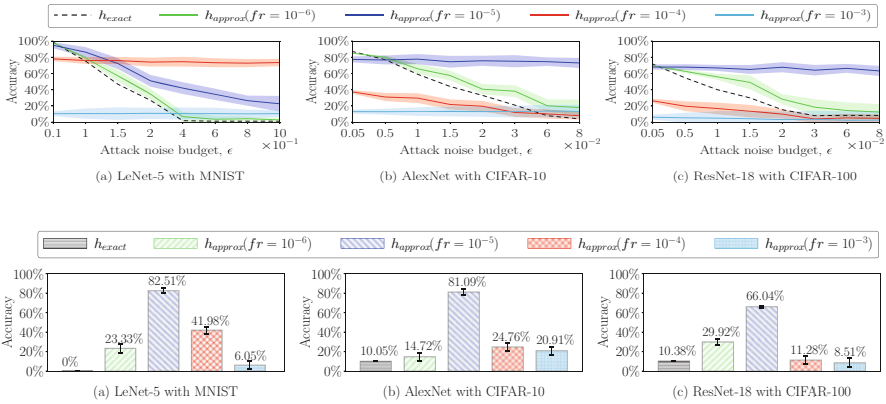


Fig. 9 Robustness of VOS-models under C&W attack for both ℓ_∞ (top) and ℓ_2 (bottom) metrics

While VOS offers a practical source of randomness that enhances DNNs robustness to adversarial attacks, it also comes with an obvious by-product gain in terms of power consumption, and offers an ad-hoc defense that does not require modifying the model or retraining it.

Trade-off The results show that a VOS-induced noise protects DNNs against adversarial attacks. However, aggressive undervolting results in a drop in utility.

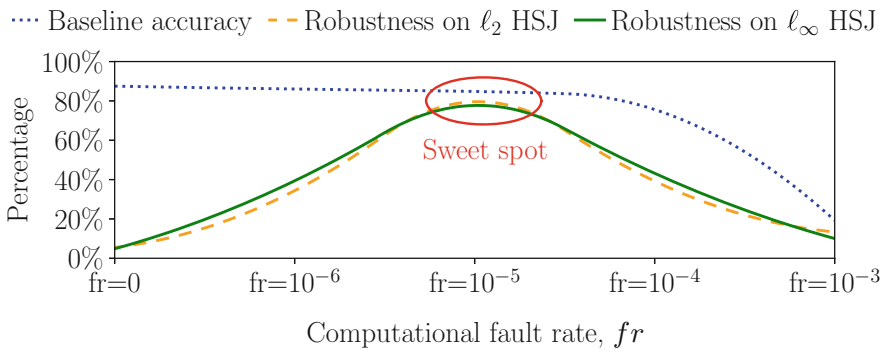


Fig. 10 An illustration of the accuracy/robustness trade-off for AlexNet with CIFAR-10 on HSJ. In the figure, $fr = 0$ indicates the exact model, h_{exact}

A trade-off between accuracy and robustness with by-product power savings could be found, to achieve high-robustness models without accuracy drop. An example of a robustness/accuracy trade-off is depicted in Fig. 10. Notice that f_r represent the fault rates, which are directly defined by the VOS level. The figure shows that with a simple space exploration, we can identify a sweet-spot for a given CNN that yields the highest possible robustness with the lowest possible accuracy drop.

3.3 Privacy

Confidentiality is a fundamental design property, especially for systems that process, store, or communicate private and sensitive data. In ML, insuring a model privacy consists in protecting the model against information leakage, whereby an adversary aims to infer sensitive information such as training data by interacting with the victim. In fact, the promising performance of ML systems spread their use to sensitive applications ranging from medical diagnosis in health-care to surveillance and biometrics. These models are trained on various data such as clinical/biomedical records, personal photos, genome data, financial, social, location traces, etc. Moreover, they are also trained with crowd-sourced data as cloud providers (e.g., Amazon AWS, Microsoft Azure, Google API) in a ML-as-a-Service fashion, which allow novice users to train models that often contains *personally identifiable information*.

ML models are vulnerable to privacy threats, which are critical when data confidentiality is an issue, e.g., when revealing the identity of the patients in clinical records. Membership Inference Attacks [33] aim at determining whether a data sample belongs to the training dataset. More generically, Property Inference Attacks [34] infer certain properties that hold only for a fraction of the training data, and are independent from the features that the DNN model aims to learn. On the other hand, Model Stealing methods [35] aim at duplicating the functionality of the

ML model and extract its parameters, and Model Inversion Attacks [36] aim to infer sensitive features of the training data.

Towards avoiding these leakages of confidential information, several privacy-preserving techniques can be employed. **Homomorphic Encryption (HE)** ensures that the data remains confidential, since the attacker does not have access to the decryption keys. CryptoNets [37] apply HE to perform DNN inference on encrypted data, and the work of [38] extends the encryption to the complete training process. However, HE-based techniques are very costly in terms of execution time and resources.

Another state-of-the-art technique towards privacy-preserving ML is **Differential Privacy** (DP) which consists of injecting random noise to the stochastic gradient descent process (Noisy SGD) [39], or through Private Aggregation of Teacher Ensembles (PATE) [40], in which the knowledge learned by an ensemble of "teacher" models is transferred to a "student" model. While DP is one of the most efficient defenses against information leakage, it comes at a considerable cost in terms of utility, i.e., it results in a baseline accuracy drop.

Training a deep neural network requires a large amount of data, which represents practically the most valuable asset in ML ecosystems. In some specific applications, data protected by privacy regulations and user level agreements. These can be specific to application domains such as HIPPA regulations in the US which prohibits patients' data sharing and GDPR in Europe, which is more generic in regulating user data collection [41]. Therefore, in medical applications, a given health institution might not be able to collect enough data that is representative and relevant to train an efficient ML model.

In another scenarios, data may be created on Edge devices, but owners are reluctant to sharing it due to privacy concerns (industrial applications, text messages, etc.), bandwidth challenges, or both.

Federated learning (FL) recently emerged as a potential solution to overcome these aforementioned issues. Specifically, FL allows train ML models collaboratively between different nodes without sharing their local data [42]. FL allows multiple participants (also called clients) to train local models and then consolidate those models into a global model. This global model benefits from all client data, without directly sharing the data, preserving data privacy. Each client trains its model on its private data, and then communicates model updates to a central server (also called aggregator). By avoiding communicating the data to a central back end for training, this data remains local to each client and therefore private. Moreover, distributing the training leads to benefits in performance and network bandwidth. In an FL model, each participant updates the global model by training it on its local data and shares the metadata with a central server. Only the trained local model **updates** are shared, and the local data to each client remains private. The server aggregates the local model updates into a single federated model and shares this model with the participants, allowing them to benefit from a model trained on the overall data. The federated model can continue to be refined as more data becomes available. This process is illustrated in Fig. 11.

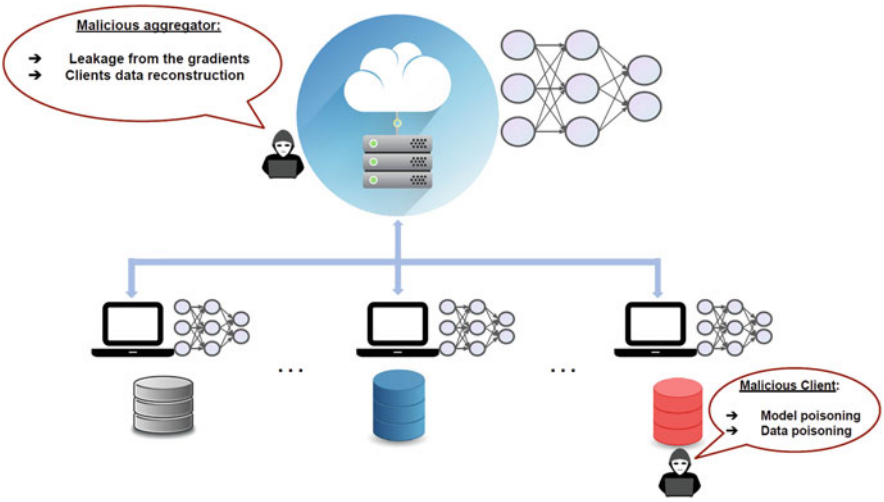


Fig. 11 An overview on FL setting: Client devices send locally trained model updates to server for aggregation of the federated model

While FL has been branded by major companies such as Google as a privacy-preserving solution, it has been shown that it is vulnerable to several attacks that can jeopardize its and confidentiality:

Model Poisoning and Data Poisoning Each of the clients in FL setting is able to arbitrarily change its local model maliciously that they send to server. The model can be manipulated either directly through its parameters or indirectly by poisoning the local training set to degrade the quality of the aggregated model making it misclassify more often, or be more susceptible to adversarial inputs. In model poisoning, a malicious client attempts to change the global model by poisoning their local model parameters directly [43]. In contrast, in data poisoning, the attacker manipulates its local training samples, affecting the model's performance indirectly throughout a substantial portion of the input space [44].

Deep Leakage from Gradients With access to the gradient of a particular client, an adversary is able reconstruct the training samples of the client. In fact, attacks like Deep Leakage from gradient (DLG) [45] and iDLG [46] show the possibility to **reconstruct training data samples** from raw gradients only. The recovered images are pixel wise accurate, and generated through an optimization problem aiming at reducing the difference between the gradient of a given candidate input and the real gradient.

Defenses and Limits Differential Privacy has been used as a defense against data leakage [39]. However, it does not protect against poisoning attacks. Moreover, secure aggregation techniques such as [47] aim at preventing the server from accessing the individual model updates, while allowing the aggregation operation.

However, this defense results by construction in an impossibility to detect integrity attacks.

To defend against integrity attacks, and limit the influence of individual participants, robust aggregation techniques have been proposed (also called Byzantine-tolerant aggregation) [48, 49].

Fairness FL approach is designed under the assumption of non-iid data. The incentive of participants to share their model updates generated on local data is to enhance the model accuracy, specifically on their own data distribution. However, robust aggregation techniques consider the tail of the gradient updates distribution as a potential integrity attack and cuts it off in the aggregation phase. Therefore, users with "atypical" data, i.e., in the tail of the overall users data distribution will not benefit from the FL setting since their contributions are discarded by the robust aggregation mechanism [50]. This results in a fairness problem: users with minority and atypical data distributions will be disadvantaged by the FL setting.

Open Problems FL offers an interesting solution towards privately sharing "knowledge representations" without necessarily sharing raw data, which allows to train more generalizing and efficient models. However, a three objectives that are necessary for FL deployment seem to be difficult to obtain simultaneously, i.e., privacy, integrity, and fairness. In fact, secure aggregation techniques solve the privacy problem and open an attack surface on the model integrity. On the other hand, tackling the integrity problem with robust aggregation schemes results in the loss of the global model fairness.

We believe that a fundamental problem to solve by the community is finding interesting and adaptive trade-off between these three objectives.

4 Conclusion

This chapter focuses on three aspects of ML trustworthiness, especially in the context of embedded systems and the Edge:

(i) The first is ML models robustness to errors, either due to hardware reliability issues or deliberately injected by malicious actors.
(ii) The second aspect is the security of ML models, especially from an adversarial ML perspective. More specifically, we explored defense techniques that are Embedded Systems-friendly, i.e., that do not result in a high overhead in power consumption or hardware resources.
(iii) The third is the privacy problem, where we focused on federated learning as an emerging training paradigm that is compatible with Embedded Systems and IoT applications.

References

1. Ben Khalifa, A., Alouani, I., Mahjoub, M.A., Rivenq, A.: A novel multi-view pedestrian detection database for collaborative intelligent transportation systems. Fut. Gener. Comput. Syst. **113**, 506–527 (2020). https://doi.org/10.1016/j.future.2020.07.025
2. Jegham, I., Khalifa, A.B., Alouani, I., Mahjoub, M.A.: Soft spatial attention-based multimodal driver action recognition using deep learning. IEEE Sensors J. **21**(2), 1918–1925 (2021). https://doi.org/10.1109/JSEN.2020.3019258
3. Al-Qizwini, M., Barjasteh, I., Al-Qassab, H., Radha, H.: Deep learning algorithm for autonomous driving using GoogLenet. In: 2017 IEEE Intelligent Vehicles Symposium (IV), pp. 89–96. IEEE, Piscataway (2017)
4. Deng, L., Liu, Y.: Springer, Berlin (2018)
5. Pierson, H.A., Gashler, M.S.: Deep learning in robotics: a review of recent research. Advanced Robotics **31**(16), 821–835 (2017)
6. Miotto, R., Wang, F., Wang, S., Jiang, X., Dudley, J.T.: Deep learning for healthcare: review, opportunities and challenges. Briefings in bioinformatics **19**(6), 1236–1246 (2018)
7. Masanet, E., Shehabi, A., Lei, N., Smith, S., Koomey, J.: Recalibrating global data center energy-use estimates. Science **367**(6481), 984–986 (2020). https://doi.org/10.1126/science.aba3758
8. Kim, Y., Daly, R., Kim, J.S., Fallin, C., Lee, J., Lee, D., Wilkerson, C., Lai, K., Mutlu, O.: Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In: ACM/IEEE 41st International Symposium on Computer Architecture, ISCA 2014, Minneapolis, pp. 361–372 (2014). https://doi.org/10.1109/ISCA.2014.6853210
9. Liu, Q., Liu, T., Liu, Z., Wang, Y., Jin, Y., Wen, W.: Security analysis and enhancement of model compressed deep learning systems under adversarial attacks. In: Proceedings of the 23rd Asia and South Pacific Design Automation Conference. ASPDAC '18, pp. 721–726. IEEE Press, Piscataway (2018). http://dl.acm.org/citation.cfm?id=3201607.3201772
10. Neggaz, M.A., Alouani, I., Lorenzo, P.R., Niar, S.: A reliability study on CNNs for critical embedded systems. In: 2018 IEEE 36th International Conference on Computer Design (ICCD), pp. 476–479. IEEE (2018)
11. Neggaz, M.A., Alouani, I., Niar, S., Kurdahi, F.J.: Are CNNs reliable enough for critical applications? An exploratory study. IEEE Des. Test **37**(2), 76–83 (2020). https://doi.org/10.1109/MDAT.2019.2952336
12. Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T., Song, D.: Robust physical-world attacks on deep learning visual classification. In: 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18–22, 2018, pp. 1625–1634. Computer Vision Foundation/IEEE Computer Society (2018). https://doi.org/10.1109/CVPR.2018.00175
13. Man, Y., Li, M., Gerdes, R.M.: GhostImage: Remote perception attacks against camera-based image classification systems. In: 23rd International Symposium on Research in Attacks, Intrusions and Defenses, RAID 2020, San Sebastian, Spain, October 14–15, 2020, 317–332 (2020)
14. Tarchoun, B., Alouani, I., Ben Khalifa, A., Mahjoub, M.A.: Adversarial attacks in a multi-view setting: An empirical study of the adversarial patches inter-view transferability. In: 2021 International Conference on Cyberworlds (CW), pp. 299–302 (2021). https://doi.org/10.1109/CW52790.2021.00057
15. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: Bengio, Y., LeCun, Y. (eds.): In: 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7–9, 2015, Conference Track Proceedings (2015)
16. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30–May 3, 2018, Conference Track Proceedings (2018)

17. Carlini, N., Wagner, D.A.: Towards evaluating the robustness of neural networks. In: 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22–26, 2017, pp. 39–57 (2017). https://doi.org/10.1109/SP.2017.49
18. Guesmi, A., Alouani, I., Baklouti, M., Frikha, T., Abid, M.: Sit: stochastic input transformation to defend against adversarial attacks on deep neural networks. IEEE Design Test 1–1 (2021). https://doi.org/10.1109/MDAT.2021.3077542
19. Osadchy, M., Hernandez-Castro, J., Gibson, S., Dunkelman, O., Pérez-Cabo, D.: No bot expects the DeepCAPTCHA! Introducing immutable adversarial examples, with applications to CAPTCHA generation. IEEE Trans. Inform. Forensics Secur. **12**(11), 2640–2653 (2017)
20. Das, N., Shanbhogue, M., Chen, S.-T., Hohman, F., Chen, L., Kounavis, M.E., Chau, D.H.: Keeping the Bad Guys Out: Protecting and Vaccinating Deep Learning with JPEG Compression (2017)
21. Chen, J., Wu, X., Rastogi, V., Liang, Y., Jha, S.: Towards understanding limitations of pixel discretization against adversarial attacks. In: 2019 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 480–495. IEEE (2019)
22. Papernot, N., McDaniel, P., Wu, X., Jha, S., Swami, A.: Distillation as a defense to adversarial perturbations against deep neural networks. In: 2016 IEEE Symposium on Security and Privacy (SP), pp. 582–597 (2016). https://doi.org/10.1109/SP.2016.41
23. Lecuyer, M., Atlidakis, V., Geambasu, R., Hsu, D., Jana, S.: Certified robustness to adversarial examples with differential privacy. In: 2019 IEEE Symposium on Security and Privacy (SP), pp. 656–672 (2019)
24. Raghunathan, A., Steinhardt, J., Liang, P.: Certified Defenses against Adversarial Examples (2018)
25. Lin, J., Gan, C., Han, S.: Defensive quantization: when efficiency meets robustness. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6–9, 2019 (2019)
26. Khalid, F., Ali, H., Tariq, H., Hanif, M.A., Rehman, S., Ahmed, R., Shafique, M.: QuSecNets: quantization-based defense mechanism for securing deep neural network against adversarial attacks. In: 25th IEEE International Symposium on On-Line Testing and Robust System Design, IOLTS 2019, Rhodes, Greece, July 1–3, 2019, 182–187 (2019). https://doi.org/10.1109/IOLTS.2019.8854377
27. Guesmi, A., Alouani, I., Khasawneh, K.N., Baklouti, M., Frikha, T., Abid, M., Abu-Ghazaleh, N.B.: Defensive approximation: securing CNNs using approximate computing. In: ASPLOS '21: 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Virtual Event, USA, April 19–23, 2021, pp. 990–1003 (2021). https://doi.org/10.1145/3445814.3446747
28. Moore, S.K.: Another step toward the end of Moore's law: Samsung and TSMC move to 5-nanometer manufacturing—[news]. IEEE Spectr. **56**(6), 9–10 (2019). https://doi.org/10.1109/MSPEC.2019.8727133
29. Guesmi, A., Alouani, I., Baklouti, M., Frikha, T., Abid, M., Rivenq, A.: Heap: a heterogeneous approximate floating-point multiplier for error tolerant applications. In: Proceedings of the 30th International Workshop on Rapid System Prototyping (RSP'19). RSP '19, pp. 36–42. Association for Computing Machinery, New York (2019). https://doi.org/10.1145/3339985.3358495
30. Alouani, I., Ahangari, H., Ozturk, O., Niar, S.: A novel heterogeneous approximate multiplier for low power and high performance. IEEE Embedded Syst. Lett. **10**(2), 45–48 (2018). https://doi.org/10.1109/LES.2017.2778341
31. Islam, S., Alouani, I., Khasawneh, K.N.: Lower voltage for higher security: using voltage overscaling to secure deep neural networks. In: 2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD), pp. 1–9 (2021). https://doi.org/10.1109/ICCAD51958.2021.9643551
32. Cohen, J.M., Rosenfeld, E., Kolter, J.Z.: Certified adversarial robustness via randomized smoothing. In: Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA, vol. 97, pp. 1310–1320 (2019)

33. Shokri, R., Stronati, M., Song, C., Shmatikov, V.: Membership inference attacks against machine learning models. In: 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22–26, 2017, pp. 3–18 (2017). https://doi.org/10.1109/SP.2017.41
34. Ganju, K., Wang, Q., Yang, W., Gunter, C.A., Borisov, N.: Property inference attacks on fully connected neural networks using permutation invariant representations. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15–19, 2018, pp. 619–633 (2018). https://doi.org/10.1145/3243734.3243834
35. Tramèr, F., Zhang, F., Juels, A., Reiter, M.K., Ristenpart, T.: Stealing machine learning models via prediction APIs. In: 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10–12, 2016, pp. 601–618 (2016)
36. Fredrikson, M., Jha, S., Ristenpart, T.: Model inversion attacks that exploit confidence information and basic countermeasures. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12–16, 2015, pp. 1322–1333 (2015). https://doi.org/10.1145/2810103.2813677
37. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K.E., Naehrig, M., Wernsing, J.: Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In: Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19–24, 2016, vol. 48, pp. 201–210 (2016)
38. Nandakumar, K., Ratha, N.K., Pankanti, S., Halevi, S.: Towards deep neural network training on encrypted data. In: IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2019, Long Beach, CA, USA, June 16–20, 2019, pp. 40–48 (2019). https://doi.org/10.1109/CVPRW.2019.00011
39. Abadi, M., Chu, A., Goodfellow, I.J., McMahan, H.B., Mironov, I., Talwar, K., Zhang, L.: Deep learning with differential privacy. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24–28, 2016, pp. 308–318 (2016). https://doi.org/10.1145/2976749.2978318
40. Papernot, N., Song, S., Mironov, I., Raghunathan, A., Talwar, K., Erlingsson, Ú.: Scalable private learning with PATE. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30–May 3, 2018, Conference Track Proceedings (2018)
41. Annas, G.J., et al.: HIPAA regulations-a new era of medical-record privacy? N. Engl. J. Med. **348**(15), 1486–1490 (2003)
42. Kairouz, P., McMahan, H.B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A.N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., et al.: Advances and open problems in federated learning (2019). arXiv preprint arXiv:1912.04977
43. Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D., Shmatikov, V.: How to backdoor federated learning. In: International Conference on Artificial Intelligence and Statistics, pp. 2938–2948. PMLR (2020)
44. Jere, M.S., Farnan, T., Koushanfar, F.: A taxonomy of attacks on federated learning. IEEE Secur. Privacy **19**(2), 20–28 (2020)
45. Zhu, L., Han, S.: Deep Leakage from Gradients, pp. 17–31. Federated Learning (2020)
46. Zhao, B., Mopuri, K.R., Bilen, H.: iDLG: Improved deep leakage from gradients (2020). arXiv preprint arXiv:2001.02610
47. Chen, Y., Su, L., Xu, J.: Distributed statistical machine learning in adversarial settings: Byzantine gradient descent. Proc. ACM Meas. Anal. Comput. Syst. **1**(2), 1–25 (2017)
48. Damaskinos, G., El-Mhamdi, E.-M., Guerraoui, R., Guirguis, A., Rouault, S.: AggregaThor: byzantine machine learning via robust gradient aggregation. Proc. Mach. Learn. Syst. **1**, 81–106 (2019)
49. Rajput, S., Wang, H., Charles, Z., Papailiopoulos, D.: Detox: a redundancy-based framework for faster and more robust gradient aggregation. In: Advances in Neural Information Processing Systems, vol. 32 (2019)
50. Yu, T., Bagdasaryan, E., Shmatikov, V.: Salvaging federated learning by local adaptation (2020). arXiv preprint arXiv:2002.04758

A Systematic Evaluation of Backdoor Attacks in Various Domains

Stefanos Koffas, Behrad Tajalli, Jing Xu, Mauro Conti, and Stjepan Picek

1 Introduction

In the last few years, deep learning has become very popular, and it has been applied to a variety of applications like computer vision [29], machine translation [54], speech recognition [18], and game playing [44]. It is also used in safety and security-critical applications like autonomous driving [12], malware detection [8], biometric-based user authentication [6], and side-channel analysis [40]. Such systems commonly need large datasets to train reliable models that generalize well and perform adequately with unseen data. However, large datasets are often scrapped from untrusted sources on the web [1, 11]. Additionally, the hardware needed to train such models can be very expensive and is not always available to developers who want to embed some machine learning functionality into their applications. Thus, a new programming paradigm has emerged: Machine Learning

S. Koffas · J. Xu
Cybersecurity Group, Delft University of Technology, Delft, The Netherlands
e-mail: s.koffas@tudelft.nl; j.xu-8@tudelft.nl

B. Tajalli
Digital Security Group, Radboud University, Nijmegen, The Netherlands
e-mail: hamidreza.tajalli@ru.nl

M. Conti
Cybersecurity Group, Delft University of Technology, Delft, The Netherlands

SPRITZ Security and Privacy Research Group, University of Padua, Padua, Italy
e-mail: mauro.conti@unipd.it

S. Picek (✉)
Cybersecurity Group, Delft University of Technology, Delft, The Netherlands

Digital Security Group, Radboud University, Nijmegen, The Netherlands
e-mail: stjepan.picek@ru.nl

S. Pasricha, M. Shafique (eds.), *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing*, https://doi.org/10.1007/978-3-031-40677-5_20

as a Service (MLaaS), made possible by the recent advances in cloud computing. These new trends lead to novel attack vectors that adversaries can exploit.

One of these attack vectors is the backdoor attack [19]. In this attack, an adversary embeds a secret functionality into a trained model, activated only if the model's input contains a specific property (trigger). At the same time, for any input that does not include the trigger, the model behaves as expected to avoid raising any suspicions. Most of the designed attacks in the literature target computer vision applications [31], but recently different applications have been targeted. In particular, backdoor attacks were shown in text classification [5, 9], audio recognition [28, 62], graph data [55, 57], federated learning [3, 45], and reinforcement learning [58]. A backdoor attack can be dangerous as machine learning is used in many security-related applications. In [19], the authors showed that a stop sign with a small post-it note could be identified as a speed limit by a compromised autonomous vehicle with serious consequences to its passengers and pedestrians. AI-enabled applications like spam-filtering [37], speaker identification [62], or malware detection [42] could also be bypassed if the model used contains a backdoor. Thus, backdoor attacks pose a serious threat, and it is required to understand the limits of such attacks to provide better defenses.

This work explores the effects of various trigger characteristics on the backdoor attack. In particular, we implement backdoor attacks with triggers of varying sizes, positions, and poisoning rates and apply them to four different domains (image, text, sound, and graph data). With it, we aim to better understand backdoor attacks and find common properties among different domains.

In [47], the authors claimed that the backdoor attack becomes ineffective when the adversary cannot alter the training labels and is forced to poison only samples from the target class. In this case, the model cannot learn a strong connection between the trigger and the target class as more substantial features from the target class are learned. This behavior is reasonable and well justified but only supported by one experiment with the CIFAR10 dataset. Here, we aim to test this claim in image classification but also in different domains, like text and sound classification.

Our contributions are:

- We run extensive experiments in different application domains (image, text, audio, and graph data) and systematically evaluate the effect of various trigger characteristics on the backdoor attack.
- We investigate two different backdoor attacks in each application and verify that the clean-label attack is not very effective as it may require large poisoning rates to achieve a high attack success rate. However, this attack could work by choosing more effective triggers without changing the poisoning rate.
- We show that in most cases, the backdoor's effectiveness increases as the trigger size increases.

2 Background

2.1 Computer Vision

Today, the computer vision domain covers diverse use cases and concepts within, ranging from capturing raw data to image pattern extraction and interpreting information from those images. It is mostly a combination of concepts, ideas, and techniques of pattern recognition, digital image processing, artificial intelligence (AI), and computer graphics [53]. Computer vision aims to provide the capability for a system to identify and perceive the visual world in the same way as human vision does. Recently, by applying AI techniques, including deep neural networks, the machines even outperformed humans in several tasks [13].

Nowadays, there are multiple applications of computer vision in our daily life, e.g., weather prediction, medical cases, sports and entertainment, industry and production lines, and human-computer interaction [24, 25, 36, 46, 49, 51, 60]. While the use cases and applications are becoming broader and more prevalent in our everyday lives, security issues regarding the techniques and algorithms are also becoming a significant challenge to deal with.

2.2 Natural Language Processing

Natural language processing (NLP) is at the intersection of computational linguistics, computer science, and artificial intelligence. It aims to make machines that understand human language and reason about it. NLP is an umbrella term that covers many different applications that deal with human language in both spoken and written formats. Applications that belong to natural language processing are, among others, speech recognition, speaker identification, question answering, text sentiment analysis, hate speech detection, natural language generation (speech-to-text and text-to-speech models), spam detection, and text translation. Initially, NLP was based on static rules, but now it uses deep learning for most tasks [23].

Recent advances in NLP have led to very efficient human-computer interfaces that have been broadly deployed. Virtual assistants like Siri and Google assistant and popular IoT devices like Amazon Alexa have been widely used with great success. However, such applications open up new attack vectors that put the user's security and privacy at risk. Therefore, before their wide adoption in the industry, we must ensure that such systems work securely.

2.3 Graph Data

Many real-world applications can be modeled as graphs, such as social networks, gene interactions, and transport networks. Similar to the great success of deep learning models in, e.g., image classification and natural language processing, deep graph models (graph neural networks—GNNs) have also achieved promising performance in processing graph data for different tasks, e.g., the graph classification task and node classification task.

Graph Neural Networks (GNNs) GNNs take a graph G as an input, including its structure information and node features, and learn a representation vector (embedding) for each node $v \in G$, z_v, or the entire graph, z_G. Modern GNNs follow a neighborhood aggregation strategy, where one iteratively updates the representation of a node by aggregating representations of its neighbors. After k iterations of aggregation, a node's representation captures both structure and feature information within its k-hop network neighborhood. Formally, the k-th layer of a GNN is (e.g., GCN [26], GraphSAGE [20], and GAT [48]):

$$Z^{(k)} = AGGREGATE(A, Z^{(k-1)}; \theta^{(k)}), \tag{1}$$

where $Z^{(k)}$ are the node embeddings in the matrix form computed after the k-th iteration and the $AGGREGATE$ function depends on the adjacency matrix A, the trainable parameters $\theta^{(k)}$, and the previous node embeddings $Z^{(k-1)}$. $Z^{(0)}$ is initialized as G's node features.

For the node classification task, the node representation $Z^{(k)}$ of the final iteration is used for prediction. For the graph classification task, the READOUT function pools the node embeddings from the final iteration K:

$$z_G = READOUT(Z^{(K)}). \tag{2}$$

READOUT can be a simple permutation invariant function such as summation or a more sophisticated graph-level pooling function [59, 63].

Graph-Level Classification Graph-level classification aims to predict the class label(s) for an entire graph [63]. The end-to-end learning for this task can be realized using graph convolutional layers and readout layers. While graph convolutional layers are responsible for extracting high-level node representations, the readout layer collapses node representations of each graph into a graph representation. By applying a multilayer perceptron and a Softmax layer to graph representations, one can build an end-to-end framework for graph classification.

Node-Level Classification Given a graph with a few labeled nodes, GNNs can learn a robust model that effectively identifies the class labels for the unlabeled nodes [26]. In a node-level classification task, there are two types of training settings—inductive and transductive. In an inductive setting, the unlabeled nodes are not seen during training, while in a transductive setting, the test nodes (but not

their labels) are also observed during the training process. The transductive training setting is popular, and in this work, we used a backdoor attack in the transductive node-level classification task.

2.4 *Backdoor Attacks*

Backdoor attacks aim to make a model misclassify some of its inputs to a preset-specific label while other classification results behave normally. This misclassification is activated when a specific property is included in the model input. This property is called the trigger and can be anything the targeted model understands. For instance, a random pixel pattern [6, 19] or an actual item [52] in computer vision, a specific phrase in text classification [32], a tone in speech recognition [28], or a subgraph with specific properties in graph data [55]. The framework for the backdoor attack is shown in Fig. 1.

The first backdoor attacks targeted computer vision [6, 19] under a simple threat model, where an adversary could inject a small portion of poisoned data into the training dataset. In particular, the adversary injects into the training dataset data

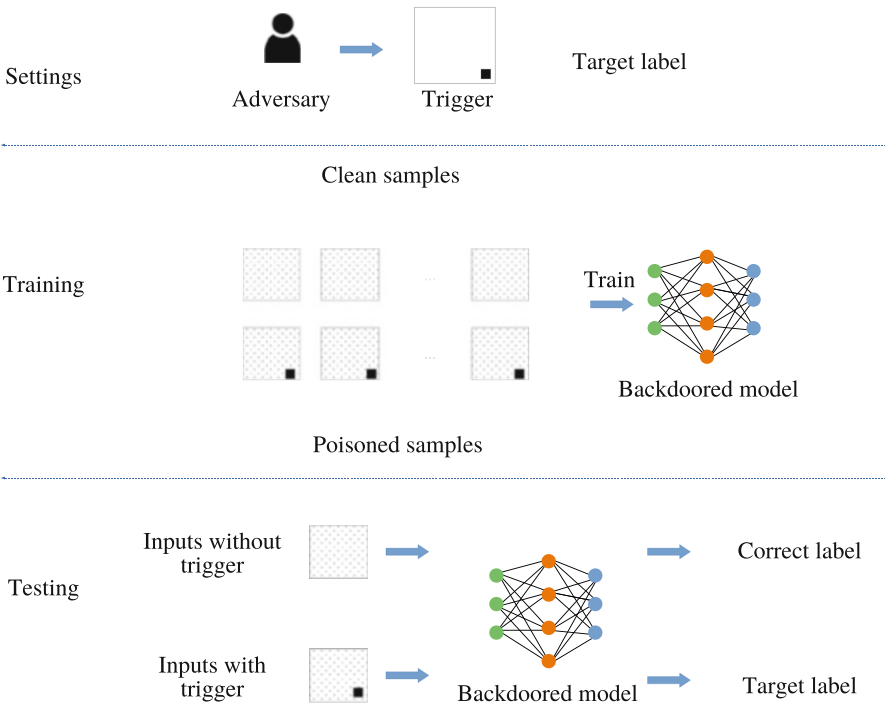


Fig. 1 Framework for the backdoor attack

stamped with a trigger that belongs to the target class. As a result, the trained model strongly associates this pattern with the target class, and whenever it is added to an input, the classification result will be the target class. Recent trends in machine learning like Machine Learning as a Service (MLaaS), outsourced training, transfer learning, and crowdsourced datasets have made this setup possible.

In MLaaS, a cloud provider provides a pay-per-request API[1] that can be used for predictions. However, the user can only use such an API as a black box without being able to verify how the model makes its predictions. Similarly, during outsourced training, the user's model is trained on the cloud and returned to the user after the training ends. Due to the lack of formal verification tools for the trained models, the user can never verify that the returned model does not contain any backdoors. Furthermore, in [19], the authors showed that a backdoor could remain effective even after a poisoned model was repurposed through transfer learning. Large crowdsourced datasets like ImageNet [11] and Mozilla's common voice [1] are so vast that cannot be exhaustively verified [39]. Thus, an adversary could inject a few poisoned samples resulting in the backdoored models.

This threat can pose real challenges as an adversary could bypass a face identification biometric access control system [6] or force an autonomous vehicle to ignore a stop sign and continue its course [19]. For this reason, backdoor attacks became very popular among researchers resulting in many novel attacks and countermeasures [15]. Novel attacks are not only limited to data poisoning but can also be based on code poisoning [2] or the direct modification of the model's parameters [22]. At the same time, due to the inability to completely understand how a deep learning model works and the lack of formal verification methods about a model's functionality, most countermeasures are empirically based on specific assumptions [4, 16, 50]. Unfortunately, in most cases, an adaptive attacker with a slightly different approach could bypass such defenses [7, 30, 43].

There are several variations of the backdoor attack resulting from different poisoning strategies. The first distinction is the class-agnostic and the class-specific backdoors [15]. The class-agnostic backdoor can be activated by a trigger injected into any input. On the other hand, the class-specific backdoor is activated only if the poisoned input belongs to a specific class. The main difference between these two strategies is that in the second case, the model needs to identify both features of the trigger and the source class making possible countermeasures more challenging [16]. Considering class-agnostic backdoor attacks, we can differentiate between the "simple" backdoor attack [19] and the clean-label backdoor attack [47].

Simple Backdoor Attack In the rest of this paper, by the simple backdoor attack, we are referring to the data poisoning backdoor attack that was introduced in BadNets [19]. In this case, the adversary adds a small subset of poisoned samples to the training dataset. These samples have been stamped with the adversary-chosen

[1] https://aws.amazon.com/transcribe/.

trigger, and their label has been changed to the target class. The target class is the output of the poisoned model when the backdoor is activated.

Clean-Label Backdoor Attack The clean-label backdoor attack was introduced in [47]. This attack is similar to a simple attack, but the adversary cannot affect the label of the injected data. The reasoning behind this attack is that the poisoned training samples can be easily identified as outliers by simple filtering mechanisms or even human inspection because the original class of these samples is different from the target class. Thus, an adaptive adversary may have to poison samples only from the target class, hoping that the model identifies the trigger pattern as a class feature. This attack is still a data poisoning backdoor attack but uses a weaker adversary making the attack more challenging.

Based on the trigger, backdoors can be either static [19] or dynamic [27]. The static backdoors are activated with a trigger that has very specific characteristics. In computer vision, such a trigger could mean a specific pixel pattern or a specific position. On the other hand, the dynamic backdoors can be activated by various triggers with different characteristics.

For graph neural networks, the first backdoor attack was proposed in [65]. In this backdoor attack, a GNN classifier predicts an attacker-chosen target label for a testing graph once a predefined subgraph is injected into the testing graph. All perturbed graphs are injected with the same trigger graph. Another backdoor attack against GNNs for the graph classification task was presented in [55], but it differs from [65] in which a universal trigger graph is assumed for all the embedded graphs. This kind of backdoor attack dynamically adapts triggers to individual graphs. The adaptive trigger is optimized in both topological structure and node features. The training processes of the trigger generation function and the backdoored GNN model are assumed as a bi-level optimization objective [14]. The authors also adapted a backtracking-based algorithm to replace a subgraph in the original graph with the adaptive trigger graph. Xu et al. [57] explored backdoor attacks on GNNs with several explainability tools. In this work, the backdoor attack is implemented with the same strategy [65] for the graph classification task. The authors also proposed a new backdoor attack strategy for the node classification task. All the above-mentioned attacks in GNNs are gray box backdoor attacks since the adversary only modifies the training dataset instead of interfering with the training of models.

2.4.1 Metrics

The successful backdoor attack should always be activated when the trigger is embedded into the model's input because an adversary wants to remain stealthy and interact with the poisoned model as little as possible. Additionally, the backdoor should not affect the original task when the trigger is not included in the input. When the poisoned model does not perform well on the original task, the backdoored model will (1) raise suspicions that something is wrong and (2) not be used, thus

preventing the adversary's plans. As a result, to measure the success of a backdoor attack, we require two metrics: the attack success rate and the clean accuracy drop.

2.4.1.1 Attack Success Rate (ASR)

The ASR shows the reliability of the attack, and it represents the number of successfully triggered backdoors from a number of poisoned inputs:

$$ASR = \frac{\sum_{i=1}^{N} F(M^*(x_i) = y_t)}{N}, \tag{3}$$

where M^* is the poisoned model, x_i is a poisoned input, y_t is the target class, and $F(x)$ is a function that returns 1 if x is true and 0 otherwise.

2.4.1.2 Clean Accuracy Drop (CAD)

This quantity shows the backdoor's effect on the original task. It is calculated by comparing the performance of a poisoned and a clean model for clean inputs. The accuracy drop should generally be small to keep the attack stealthy.

3 Methodology

3.1 Threat Model

In this work, we implement data poisoning backdoor attacks. The adversary injects a small subset of poisoned data without knowing any information about the model architecture or the training algorithm. Thus, the attack follows a gray box threat model. This threat model is realistic as current large datasets are crowdsourced [1, 11] and malicious data may go through the validation process [39]. So, an adversary could inject trigger-stamped data in such datasets that will remain unnoticed and used during training resulting in a successful backdoor attack.

In our experiments, we investigate two different attacks, the simple data poisoning attack [19] and the clean-label attack that does not alter the labels of the poisoned data [47]. For both attacks, the adversary aims to cause targeted misclassifications with a very high probability without affecting the model's performance on the original task.

3.2 Image Classification

Attacks We use two different attacks: the simple backdoor attack and the clean-label attack.

Datasets For our image classification backdoor attacks, we use two popular image datasets: (i) CIFAR10 that consists of 60,000 32×32 color images in ten classes, with 6000 images per class. There are 50,000 training images and 10,000 test images. (ii) Fashion-MNIST (FMNIST) [56]—a dataset of Zalando's article images consisting of a training set of 60,000 images and a test set of 10,000 images. Each image is a 28×28 gray-scale image associated with a label from ten classes.

For the CIFAR10 dataset, we split the test set in an i.i.d manner into two 5000 sample datasets, each used for validation and test, respectively. For the FMNIST dataset, we split the training set into two different sized datasets in an i.i.d manner: the first having 50,000 samples used for training and the second having 10,000 samples for validation. With this, we have the same size of training samples for both datasets, so comparing results between these two is easier.

Features The input features for both neural networks are the tensor of images. For the CIFAR10 dataset, each RGB image is considered as a [3, 32, 32] shape tensor. For FMNIST, however, the images are gray-scale, so the input has only one channel ([1, 28, 28] shape tensor). We also did the standard normalization for input values before all train, validation, and test phases.

Models We use two models: STRIPNet [16] and ResNet [21] with nine residual blocks (ResNet-9).

Trigger As described in [27], various triggers have been used in image classification, and all of them resulted in successful backdoor attacks. This means that the trigger shape and pattern are not crucial for the success of a backdoor attack. Thus, for our experiments, we chose a square trigger. Its pixel intensities are random values retrieved from a continuous uniform distribution (pseudorandom generator). The seed in this generator was fixed for consistency in our experiments.

3.3 Natural Language Processing

Attacks Similar to image classification, we use simple and clean-label backdoor attacks.

Datasets In our experiments, we used the IMDB [33] and the AG News topic classification [64] datasets. The IMDB dataset consists of 50,000 (50%/50% training/test split) movie reviews of high polarity (either positive or negative). We used 20% of the training data for validation. The AG News topic classification dataset consists of news articles belonging to four categories (world, sports, business, and science/technology). The training set consists of 120,000 samples and the testing set

of 7600 samples. Again, we used 20% of the training data for validation resulting in 96,000 and 24,000 samples for training and validation sets, respectively.

Features The first step of our pipeline is a `TextVectorization` layer that transforms each input to a convenient form for processing as described in [27]. As the datasets are different, we used a different sequence length for each dataset. We forced the length of each sentence to be 250 words for the IMDB dataset and 197 for the AG News dataset. Additionally, we used a vocabulary of 10,000 words that proved enough for such small datasets.

Models We used two publicly available CNN architectures. Both the first CNN[2] and the second CNN[3] use an embedding layer as their input. However, the first CNN uses a small trainable embedding of size 16, and the second uses a pretrained GloVe embedding [38] of size 100. We want to investigate if the attack becomes more difficult when the model uses a pretrained embedding because this is more frequent in practice. Such embeddings have been trained in large corpora of text and interpret possible connections between different words more accurately. To illustrate, Google's pretrained word2vec is trained with 100 billion words from Google News, and it contains 300-dimensional vectors for 3 million words and phrases [34]. The GloVe is trained from a corpus of 6 billion words and has a vocabulary of 400,000 words [38].

Trigger As the trigger, we used a sentence of 1 up to 4 words from the list ["trope," "everyday," "mythology," "sparkles," "ruthless"] as defined in [32]. We applied the trigger in three positions (beginning, middle, and end) to investigate whether our models are more sensitive in specific positions.

3.4 Speech Recognition

Attacks Again, we use simple and clean-label backdoor attacks.

Datasets For this application, we used two versions of the Speech Commands dataset as described in [28]. The first version uses ten classes of the dataset and the second 30 classes. From our experiments, we excluded the samples that lasted less than one second to avoid variable-sized inputs in our pipeline resulting in 21,312 .wav files in the first case and 58,252 files in the second case. In both cases, we use 64%/16%/20% for training, validation, and testing.

Features As our input features, we used the MFCCs of each training input. The exact hyperparameters for this calculation are described in [28].

Models We used one CNN [32] and one LSTM [10] for our experiments.

[2] https://www.tensorflow.org/tutorials/keras/text_classification.

[3] https://keras.io/examples/nlp/pretrained_word_embeddings/.

Trigger Our dataset's sound files are sampled at 16 kHz, and according to the Nyquist-Shannon sampling theorem, the largest tone frequency that can be included in such digital signals is 8 kHz. Thus, following [27], our trigger is a 7 kHz tone which is a high pitch audible sound. Following the rest of the triggers tried, this trigger differs from the normal dataset samples. It lasts from 20 to 80 ms because we want to model an adversary that is as stealthy as possible. The trigger is injected in three different positions of each sound sample (beginning, middle, and end).

3.5 Graph Data

Attacks As described in Sect. 2.4, for graph neural networks, we utilize two backdoor attacks, i.e., AT^{I} [65] and AT^{II} [55]. The framework for AT^{I} is illustrated in Fig. 2. In the training phase (Fig. 2a), the attacker injects a trigger (subgraph g_t) to a subset of training graphs and changes their labels to an attacker-chosen target label. A GNN classifier is then trained using the backdoored training dataset, and such GNN is called backdoor GNN Φ_b. In the test phase (Fig. 2b), once the test graph is injected with the same trigger graph, the backdoored GNN is likely to misclassify the testing sample to the target label. For the node classification task, we used the backdoor attack from [57].

Since [65] and [57] designed the same strategy to implement the backdoor attack for the graph classification task, we illustrate the results of [65] and [55] for the graph classification task. The results based on [57] are presented for the node classification task.

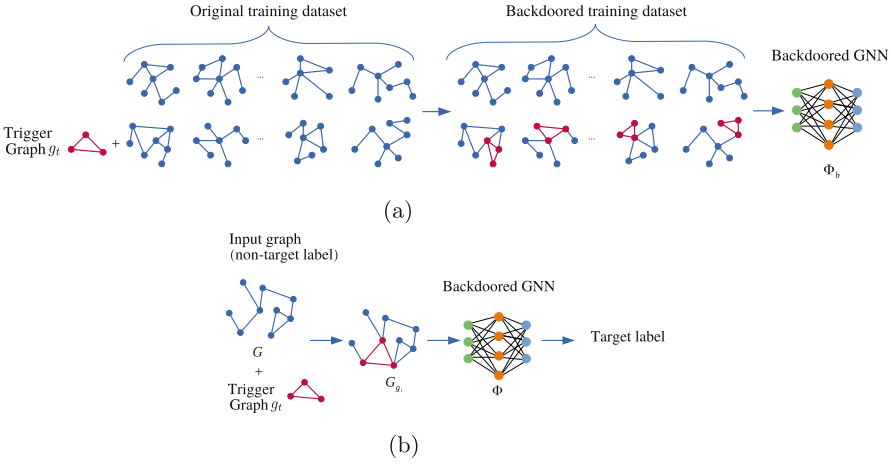


Fig. 2 Subgraph-based backdoor attack for the graph classification task. (**a**) Training. (**b**) Testing

Table 1 Graph datasets statistics

Datasets	# Graphs	Avg. # nodes	Avg. # edges	Classes	Class distribution
AIDS	2000	15.69	16.20	2	400[0], 1600[1]
TRIANGLES	45,000	20.85	32.74	10	4500[0–9]
Cora	1	2708	5429	7	351[0], 217[1], 418[2], 818[3], 426[4], 298[5], 180[6]
CiteSeer	1	3327	4608	6	264[0], 590[1], 668[2], 701[3], 596[4], 508[5]

Datasets Table 1 shows the statistics for all considered datasets for graph neural networks. For the graph classification task, we use two publicly available graph datasets. (i) AIDS [35]—a dataset consisting of graphs representing molecular compounds that are active against HIV or not; (ii) TRIANGLES [35]—a synthetic dataset designed to solve the task of counting the number of triangles in a graph. For each graph classification dataset, we sample 2/3 of the graphs as the original training dataset and treat the remaining graphs as the original testing dataset. Among the original training dataset, we randomly sample α fraction of graphs to inject the trigger and relabel them with the target label, called the backdoored training dataset. Several parameters can affect the attack effectiveness: trigger size s, trigger density ρ, and poisoning intensity α. Unlike other domains, e.g., image classification, the trigger position in graph data is irrelevant and cannot be defined because a graph is non-Euclidean data where we cannot put nodes in some order. For AT^{I}, we use Erdős-Rényi (ER) model [17] to generate the trigger graph, as it is more effective than the other methods [65].

For the node classification task, we use two real-world datasets: (i) Cora [41]—a citation network in which each publication is described by a binary-valued word vector indicating the absence/presence of the corresponding word in the collection of 1433 unique words. (ii) CiteSeer [41]—another citation network with more nodes but less edges. For each node classification dataset, we split 20% of the total nodes as the original training dataset (labeled) and the rest as the original testing dataset (unlabeled). To generate the backdoored training dataset, we sample α of the original training dataset to inject the feature trigger and relabel these nodes with the target label. The feature trigger width is set to be n. Moreover, based on the conclusion in [57], different feature trigger injecting positions have a negligible impact on the attack performance, so the trigger injecting position is randomly selected. Here, we explore the impact of poisoning intensity α and feature trigger width n on the attack performance. In the node classification task, each node feature has a value of 0 or 1, and here we set the value of the modified node features to 1 (note, the values could also be set to 0).

Features Each graph contains topological and node feature information. For each graph dataset in this work, there is an adjacency matrix and feature information matrix. For AIDS, Cora, and CiteSeer, there is a specific node feature vector for

each node in the graph, but for TRIANGLES, the one-hot degree of a node is used as the node feature.

Models We use two state-of-the-art GNN models for the graph classification task: GCN [26] and GraphSAGE [20]. We use GCN [26] and GAT [48] for the node classification task.

Trigger For the graph classification task, our trigger is a global (adaptive) subgraph in $AT^{\mathrm{I}}(AT^{\mathrm{II}})$. For the node classification task, our trigger is a subset of node features with a fixed value, e.g., 0.

4 Experimental Results

4.1 Image Classification

Chosen Settings and Selected Parameters We ran our experiments with a different number of poisoned samples (25, 300, 575, 850), trigger sizes (4×4, 8×8, 12×12), and trigger positions (Upper-Mid, Mid-Left, Mid-Right, Lower-Mid) on the image. Figure 3 demonstrates four different positions of a 4×4 trigger for several FMNIST sample images. We repeated each experiment two times, which makes the total number of 768 experiments regarding the chosen settings. We set class number 5 as the target for all experiments and in both datasets.

Every backdoor attack should remain stealthy without affecting the original task. Therefore, the poisoned model should perform as expected when the input does not contain the trigger. In Table 2, we compare the performance of clean and backdoored models for clean inputs. The attack accuracy mentioned in this table is the arithmetic mean ($\pm$ the standard deviation) of the accuracy on clean inputs from all the poisoned models trained in our experiments. For the original accuracy, we trained multiple clean models and averaged their performance. The model remains unaffected from both backdoor attacks even if we use 850 poisoned samples. Such poisoning rates are small and cannot affect the model's performance in general. From Table 2, we can also verify that our models perform similarly well in both

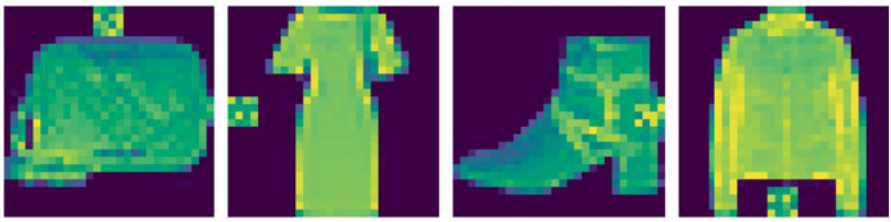


Fig. 3 Applied 4×4 trigger in different positions: Upper-Mid, Mid-Left, Mid-Right, Lower-Mid

Table 2 Clean accuracy drop in image classification

Dataset	Model	Original Acc		Number of poisoned Samples			
			Attack type	25	200	375	850
CIFAR10	STRIPNet	85.09 (± 0.599)	Clean-label	85.22 (± 0.508)	85.38 (± 0.523)	85.37 (± 0.378)	85.23 (± 0.471)
			Simple	85.31 (± 0.467)	85.28 (± 0.438)	85.35 (± 0.360)	85.12 (± 0.477)
	Resnet-9	89.99 (± 0.499)	Clean-label	89.98 (± 0.356)	89.86 (± 0.300)	89.79 (± 0.461)	89.69 (± 0.377)
			Simple	89.94 (± 0.353)	89.77 (± 0.274)	89.75 (± 0.266)	89.74 (± 0.378)
FMNIST	STRIPNet	93.62 (± 0.205)	Clean-label	93.66 (± 0.080)	93.68 (± 0.127)	93.64 (± 0.131)	93.69 (± 0.139)
			Simple	93.59 (± 0.135)	93.61 (± 0.148)	93.67 (± 0.152)	93.65 (± 0.139)
	Resnet-9	93.63 (± 0.154)	Clean-label	93.59 (± 0.152)	93.53 (± 0.157)	93.62 (± 0.171)	93.59 (± 0.176)
			Simple	93.57 (± 0.129)	93.55 (± 0.181)	93.59 (± 0.164)	93.58 (± 0.173)

datasets, which is helpful when comparing the performance of the attack for each case.

Results for FMNIST As it can be inferred from Fig. 4, the clean-label attack is not that effective against the FMNIST dataset. By increasing the number of poisoned samples, there are small or no improvements in attack success rate (there are small improvements when increasing the number of samples from 25 to 300, but as we increase from 300 to 575 and from 575 to 850, the improvements become even smaller). We assume this is mainly due to the dataset nature and the capability of the CNNs to learn the exclusive features of each class easily and robustly so that injecting a trigger (even of size 12×12) could not disturb the network from learning those.

Since both ResNet-9 and STRIPNet have convolutional layers, we expect negligible effects of trigger position on attack success rate. The results confirm this as there are only minor effects stemming from the trigger positions (for instance, in both networks, the trigger on the lower-mid results in the least ASR, while on the mid-right, it has a little more chance of being learned by the network. Again, we suppose this is because of the attributes of the FMNIST images and the models' focus on specific regions of an image to learn). Additionally, in almost all cases (except a few ones like upper-mid in ResNet-9), increasing the trigger size leads to higher ASR.

For the simple attack, we obtained 100% ASR for 300 attack samples or more. With 25 poisoned samples, some trigger positions have positive effects on ASR regardless of trigger size (for instance, the mid-left trigger achieves high ASR even with 4×4 size triggers).

Results for CIFAR10 The clean-label attack is significantly more effective for CIFAR10 than FMNIST (Figs. 4 and 5). We believe this is primarily because the CIFAR10 images are RGB, and the crafted trigger has more layers (3 channels). As a result, the model learns the embedded trigger with less poisoned samples. As expected, the trigger position does not play an important role in ASR, and in almost all cases, ASR improves using a larger trigger size.

Another observation is that the smaller the size of the trigger, the more noticeable the ASR improvement when increasing the number of poisoned samples from 25 to 850. For instance, for a 12×12 trigger, there is no noticeable improvement in ASR when the number of poisoned samples increases from 575 to 850. On the other hand, using a 4×4 trigger, ASR's growth is easily observable between all four different poisoning rates.

Analyzing the simple attack, similarly to FMNIST, we achieved 100% ASR for 300 poisoned samples or more. Additionally, ResNet-9 is more vulnerable to backdoor attacks, particularly when using fewer poisoned samples and smaller triggers. We believe this is mostly because ResNet-9 is a larger network than STRIPNet and can extract more data from the given dataset.

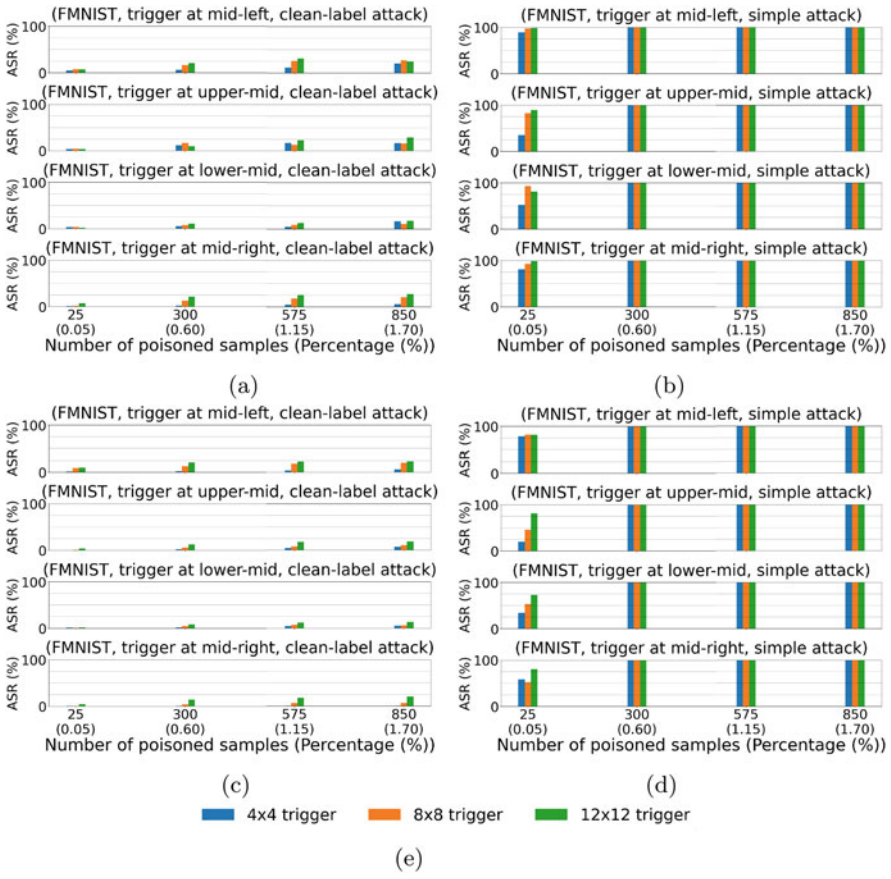


Fig. 4 Attack accuracy for the FMNIST dataset. From these figures, we conclude that the clean-label attack is not effective but is slightly improved when increasing the poisoning rate. On the other hand, the simple attack can be very effective even with a small poisoning rate (0.6%). Additionally, larger triggers lead to higher ASR, but different trigger positions do not result in ASR fluctuations as the convolutional layers identify the trigger. (**a**) ResNet-9 + clean-label attack. (**b**) ResNet-9 + simple attack. (**c**) STRIPNet + clean-label attack. (**d**) STRIPNet + simple attack. (**e**) Legend

4.2 Natural Language Processing

In Tables 3 and 4, we compare the performance of clean and backdoored models in text classification when clean inputs are used. These tables are generated by averaging the performance of clean and poisoned models as described in Sect. 4.1. In all cases, the model's performance remains almost unaffected after the backdoor insertion. There are a few minor accuracy drops that are at most 0.6% making the

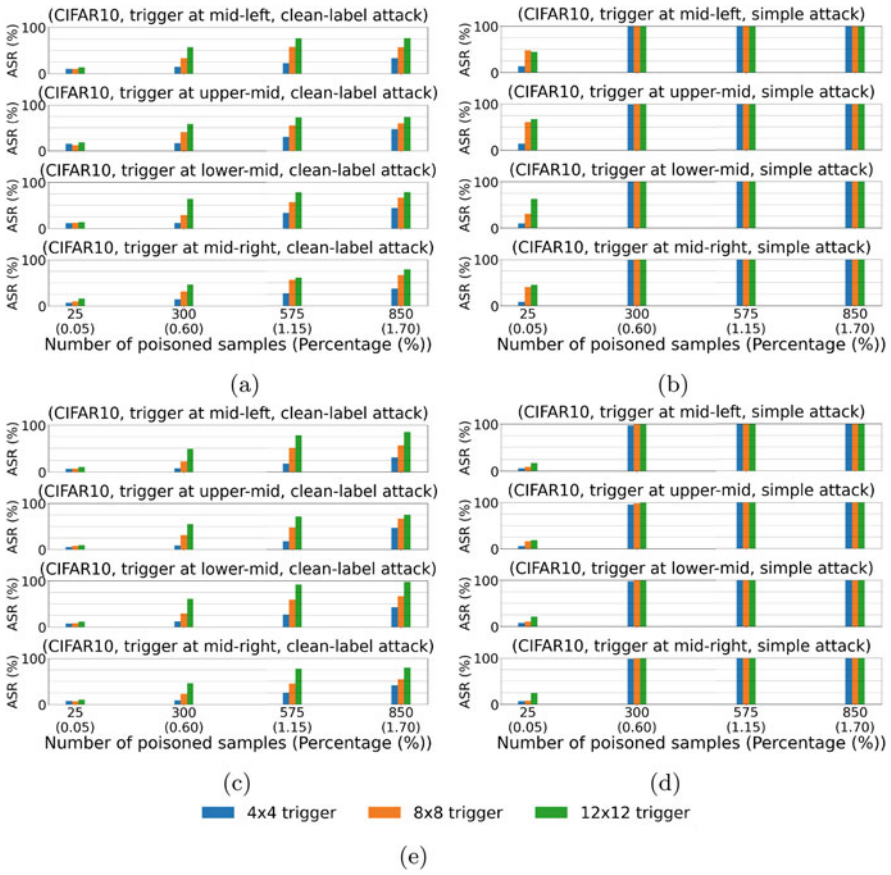


Fig. 5 Attack accuracy for the CIFAR10 dataset. The clean-label attack is significantly more effective than for FMNIST because the 3-channel trigger contains more information. We also see that the trigger position is not very important, and ASR increases as the trigger size increases. The ASR with the simple attack is 100% for a 0.6% poisoning rate or more. However, STRIPNet is not as vulnerable as ResNet due to its smaller capacity. (**a**) ResNet-9 + clean-label attack. (**b**) ResNet-9 + simple attack. (**c**) STRIPNet + clean-label attack. (**d**) STRIPNet + simple attack. (**e**) Legend

attack stealthy. This behavior is expected as we poison only a small subset of the training data that cannot substantially affect the model's learning.

In Figs. 6 and 7, we show the results of our experiments for the AG News topic classification dataset and IMDB dataset, respectively. From these figures, we can draw several conclusions. In most cases, the ASR is correlated with the trigger size and increases as the trigger size increases. This is true even when the attack is not effective (see Fig. 6a).

Table 3 Clean accuracy drop in text classification (AG News)

Model	Original Acc	Attack type	Number of poisoned samples			
			250	500	750	1000
CNN	90.74 (± 0.314)	Clean-label	90.48 (± 0.557)	90.52 (± 0.531)	90.52 (± 0.614)	90.51 (± 0.541)
		Simple	90.49 (± 0.534)	90.38 (± 0.593)	90.53 (± 0.484)	90.33 (± 0.760)
CNN + GloVe	89.78 (± 0.169)	Clean-label	89.72 (± 0.218)	89.71 (± 0.201)	89.71 (± 0.186)	89.70 (± 0.220)
		Simple	89.72 (± 0.209)	89.68 (± 0.206)	89.69 (± 0.227)	89.68 (± 0.202)

Table 4 Clean accuracy drop in text classification (IMDB)

Model	Original Acc	Attack type	Number of poisoned samples			
			50	100	150	200
CNN	87.05 (± 0.062)	Clean-label	86.91 (± 0.084)	86.93 (± 0.071)	86.91 (± 0.078)	86.90 (± 0.072)
		Simple	86.90 (± 0.083)	86.89 (± 0.089)	86.88 (± 0.090)	86.88 (± 0.096)
CNN + GloVe	84.20 (± 0.308)	Clean-label	83.73 (± 0.681)	83.71 (± 0.702)	83.62 (± 0.775)	83.77 (± 0.680)
		Simple	83.79 (± 0.535)	83.82 (± 0.519)	83.61 (± 0.570)	83.59 (± 0.724)

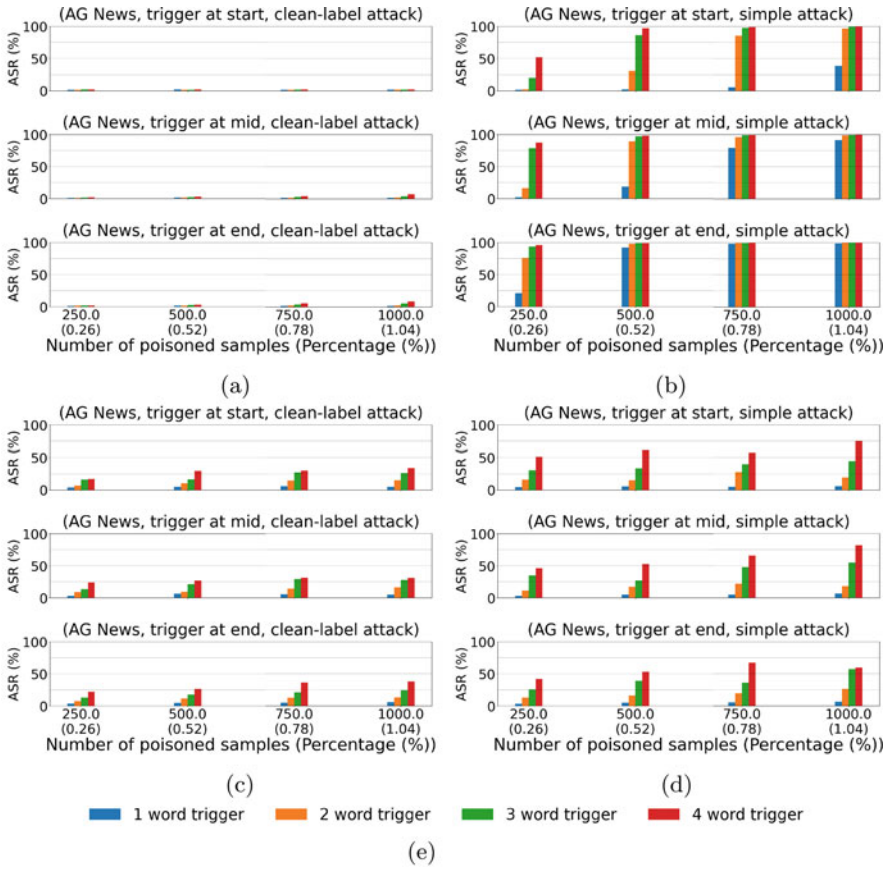


Fig. 6 Attack accuracy for the AG News dataset. The ASR is positively correlated with the trigger size (even when the ASR is very low), and the poisoning rate significantly influences the attack's effectiveness. Additionally, the clean-label attack needs more poisoned data to work. When GloVe is used, inserting the trigger in the end results in higher ASR (especially for low poisoning rates), but in the simple CNN, the trigger positions do not affect ASR. (**a**) CNN with GloVe + clean-label attack. (**b**) CNN with GloVe + simple attack. (**c**) CNN + clean-label attack. (**d**) CNN + simple attack. (**e**) Legend

Especially for the first CNN, this relation seems to be linear (see Figs. 6c, 6d, 7c and 7d). This simple model uses global average pooling as its penultimate layer, averaging the feature map before the output. As a result, the trigger will be more influential when it consists of more words. In almost all experiments, the poisoning rate is a highly influential hyperparameter of the backdoor attack, and any increase in it leads to an increase in the attack success rate.

Our models learn differently, which can be seen from the varying attack success rate when the trigger is injected in different positions. For example, the attack success rate is higher if the trigger is inserted at the end of the sentence when we use

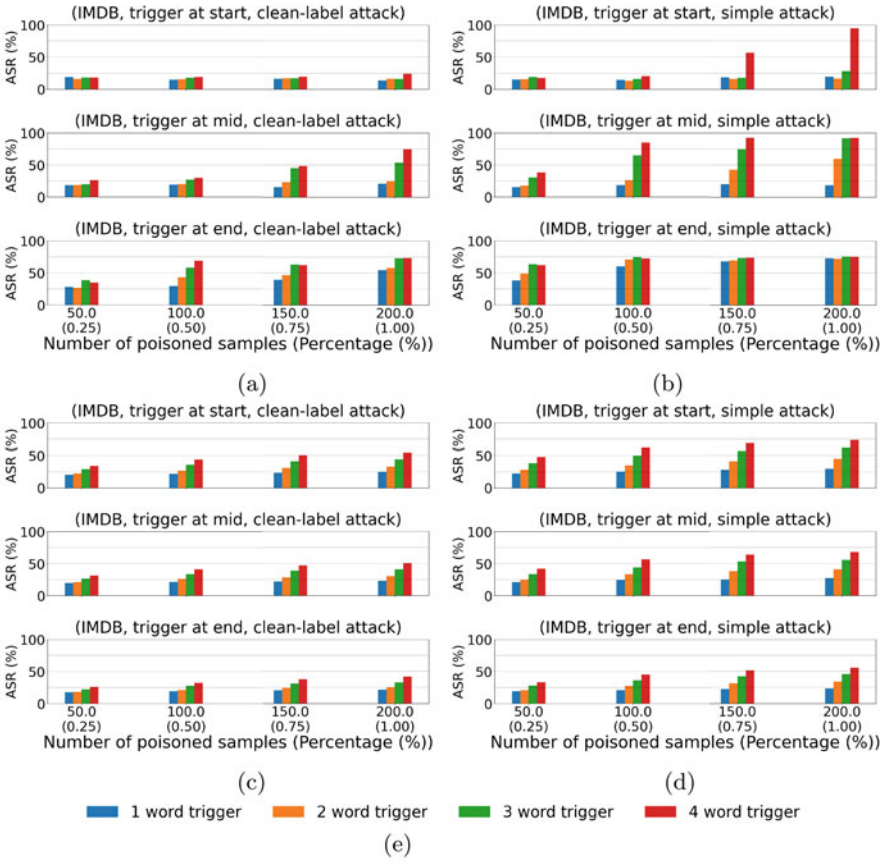


Fig. 7 Attack accuracy for the IMDB dataset. The ASR is positively correlated with the trigger size, and the poisoning rate significantly influences the attack's effectiveness. Additionally, the clean-label attack is more effective with this dataset. For the CNN that uses GloVe, placing the trigger at the end of the sentence yields the best results, but for the simple CNN, this is the least effective position. (**a**) CNN with GloVe + clean-label attack. (**b**) CNN with GloVe + simple attack. (**c**) CNN + clean-label attack. (**d**) CNN + simple attack. (**e**) Legend

the first model and the simple backdoor attack (see Figs. 6b and 7b). This difference is very clear for low poisoning rates (0.25%), where even a small trigger of 2 words could be substantially more effective when placed at the end of the sentence. On the other hand, for the other model, placing the trigger at the end does not result in higher ASR (see Figs. 6d and 7d). These differences indicate that we could use the backdoor attack as a tool for AI explainability and further understand what and how a model learns by using triggers with different characteristics.

In [47], the authors claimed that the clean-label backdoor attack needs a very large poisoning rate to be effective. We also see this behavior in the AG News dataset, especially for the architecture that uses the pretrained GloVe embedding

(Fig. 6a). In the other architecture, the clean-label attack is more effective as the feature space created by the trainable embedding encodes some information about the trigger and the target class (see Fig. 6c). However, when the IMDB dataset is used, both models perform similarly without poisoning a very large part of the training data for the clean-label (see Figs. 7a and 7c). This can be explained by the differences between the datasets. Each sentence in AG News is shorter than the movie reviews in the IMDB dataset. Additionally, most of the words in AG News are strongly connected with the topic that each sentence belongs to (world, sports, business, and science/technology), which is not true for the IMDB dataset. In the IMDB dataset, the sentences are longer, and usually, only a few words are related to their sentiment. As a result, in AG News, our attack needs more poisoned samples to overcome the effect of the original features of the source class.

4.3 Speech Recognition

In Tables 5 and 6, we compare the performance of clean and backdoored models for sound classification when clean inputs are used. As was also shown in [28], the differences between the clean and the backdoored models are negligible. In particular, the backdoored models perform a little better when the 10 classes dataset is used, meaning that the poisoned samples could serve as a generalization factor. However, when the full dataset is used, the backdoor insertion results in a small performance drop for the CNN. In this case, we use more classes, and the model has to learn a more difficult task that is affected even by a few poisoned samples. The performance of the LSTM is slightly increased, meaning that the LSTM builds different models and utilizes its capacity better when we use the full dataset [28]. All these differences are small, and our claims need additional experimental data to be confirmed.

In Figs. 8 and 9, we show the results of our experiments for the first (10 classes) and the second (30 classes) version of the Speech Commands dataset. In almost all cases, the attack success rate increases as the trigger duration increases. This is true even when the attack is not successful (see the clean-label attack in Figs. 8c and 9c). This makes sense as more input features are affected when a longer trigger is used, and the network can learn this relation easier. Additionally, the poisoning rate is very influential, and its increase leads to more effective backdoors.

The end of the input is the most effective trigger position for the LSTM network in both versions of the dataset. Even though this network uses two bidirectional LSTM layers and an attention layer, it seems to learn the features that are placed towards the end of its inputs more easily. The LSTM network was designed to tackle the problem of long-term dependencies on its inputs. A possible reason for this behavior is the nature of this particular dataset, which consists of 1-second clips of spoken words. If these words are not perfectly centered and distributed to the upper half of each sample, our network will give more attention to the end of each training sample. This is not true for the CNN used as, in that case, all the positions seem to

Table 5 Clean accuracy drop in sound classification (10 classes)

Model	Original Acc	Attack type	Number of poisoned samples			
			50	100	150	200
CNN	94.82 (± 0.360)	Clean-label	95.07 (± 0.437)	95.00 (± 0.477)	95.07 (± 0.427)	94.96 (± 0.508)
		Simple	95.09 (± 0.417)	95.04 (± 0.438)	94.99 (± 0.474)	94.95 (± 0.480)
LSTM	89.47 (± 1.412)	Clean-label	89.77 (± 1.426)	89.69 (± 1.350)	90.05 (± 1.362)	89.89 (± 1.373)
		Simple	89.71 (± 1.605)	89.96 (± 1.335)	89.66 (± 1.482)	89.80 (± 1.586)

Table 6 Clean accuracy drop in sound classification (30 classes)

Model	Original Acc	Attack type	Number of poisoned samples			
			137	274	411	547
CNN	94.70 (± 0.395)	Clean-label	94.61 (± 0.410)	94.55 (± 0.439)	94.49 (± 0.575)	94.59 (± 0.461)
		Simple	94.54 (± 0.471)	94.63 (± 0.511)	94.56 (± 0.496)	94.53 (± 0.477)
LSTM	90.50 (± 0.967)	Clean-label	90.88 (± 1.267)	90.59 (± 1.238)	90.82 (± 1.232)	90.79 (± 1.334)
		Simple	90.86 (± 1.236)	90.81 (± 1.167)	90.67 (± 1.317)	90.63 (± 1.278)

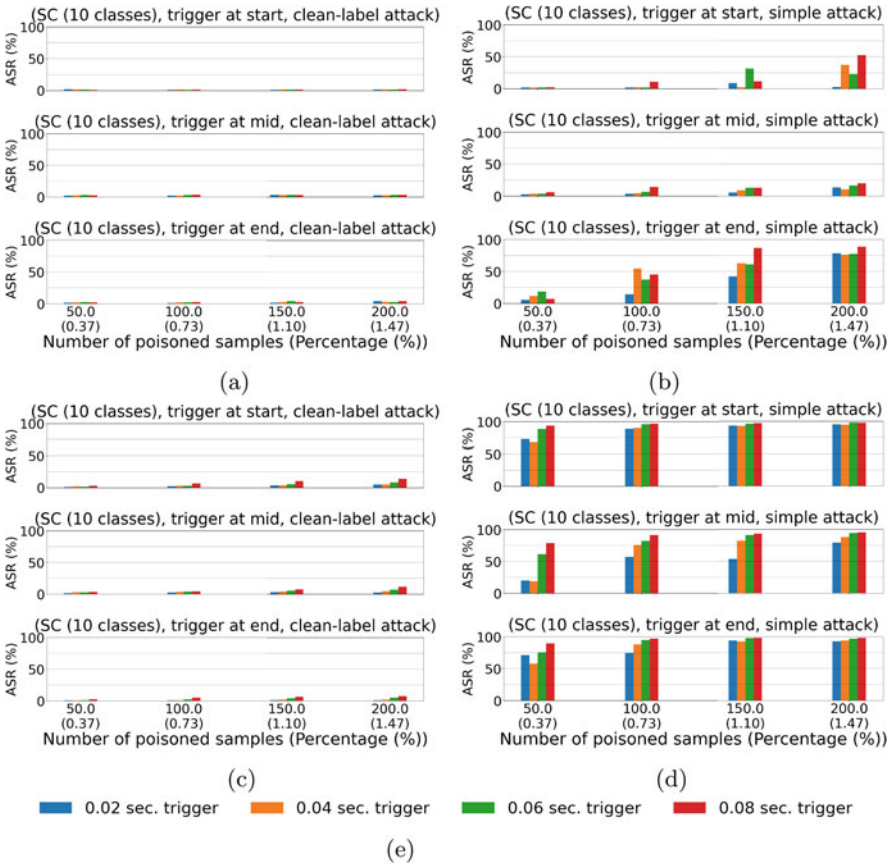


Fig. 8 Attack accuracy for the Speech Commands dataset (10 classes). In most cases, ASR is increased as the trigger duration or the poisoning rate increase. The clean-label attack is ineffective for both models. For LSTM, the best position for the trigger is at the end. However, for CNN, any position works. For CNN, the simple attack works almost perfectly for 100 poisoned samples or more. (**a**) LSTM + clean-label attack. (**b**) LSTM + simple attack. (**c**) CNN + clean-label attack. (**d**) CNN + simple attack. (**e**) Legend

be equally effective (see Figs. 8d and 9d). Similarly to text classification, different models learn different patterns from the same dataset making the backdoor attack effective in different cases. Thus, we could use the backdoor attack and its triggers to understand what a model learns and how it makes its decisions.

In our sound classification experiments, the clean-label attack is not successful for both neural networks and datasets. However, when the full dataset and CNN are used (Fig. 9c), the attack success rate slightly increases with large triggers. The clean-label could work without requiring more poisoned data if we choose a larger trigger. This claim, though, needs to be verified in the future with more experimental evidence. Another interesting observation is that the simple backdoor

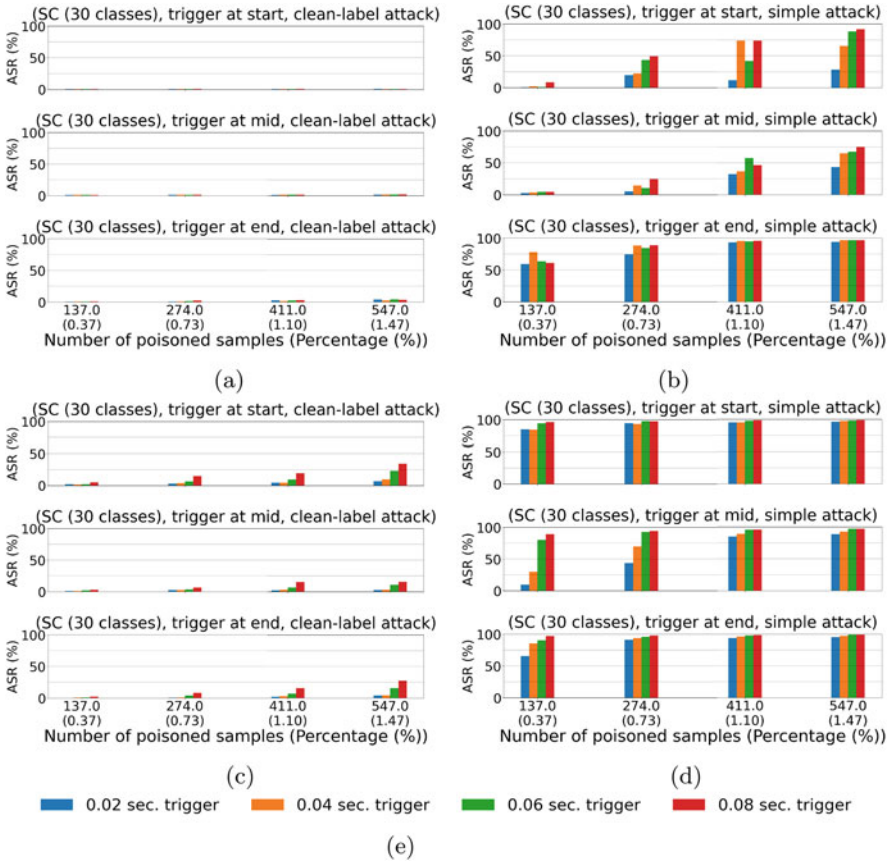


Fig. 9 Attack accuracy for the Speech Commands dataset (30 classes). In most cases, ASR is increased as the trigger duration or the poisoning rate increase. The clean-label attack is ineffective in that case, but it works slightly better for CNN. In the simple attack, the best position for the trigger is at the end for LSTM. However, there is no difference for CNN (in most cases, ASR is close to 100%). When using LSTM, ASR is higher than ASR for the ten classes. We assume that the absolute number of poisoned samples could be the reason behind that behavior. (**a**) LSTM + clean-label attack. (**b**) LSTM + simple attack. (**c**) CNN + clean-label attack. (**d**) CNN + simple attack. (**e**) Legend

attack becomes more effective when we use the full dataset and the LSTM network (compare Fig. 8b to Fig. 9b). One reason for this behavior is the absolute number of training samples that were increased when the full dataset was used. However, this should be investigated further.

4.4 Graph Data

Results for the Graph Classification Task For the graph classification task, two parameters affect the performance of the backdoor attack: poisoning intensity and trigger size (the number of nodes in the trigger graph). The attack results for the GCN model on AIDS with different poisoning intensity α and trigger size s are shown in Fig. 10. As we can see from Fig. 10a, with the increase of poisoning intensity, the attack success rate is generally increasing for each trigger size, but there is no obvious improvement between $\alpha = 0.15$ and $\alpha = 0.2$. Here, we select poisoning intensity $\alpha = 0.15$ for GCN on AIDS. Figure 10b shows the impact of trigger size under the selected poisoning intensity ($\alpha = 0.15$). The attack success rate is highest with $s = 5$, while the clean accuracy drop is the smallest when $s = 5$. To compare the two backdoor attacks, we set $\alpha = 0.15$, $s = 5$ and $\alpha = 0.2$, $s = 7$ for AIDS and TRIANGLES, respectively.

Specifically, we present the attack results of two backdoor attacks on the graph classification task in Tables 7 and 8. As we can see from Table 7, ATI can achieve more than 99% attack success rate and less than 1% clean accuracy drop on AIDS, while the performance of ATI degrades slightly with an attack success rate of more than 95% and clean accuracy drop around 1.5%. As illustrated in Table 8, the attack success rate of ATI is significantly higher than ATI for TRIANGLES, i.e., more than 10%. However, the clean accuracy drop of ATI is larger than ATI, which is more than 4% for both models, while that of ATI is around 3% and less than 1%

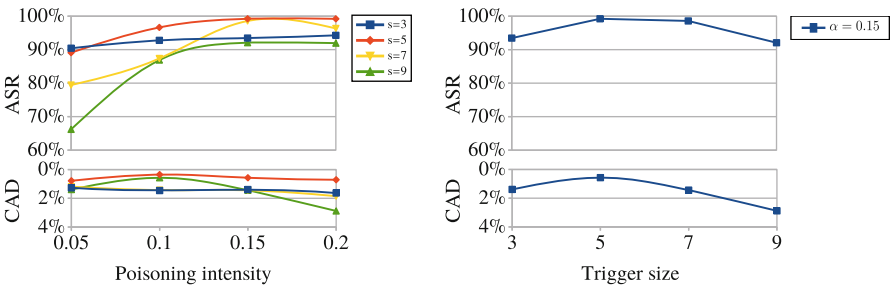


Fig. 10 Impact of poisoning intensity and trigger size on attack performance in the graph classification task. (**a**) GCN_AIDS. (**b**) GCN_AIDS ($\alpha = 0.15$)

Table 7 Backdoor attack results for the graph classification task and the AIDS dataset

	ATI		ATI	
Setting	ASR (%)	CAD (%)	ASR (%)	CAD (%)
GCN	95.86	1.25	99.92	0.46
GraphSAGE	97.59	1.46	99.80	0.91

Table 8 Backdoor attack results for the graph classification task and the TRIANGLES dataset

	AT^{I}		AT^{II}	
Setting	ASR (%)	CAD (%)	ASR (%)	CAD (%)
GCN	86.00	3.18	99.21	5.32
GraphSAGE	87.70	0.50	98.24	4.32

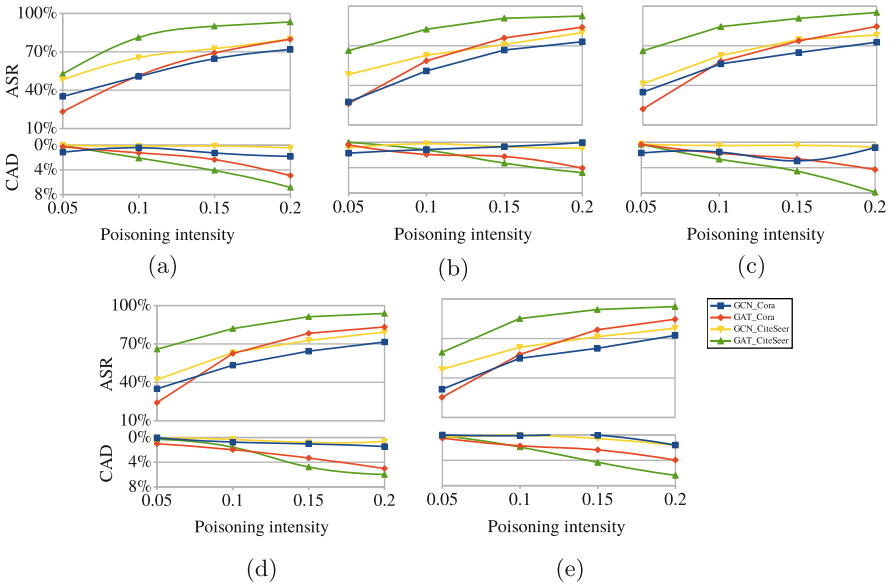


Fig. 11 Impact of poisoning intensity and feature trigger width on attack performance in the node classification task. (**a**) $n = 5$. (**b**) $n = 10$. (**c**) $n = 15$. (**d**) $n = 20$. (**e**) $n = 25$

for GCN and GraphSAGE, respectively. In addition, the computation time for AT^{II} is around 1.7 times of AT^{I}.

Results for the Node Classification Task For the node classification task, the backdoored data is influenced by two parameters: poisoning intensity α and feature trigger width n. The attack performance, including attack success rate and clean accuracy drop with different variants, is shown in Fig. 11. For each feature trigger width, the attack success rate on different models and datasets generally increases when the poisoning intensity increases from 0.05 to 0.2. At the same time, the clean accuracy drop of the GCN model keeps increasing, and there is a significant increase between $\alpha = 0.15$ and $\alpha = 0.2$. However, the clean accuracy drop of the GAT model remains almost unchanged. To achieve a high attack success rate and low clean accuracy drop, we set $\alpha = 0.2$ for GCN and $\alpha = 0.15$ for GAT. To evaluate the impact of feature trigger width on attack performance, we show the attack results with different feature trigger widths in Fig. 12. Observe that the feature trigger width

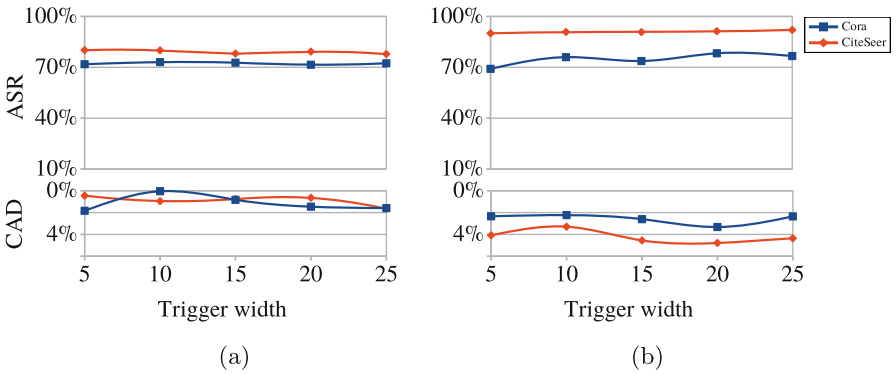


Fig. 12 Attack performance with different feature trigger widths. (**a**) GCN ($\alpha = 0.2$. (**b**) GAT ($\alpha = 0.15$

Table 9 Backdoor attack results in the node classification task ($n = 5$)

	GCN ($\alpha = 0.2$)		GAT ($\alpha = 0.15$)	
Setting	ASR (%)	CAD (%)	ASR (%)	CAD (%)
Cora	72.35	1.59	86.63	2.35
CiteSeer	77.82	1.63	92.04	1.35

has no obvious influence on the attack success rate and clean accuracy drop for both GNN models and datasets.

Specifically, Table 9 shows the attack success rate and clean accuracy drop of backdoor attack for the node classification task with selected parameters. Notice that the backdoor attack on GCN reaches over 70% attack success rate for both datasets and that on GAT obtains a higher attack success rate, i.e., over 85% and 90% for Cora and CiteSeer, respectively. Furthermore, the clean accuracy drop is lower than 2% for all models and datasets except for the GAT model on the Cora dataset, which is 2.35%.

4.5 General Observations

First, we verified that the backdoor attack is a real threat as it can be injected into every application domain tried without affecting the model's original task just by poisoning a small subset of the training data. Additionally, we saw that the poisoning rate is the most influential characteristic of the trigger in all applications. However, this value cannot be increased arbitrarily because the backdoor attack will become evident through a simple data filtering mechanism, and the poisoned model's performance on clean inputs will decrease substantially.

The trigger size is positively correlated with the backdoor's attack success rate in image, text, and sound. This is expected as a larger trigger contains more

information that can be encoded easier in the trained model. However, in graph classification, the attack success rate increased to a point ($s = 5$) and then decreased for larger triggers. The variations are small, though, as ASR remained above 90% in our experiments, and thus, we cannot draw general conclusions. We need to verify this effect with more complex datasets and models.

The most effective position of the trigger (if there is any) depends on many factors, like the network architecture or the dataset. The position is not very influential on the attack success rate in most cases, but this is not always true. Thus, we cannot draw any general conclusions. In image classification, no position was proven more effective as the convolutional layers extract information from any point in the image. Similar behavior has been observed in graph neural networks [57], where the trigger position did not result in more effective backdoors. On the other hand, in text classification, the attack performed similarly for all the trigger positions for the simple CNN, but the "end" was slightly more effective when the GloVe embedding was used. In sound classification, the trigger was more effective in the end if LSTM was used but had no difference for CNN. These differences suggest a potential beneficial use case for backdoor attacks in general. In this case, we can use them to understand better how and what our models learn. Such an approach complements the work described in [61], where the authors drew valuable insights about the input's crucial features after graying out small square areas of the input images.

The clean-label attack is challenging in image, text, and sound classification. However, in some cases, it may be successful just by using a large trigger without having to poison more data. Additionally, if the trigger encloses more information, the clean-label's performance can be improved. We verified this for the CIFAR10 dataset, where we injected our trigger in all three image channels. We believe that the dataset influences the performance of this attack. If each element contains many features, the model will require a large poisoning rate to perceive the trigger as a feature of this class. In the clean-label attack, the trigger is injected only in elements from the target class, and it is not easy to overcome the effect of the actual features of this class. This was highlighted in the AG News and IMDB datasets in text classification. On the other hand, the simple backdoor attack can be very effective with just a few poisoned samples in all the applications we tried.

As a general remark, we believe that the backdoor is easier inserted into models that can overfit small subsets of their datasets. Models with strong generalizations are more robust against data poisoning backdoor attacks. We verified this behavior using simple CNN in text classification. In that case, our attack could not reach an attack success rate larger than 80% even with the simple attack, as the model is very simple and the learned function is very smooth. Finally, as our experiments are far from exhaustive, our findings should be taken as indications, not definitive conclusions.

In summary, the key takeaways are:

- The backdoor attack is a realistic and stealthy threat.
- As expected, increasing the poisoning rate and using larger triggers leads to higher ASR.

– Different models can behave differently during the attack even though we use the same data. Therefore, we can use the backdoor attacks as a tool for explainable AI.
– In [47], the authors claimed that the clean-label attack is not very effective. In most cases, this is true, but we saw that we could make it effective with more sophisticated triggers.
– The backdoor is easier for models that can overfit a small subset of their datasets.

5 Conclusions

Recent trends in machine learning lead to novel attack vectors like the backdoor attack. This attack is very dangerous as it can compromise AI-powered systems. Naturally, the backdoor attack also attracted significant attention, resulting in numerous novel attack and defense versions. In this work, we explored the effects of various trigger characteristics on the backdoor's performance in four domains. Our results show that deploying backdoor attacks is relatively easy for all investigated domains. There are sufficient commonalities between the attacks in different domains to ease their deployment in real-world applications and devise novel, more generic defenses.

References

1. Ardila, R., Branson, M., Davis, K., Henretty, M., Kohler, M., Meyer, J., Morais, R., Saunders, L., Tyers, F.M., Weber, G.: Common voice: a massively-multilingual speech corpus (2019). http://arxiv.org/abs/1912.06670
2. Bagdasaryan, E., Shmatikov, V.: Blind backdoors in deep learning models. In: 30th USENIX Security Symposium (USENIX Security 21), pp. 1505–1521. USENIX Association (2021). https://www.usenix.org/conference/usenixsecurity21/presentation/bagdasaryan
3. Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D., Shmatikov, V.: How to backdoor federated learning. In: International Conference on Artificial Intelligence and Statistics, pp. 2938–2948. PMLR (2020)
4. Chen, B., Carvalho, W., Baracaldo, N., Ludwig, H., Edwards, B., Lee, T., Molloy, I., Srivastava, B.: Detecting backdoor attacks on deep neural networks by activation clustering (2018). arXiv preprint arXiv:1811.03728
5. Chen, X., Salem, A., Chen, D., Backes, M., Ma, S., Shen, Q., Wu, Z., Zhang, Y.: BadNL: Backdoor attacks against NLP models with semantic-preserving improvements. In: Annual Computer Security Applications Conference, pp. 554–569 (2021)
6. Chen, X., Liu, C., Li, B., Lu, K., Song, D.: targeted backdoor attacks on deep learning systems using data poisoning (2017). arXiv preprint arXiv:1712.05526
7. Costales, R., Mao, C., Norwitz, R., Kim, B., Yang, J.: Live trojan attacks on deep neural networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, pp. 796–797 (2020)

8. Dahl, G.E., Stokes, J.W., Deng, L., Yu, D.: Large-scale malware classification using random projections and neural networks. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 3422–3426. IEEE (2013)
9. Dai, J., Chen, C., Li, Y.: A backdoor attack against LSTM-based text classification systems. IEEE Access **7**, 138872–138878 (2019)
10. de Andrade, D.C., Leo, S., Viana, M.L.D.S., Bernkopf, C.: A neural attention model for speech command recognition (2018)
11. Deng, J., Dong, W., Socher, R., Li, L., Kai Li, Li Fei-Fei: ImageNet: a large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition, pp. 248–255 (2009). https://doi.org/10.1109/CVPR.2009.5206848
12. Dikmen, M., Burns, C.M.: Autonomous driving in the real world: experiences with tesla autopilot and summon. In: Proceedings of the 8th International Conference on Automotive User Interfaces and Interactive Vehicular Applications, pp. 225–228 (2016)
13. Dodge, S., Karam, L.: A study and comparison of human and deep learning recognition performance under visual distortions. In: 2017 26th International Conference on Computer Communication and Networks (ICCCN), pp. 1–7. IEEE (2017)
14. Franceschi, L., Frasconi, P., Salzo, S., Grazzi, R., Pontil, M.: Bilevel programming for hyperparameter optimization and meta-learning. In: International Conference on Machine Learning, pp. 1568–1577. PMLR (2018)
15. Gao, Y., Doan, B.G., Zhang, Z., Ma, S., Zhang, J., Fu, A., Nepal, S., Kim, H.: Backdoor attacks and countermeasures on deep learning: a comprehensive review (2020). arXiv preprint arXiv:2007.10760
16. Gao, Y., Xu, C., Wang, D., Chen, S., Ranasinghe, D.C., Nepal, S.: Strip: a defence against trojan attacks on deep neural networks. In: Proceedings of the 35th Annual Computer Security Applications Conference, pp. 113–125 (2019)
17. Gilbert, E.N.: Random graphs. The Annals of Mathematical Statistics **30**(4), 1141–1144 (1959). https://doi.org/10.1214/aoms/1177706098
18. Graves, A., Mohamed, A.r., Hinton, G.: Speech recognition with deep recurrent neural networks. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 6645–6649. IEEE (2013)
19. Gu, T., Liu, K., Dolan-Gavitt, B., Garg, S.: BadNets: Evaluating backdooring attacks on deep neural networks. IEEE Access **7**, 47230–47244 (2019). https://doi.org/10.1109/ACCESS.2019.2909068
20. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Advances in Neural Information Processing Systems, vol. 30 (2017)
21. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
22. Hong, S., Carlini, N., Kurakin, A.: Handcrafted backdoors in deep neural networks (2021). arXiv preprint arXiv:2106.04690
23. IBM: Natural language processing (2021). https://www.ibm.com/cloud/learn/natural-language-processing. Accessed 27 July 2022
24. Karlsen, S.S.: Automated Front Detection-Using computer vision and machine learning to explore a new direction in automated weather forecasting. Master's Thesis, The University of Bergen (2017)
25. Khan, A.I., Al-Habsi, S.: Machine learning in computer vision. Proc. Comput. Sci. **167**, 1444–1451 (2020)
26. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations (ICLR) (2017)
27. Koffas, S., Picek, S., Conti, M.: Dynamic backdoors with global average pooling (2022). arXiv preprint arXiv:2203.02079
28. Koffas, S., Xu, J., Conti, M., Picek, S.: Can you hear it? backdoor attacks via ultrasonic triggers. In: Proceedings of the 2022 ACM Workshop on Wireless Security and Machine Learning, pp. 57–62. WiseML '22, Association for Computing Machinery, New York (2022). https://doi.org/10.1145/3522783.3529523

29. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. Commun. ACM **60**(6), 84–90 (2017). https://doi.org/10.1145/3065386
30. Li, S., Xue, M., Zhao, B.Z.H., Zhu, H., Zhang, X.: Invisible backdoor attacks on deep neural networks via steganography and regularization. IEEE Trans. Depend. Secure Comput. **18**(5), 2088–2105 (2020)
31. Li, Y., Jiang, Y., Li, Z., Xia, S.T.: Backdoor learning: a survey. IEEE Transactions on Neural Networks and Learning Systems (2022)
32. Liu, Y., Ma, S., Aafer, Y., Lee, W.C., Zhai, J., Wang, W., Zhang, X.: Trojaning attack on neural networks. In: NDSS (2018)
33. Maas, A.L., Daly, R.E., Pham, P.T., Huang, D., Ng, A.Y., Potts, C.: Learning word vectors for sentiment analysis. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, pp. 142–150. Association for Computational Linguistics, Portland, Oregon, USA (2011). http://www.aclweb.org/anthology/P11-1015
34. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space (2013). arXiv preprint arXiv:1301.3781
35. Morris, C., Kriege, N.M., Bause, F., Kersting, K., Mutzel, P., Neumann, M.: TUDataset: A collection of benchmark datasets for learning with graphs. In: ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020) (2020). www.graphlearning.io
36. Mubin, N.A., Nadarajoo, E., Shafri, H.Z.M., Hamedianfar, A.: Young and mature oil palm tree detection and counting using convolutional neural network deep learning method. International J. Remote Sensing **40**(19), 7500–7515 (2019)
37. Nelson, B., Barreno, M., Jack Chi, F., Joseph, A.D., Rubinstein, B.I.P., Saini, U., Sutton, C., Tygar, J.D., Xia, K.: Misleading Learners: Co-Opting Your Spam Filter, pp. 17–51. Springer US, Boston, MA (2009). https://doi.org/10.1007/978-0-387-88735-7_2
38. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: Empirical Methods in Natural Language Processing (EMNLP), pp. 1532–1543 (2014). http://www.aclweb.org/anthology/D14-1162
39. Prabhu, V.U., Birhane, A.: Large image datasets: a pyrrhic win for computer vision? CoRR abs/2006.16923 (2020). https://arxiv.org/abs/2006.16923
40. Rijsdijk, J., Wu, L., Perin, G., Picek, S.: Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. IACR Trans. Cryptograp. Hardw. Embedd. Syst. **2021**(3), 677–707 (2021). https://doi.org/10.46586/tches.v2021.i3.677-707
41. Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., Eliassi-Rad, T.: Collective classification in network data. AI Mag. **29**(3), 93–93 (2008)
42. Severi, G., Meyer, J., Coull, S., Oprea, A.: Explanation-Guided backdoor poisoning attacks against malware classifiers. In: 30th USENIX Security Symposium (USENIX Security 21), pp. 1487–1504. USENIX Association (2021). https://www.usenix.org/conference/usenixsecurity21/presentation/severi
43. Shokri, R., et al.: Bypassing backdoor detection algorithms in deep learning. In: 2020 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 175–183. IEEE (2020)
44. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)
45. Sun, Z., Kairouz, P., Suresh, A.T., McMahan, H.B.: Can you really backdoor federated learning? (2019). arXiv preprint arXiv:1911.07963
46. Trigueiros, P., Ribeiro, F., Reis, L.P.: Hand gesture recognition system based in computer vision and machine learning. In: Developments in Medical Image Processing and Computational Vision, pp. 355–377. Springer, Berlin (2015)
47. Turner, A., Tsipras, D., Madry, A.: Label-consistent backdoor attacks (2019). arXiv preprint arXiv:1912.02771
48. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph Attention Networks. International Conference on Learning Representations (2018). https://openreview.net/forum?id=rJXMpikCZ. Accepted as poster

49. Vinyes Mora, S.: Computer vision and machine learning for in-play tennis analysis: framework, algorithms and implementation. Ph.D. Thesis, Imperial College London (2018)
50. Wang, B., Yao, Y., Shan, S., Li, H., Viswanath, B., Zheng, H., Zhao, B.Y.: Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In: 2019 IEEE Symposium on Security and Privacy (SP), pp. 707–723. IEEE (2019)
51. Wang, H., Mazari, M., Pourhomayoun, M., Smith, J., Owens, H., Chernicoff, W.: An end-to-end traffic vision and counting system using computer vision and machine learning: the challenges in real-time processing. SIGNAL 2018 Editors, p. 13 (2018)
52. Wenger, E., Passananti, J., Bhagoji, A.N., Yao, Y., Zheng, H., Zhao, B.Y.: Backdoor attacks against deep learning systems in the physical world. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 6206–6215 (2021)
53. Wiley, V., Lucas, T.: Computer vision and image processing: a paper review. Int. J. Artif. Intell. Res. **2**(1), 29–36 (2018)
54. Wu, Y., Schuster, M., Chen, Z., Le, Q.V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al.: Google's neural machine translation system: bridging the gap between human and machine translation (2016). arXiv preprint arXiv:1609.08144
55. Xi, Z., Pang, R., Ji, S., Wang, T.: Graph backdoor. In: 30th USENIX Security Symposium (USENIX Security 21), pp. 1523–1540 (2021)
56. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms (2017)
57. Xu, J., Xue, M., Picek, S.: Explainability-based backdoor attacks against graph neural networks. In: Proceedings of the 3rd ACM Workshop on Wireless Security and Machine Learning, pp. 31–36 (2021)
58. Yang, Z., Iyer, N., Reimann, J., Virani, N.: Design of intentional backdoors in sequential models (2019). arXiv preprint arXiv:1902.09972
59. Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., Leskovec, J.: Hierarchical graph representation learning with differentiable pooling. In: Advances in Neural Information Processing Systems, vol. 31 (2018)
60. Yunchao, G., Jiayao, Y.: Application of computer vision and deep learning in breast cancer assisted diagnosis. In: Proceedings of the 3rd International Conference on Machine Learning and Soft Computing, pp. 186–191 (2019)
61. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: European Conference on Computer Vision, pp. 818–833. Springer, Berlin (2014)
62. Zhai, T., Li, Y., Zhang, Z., Wu, B., Jiang, Y., Xia, S.T.: Backdoor attack against speaker verification. In: ICASSP 2021–2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 2560–2564. IEEE (2021)
63. Zhang, M., Cui, Z., Neumann, M., Chen, Y.: An end-to-end deep learning architecture for graph classification. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
64. Zhang, X., Zhao, J., LeCun, Y.: Character-level convolutional networks for text classification. In: Advances in Neural Information Processing Systems, vol. 28 (2015)
65. Zhang, Z., Jia, J., Wang, B., Gong, N.Z.: Backdoor attacks to graph neural networks. In: Proceedings of the 26th ACM Symposium on Access Control Models and Technologies, pp. 15–26 (2021)

Deep Learning Reliability: Towards Mitigating Reliability Threats in Deep Learning Systems by Exploiting Intrinsic Characteristics of DNNs

Muhammad Abdullah Hanif and Muhammad Shafique

1 Introduction

Deep Neural Networks (DNNs) have emerged as a promising set of algorithms for solving complex AI problems such as image classification, object detection and localization, semantic segmentation, speech recognition, language translation, and video processing [1]. The state-of-the-art performance of these models has laid the foundation for DNNs to be used in safety-critical applications as well such as autonomous driving [2] and smart healthcare [3]. Driven by the compute- and memory-intensive nature of DNNs and the need for deploying such high-accuracy models in resource-constrained edge devices, a significant amount of research has been carried out towards designing specialized hardware accellerators that can enable low-cost DNN inference at the edge. Some prominent DNN hardware accelerators include Eyeriss [4], MAERI [5], TPU [6], and MPNA [7].

On the one end, these DNN hardware accelerators promise low-cost and real-time execution of DNNs; however, on the other end, they bring some critical reliability challenges that can significantly degrade the performance and dependability of the system. These reliability threats are specifically important to address for safety-critical systems, as even a single fault at a critical location in such systems can result in severe consequences. For example, in the case of autonomous driving vehicles, a critical fault in the perception unit can result in the misclassification of a traffic sign (e.g., a stop sign) which can lead to a fatal accident. Such faults can even lead to total disruption of traffic service if the vehicle is connected to the traffic infrastructure in a smart city. An overview of different hardware-induced reliability threats, how they

M. A. Hanif (✉) · M. Shafique
New York University Abu Dhabi, Abu Dhabi, UAE
e-mail: mh6117@nyu.edu; muhammad.shafique@nyu.edu

S. Pasricha, M. Shafique (eds.), *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing*, https://doi.org/10.1007/978-3-031-40677-5_21

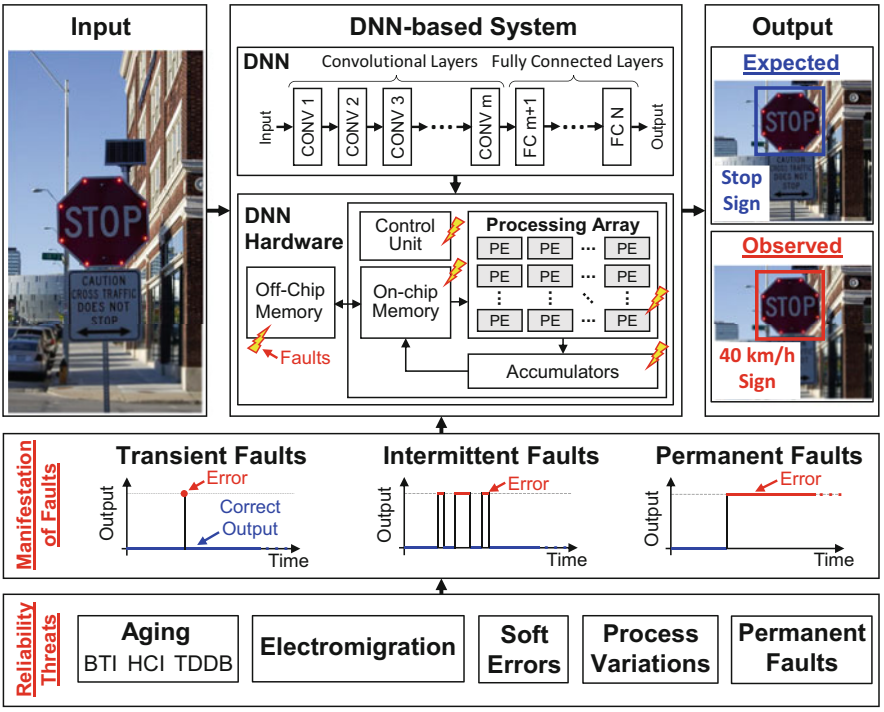


Fig. 1 Overview of different reliability threats and their repercussions. The picture used in the figure is from the COCO dataset

manifest in a system and how can they impact the functionality of a DNN inference is shown in Fig. 1.

Reliability Threats Gradual progress in the fabrication process and the desire for extreme-performance devices has lead us to the era of nano-scale devices. However, electronic devices fabricated using nano-scale technology face various reliability issues. Some of these issues are associated with the limitations of the fabrication process while others are associated with the extreme sizes of the transistors. The following text provides a brief introduction to different hardware-induced reliability threats that can degrade the performance of a system.

- **Soft Errors** are transient bit-flips caused by high-energy particle strikes. These particles can be alpha particles emitted from the impurities in the packaging materials of the chip or neutrons from cosmic radiations [8]. These soft errors can propagate all the way to application layer of a system and results in significant accuracy degradation. External factors such as temperature and altitude can have a notable impact on the Soft Error Rate (SER).
- **Aging** of nano-scale electronic devices occurs due to various physical phenomena such as Bias Temperature Instability (BTI), Time-Dependent Dielectric

Breakdown (TDDB), Hot Carrier Injection (HCI), and Electromigration (EM). It typically results in increased threshold voltage (V_{TH}) [9] or breakdown of dielectric and wires. In the early stages, aging manifests as timing errors in a system, and later it can even transform into permanent faults. Typically, large guardbands are added to the operating frequency of a circuit to ensure reliable operation. Similar to other reliability threats, aging rate also increases with temperature.

- **Process Variations** are variations in the hardware characteristics of transistors that occur due to imperfections in the fabrication process [10]. In general, these variations manifest as timing errors in a system and, therefore, are addressed by adding guardbands, i.e., either by increasing the supply voltage or reducing the operating frequency of the device. Extreme variations can even lead to permanent faults, which can have a significant impact on the manufacturing yield.

Apart from aggressive guard-banding, a number of other techniques have also been proposed to improve the resilience of systems against reliability threats. However, most of these techniques are based on redundancy, e.g., Error Correction Codes (ECC), Dual Modular Redundancy (DMR) [11], and Triple Modular Redundancy (TMR) [12]. The redundancy-based techniques, on the one hand, are highly effective; however, on the other hand, they lead to high performance and energy overheads. This together with the compute- and memory-intensive nature of DNNs makes such techniques infeasible for DNN-based systems. *Therefore, alternate techniques are required that can improve the resilience of DNN-based systems against hardware-induced reliability threats at minimal cost.*

In the following section, we present a brief *overview of DNNs and DNN hardware accelerators*. Then, in Sect. 3, we present an overall *methodology for building reliable DNN inference systems* and also discuss individual *low-cost techniques for mitigating permanent faults, aging, and soft errors*.

2 Preliminaries

2.1 Deep Neural Networks

A neural network can be visualized as a network of interconnected neurons (see Fig. 2a), where a neuron is the fundamental computational unit of the network. The functionality of a typical neuron used in neural networks can be described using the following equation:

$$Out = f\left(\sum_{i=0}^{N} W_i \times A_i + b\right) \tag{1}$$

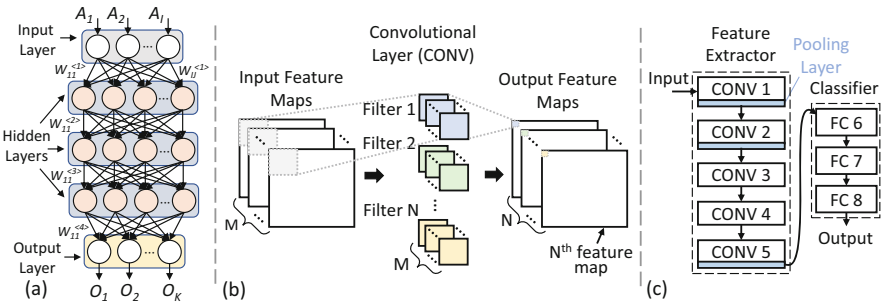


Fig. 2 (**a**) An example MLP network. (**b**) Convolutional layer. (**c**) Architecture of a convolutional neural network

where W_i represents the ith weight, A_i represents the ith activation, N represents the number of weights (and activations) in the input vectors, b represents the bias, and $f(.)$ represents the activation function (a non-linear function to introduce non-linearity). The neurons are typically arranged in layers, and a neural network having more than three layers is termed as a Deep Neural Network (DNN). Figure 2a shows an example of a Fully Connected Neural Network (FCNN), in which all the neurons in each layer are connected with all the neurons in the previous layer and the next layer. A FCNN is also known as a Multi-layer Perceptron (MLP).

Different types of DNNs have been proposed in the literature. Apart from FCNNs, Convolutional Neural Networks (CNNs) are also widely known. CNNs are mainly used to process spatially or temporally correlated data such as images and videos. A CNN is generally composed of multiple convolutional layers and fully connected layers, see Fig. 2c. The convolutional layers are used for extracting features from the input while the fully connected layers are responsible for the final classification based on the features extracted by the convolutional layers. Figure 2b shows a detailed view of a convolutional layer.

Various other types of DNNs also exist, for example, Recurrent Neural Networks (RNNs), Generative Adversarial Networks (GANs), and Graph Neural Networks (GNNs). However, as most of the reliability studies have been demonstrated on MLPs and CNNs, this chapter also considers the same to highlight the effectiveness of different methods.

2.2 DNN Hardware Accelerators

To enable the deployment of DNNs in resource-constrained edge devices, DNN hardware accelerators are employed. Various accelerator designs have been proposed in the literature, where each design supports a specific set of dataflows in

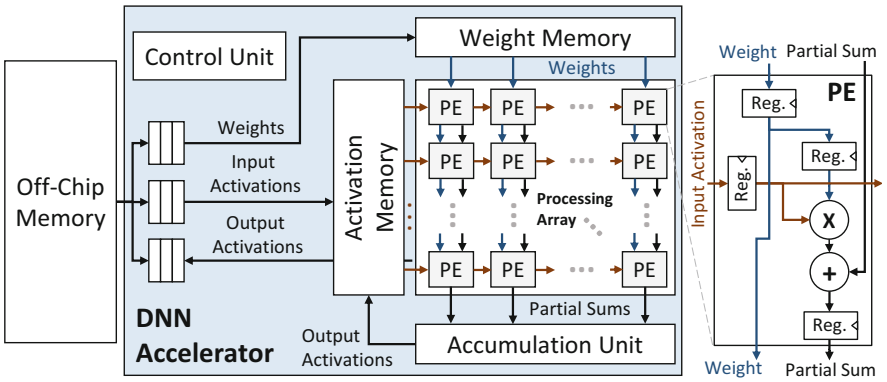


Fig. 3 A systolic-array-based DNN hardware accelerator

a more efficient manner, e.g., see [4, 6, 7, 13]. An overview of a DNN hardware accelerator is shown in Fig. 3. A DNN accelerator is mainly composed of Processing Elements (PEs), partial sum accumulation units, and on-chip memory. Each PE contains some arithmetic modules and some registers. The arithmetic modules are for performing the MAC operations involved in the DNN execution and the registers are for storing weights, activations, and partial sums. The exact configuration of the PEs and their connectivity in the accelerator are based on the supported dataflows.

The accelerator shown in Fig. 3 is a systolic-array-based design composed of homogeneous PEs, similar to the design in [6]. The PEs are connected in a 2D-grid-like manner. The accelerator follows a weight-stationary dataflow, where weights are loaded into the array (through vertical channels) and kept stationary during operations. Note, weights from the same filter/neuron are mapped on the same column, while weights from multiple filters/neurons can be mapped simultaneously by mapping different filters/neurons (from the same DNN layer) to different columns. After the weights have been loaded inside the PEs, the input activations are fed from the left and are multiplied with the weight values to generate products. Each product is then added with the partial sum from the above PE, and the updated partial sum is then passed downstream to be used in the next cycle by the downstream PE. As the size of the processing array is usually limited, a dot-product operation is broken down into multiple chunks (based on the size of the processing array) and a single chunk is mapped onto the array at a time. The partial sums generated by the array are stored in the accumulation units to be added with the corresponding partial sum/s from the other chunks (if any). A more detailed explanation of the architecture can be found in [14].

3 Reliable Deep Learning

This section presents a systematic methodology for building reliable systems for DNN-based applications. The section also highlights the impact of different types of reliability threats on the application-level accuracy of DNNs and presents different low-cost techniques for improving the resilience of DNN-based systems against hardware-induced reliability threats at minimal cost.

3.1 A Systematic Methodology for Building Reliable DNN Systems

Figure 4 presents an overview of our systematic design methodology for building reliable systems for DNN-based applications. The methodology is composed of different design-time steps, post-fabrication steps, and run-time steps.

The **design-time** steps focus on building a hardware accelerator capable of mitigating all types of hardware-induced reliability threats. Towards this, first, a baseline hardware accelerator is designed based on the user-defined performance constraints. Then, the additional circuitry required for mitigating permanent faults (see Sect. 3.3), aging (see Sect. 3.4), and soft errors (see Sect. 3.5) is added to the accelerator design. Note, to achieve high resilience against reliability threats at a low cost, the error resilience of a representative set of DNNs is taken into consideration. The error resilience helps estimate the extent of protection required against each threat. After reinforcing the accelerator with the additional circuitry, the hardware is synthesized using reliability-aware synthesis techniques, for example, using selective hardening where vulnerable nodes in the hardware are hardened using node-level redundancy [15].

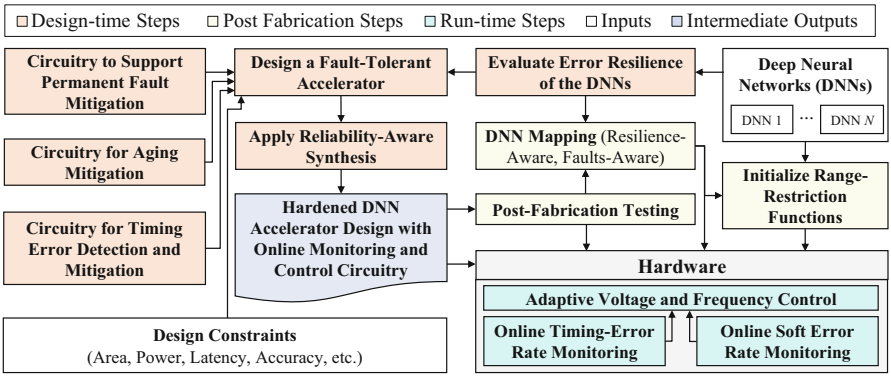


Fig. 4 Our methodology for designing reliable hardware for DNN-based applications (adapted from [16])

The **post-fabrication** steps focus on exploiting the information collected through post-fabrication testing (e.g., fault maps and process variation maps of the fabricated hardware) to define DNN mapping policies. The fault-aware and variation-aware mapping of DNNs can significantly reduce the negative impact of faults and variations on the application-level accuracy and performance characteristics of the system (see Sect. 3.3). The mapping information together with fault maps are also used by range initialization block for soft error mitigation.

The **run-time** steps focus on trading energy for reliability through adaptive voltage and frequency scaling. The online error rate monitoring blocks monitor the frequency of errors, and the system then responds by increasing the supply voltage or decreasing the operating frequency to reduce the frequency of errors, whenever required. Software-level redundancy can also be employed to improve the reliability by processing critical layers (or neurons) multiple times.

3.2 Resilience of DNNs to Reliability Threats

Occasionally, DNNs are assumed to be inherently resilient to errors [17]. However, studies have shown that DNNs respond differently to different types of errors. Errors that occur at critical locations in the system can significantly degrade the application-level accuracy of DNNs while errors at non-critical locations do not impact the accuracy much. This section presents the resilience of DNNs to different types of reliability threats. The section also highlights the importance of low-cost fault-mitigation techniques for dependable performance.

3.2.1 Resilience of DNNs to Permanent Faults

This section presents an empirical analysis from [14] highlighting the impact of stuck-at permanent faults in the computational array of a systolic-array-based DNN accelerator (shown in Fig. 3) on the application-level accuracy of DNNs. The analysis is performed for two different networks trained on two different datasets, i.e., the MNIST and TIMIT datasets. The details of the DNN architectures used are presented in Table 1. For this analysis, a systolic array of 256 × 256 MAC units synthesized using 45nm OSU PDK to generate a gate-level netlist is considered. For permanent faults, stuck-at faults are inserted randomly at internal nodes in the netlist.

Table 1 Datasets and the corresponding DNNs used for analyzing the impact of permanent faults

Dataset	Network architecture	Accuracy (%)
MNIST [18]	Fully connected (L1-L4): 784×256×256×256×10	98.15
TIMIT [19]	Fully connected (L1-L4): 1845×2000×2000×2000×183	73.91

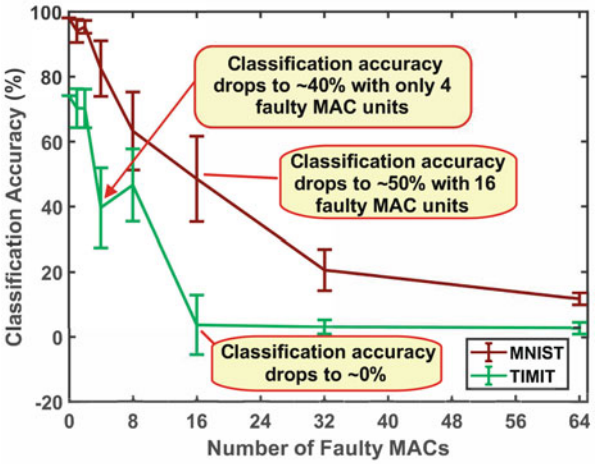


Fig. 5 Impact of Stuck-at-Faults in a systolic-array-based DNN accelerator on the classification accuracy of two different DNNs [20]

Figure 5a shows the impact of using a faulty DNN accelerator on the classification accuracy of two different DNNs. As can be observed from the figure, for both the DNNs, the classification accuracy decreases sharply with the increase in faulty PEs. For example, in the case with the number of faulty PEs equals 16, the average accuracy of the DNN trained on the TIMIT dataset falls to zero, while the accuracy of the DNN trained on the MNIST dataset falls to around 50%. Note that 16 PEs is equivalent to around 0.025% of total PEs in a 256×256 array. This shows that even a very small number of permanent faults in a DNN-based system can significantly degrade the system's performance. This analysis clearly highlights the need for permanent fault mitigation to increase the manufacturing yield of DNN accelerators, as faulty hardware having permanent faults cannot produce reliable results.

3.2.2 Resilience of DNNs to Timing Faults

Timing failures in high-performance nano-scale CMOS devices are a significant reliability concern. These errors arise due to various reasons, e.g., power supply disturbance, crosstalk, process variations, and aging. Moreover, the operating conditions, such as supply voltage, also significantly affect the frequency of timing failures. This section highlights the impact of timing errors on the classification accuracy of a DNN trained on the MNIST dataset using the analysis presented in [14]. The DNN architecture is presented in Table 1 and the considered hardware accelerator is shown in Fig. 3. To illustrate the impact of timing errors on DNN accuracy, [14] considered a Timing Error Propagation (TEP) case where timing errors are allowed to propagate to the output. The timing errors are introduced in

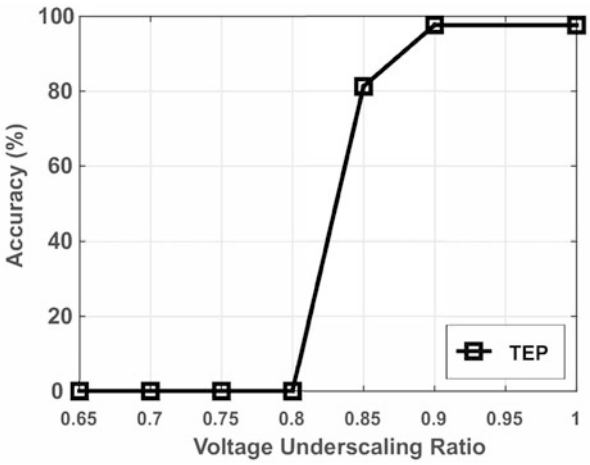


Fig. 6 Impact of timing errors induced through voltage under-scaling on the classification accuracy of a DNN trained on the MNIST dataset [14]

the accelerator array through voltage under-scaling. Figure 6 shows the impact of voltage under-scaling on the classification accuracy of the DNN. Note, as the supply voltage of the array is reduced, timing errors start increasing. Figure 6 clearly shows that in the TEP case, the accuracy of the DNN starts decreasing abruptly as the fault rate starts increasing. Therefore, to ensure reliable execution of DNNs it is essential to mitigate timing errors.

3.2.3 Resilience of DNNs to Memory Faults

To illustrate the impact of memory faults on the accuracy of DNNs, Hanif et al. [16] presented an analysis where they injected random faults in the weight memory of a DNN accelerator. The analysis showed that faults at higher significance bit-locations in the weights can drastically reduce the application-level accuracy of DNNs while faults at lower significance bit-locations do not impact the accuracy much. Moreover, the analysis also showed that the accuracy drop increases sharply with the increase in the fault rate. They also studied the impact of different types of faults individually and showed that 0-to-1 bit-flips have a more severe impact compared to 1-to-0 bit-flips, as 0-to-1 bit-flips at higher significance bit-locations can significantly increase the weight values. This conclusion is also in line with the dropout [21] and DropConnect [22] concepts in the sense that 1-to-0 bit-flips push the weight values toward zero, which is equivalent to dropout at a fine-grained level. Note that the conclusion may differ for different data representation formats. Similar fault injection studies have also been conducted in [23] and [24] to analyze the resilience of DNNs.

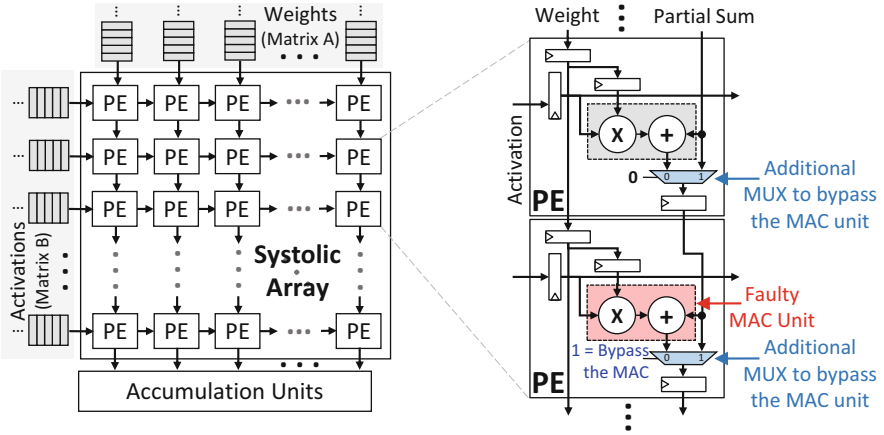


Fig. 7 Modified systolic-array design for permanent fault mitigation through fault-aware pruning

3.3 Permanent Fault Mitigation

As highlighted in Sect. 3.2.1, permanent faults can restrain a system from generating correct outputs. Therefore, it is essential to mitigate such errors to ensure high manufacturing yield. **Fault-Aware Pruning (FAP)** [20] has been proposed to mitigate permanent faults in the computational array of a systolic-array-based DNN accelerator. The key idea behind this approach is to replace critical faults with non-critical faults. In [20], this is achieved through dropping the computations mapped onto the faulty components, as dropping a small percentage of computations in DNNs do not impact the accuracy much, see dropout [21] and DropConnect [22] concepts.

Figure 7 illustrates the modified systolic-array design proposed in [20] for mitigating permanent faults in the MAC units of a systolic-array-based DNN accelerator. As illustrated in the figure, each PE is equipped with an additional MUX to bypass the MAC unit inside the PE. In case a fault is detected in the MAC unit of a PE during post-fabrication testing, the corresponding MUX is configured to bypass the faulty MAC unit. Note, the systolic-array architecture shown in Fig. 7 follows a weight-stationary dataflow where weights from the same neuron/filter are mapped onto the same column and are kept stationary inside the PEs during execution. Hence, the bypass operation corresponds to pruning of the weights mapped onto the faulty units.

To improve the performance of FAP, **Fault-Aware Pruning + Training (FAP+T)** is proposed [20]. The technique is based on the observation that DNNs are typically over-parameterized and pruning a small set of weights during training do not affect the final accuracy much [25]. Figure 8 presents a general flow for training a DNN against permanent faults. As highlighted in the figure, the fault map extracted through post-fabrication testing of the fabricated chip (along with the DNN mapping

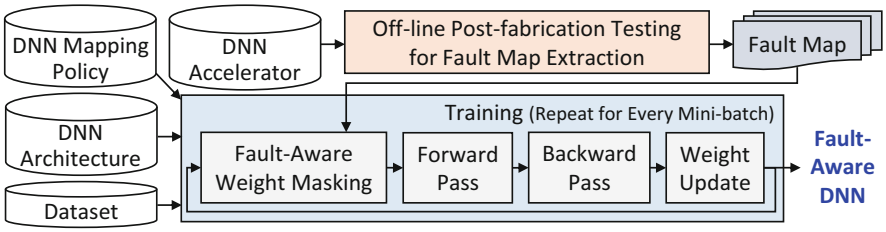


Fig. 8 General flow adopted for fault-aware retraining

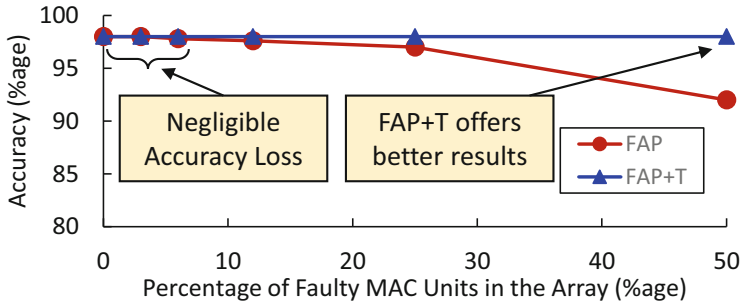


Fig. 9 Impact of using FAP and FAP+T on the classification accuracy of a fully connected DNN trained on the MNIST dataset at different fault rates [20]

policy) is used to force the weights to be mapped onto faulty MAC units to zero in each iteration of the training loop. This enables the DNN to adapt to the faults in the system and offer better performance compared to simple FAP.

Figure 9 highlights the effectiveness of FAP and FAP+T using a fully connected DNN trained on the MNIST dataset. As can be seen from the figure, both FAP and FAP+T help improve the resilience of the DNN against permanent faults in the computational array of a DNN accelerator; however, FAP+T offers better results at higher fault rates, i.e., negligible accuracy loss even when 50% of the total MAC units are faulty.

Although FAP+T is highly effective against permanent faults, its main drawback is that it involves retraining the given DNN, which may not be possible under some scenarios due to the lack of computational resources or a comprehensive training dataset. To address this issue, **Fault-Aware Mapping (FAM)** has been proposed [26]. FAM employs a saliency-driven approach to determine the mapping of the given pre-trained DNN for the given faulty chip. Figure 10 shows the general flow for applying FAM. First, the saliency of each DNN weight is computed using the L1 or L2-norm. Then, using an optimization algorithm and the knowledge of the faults, a mapping policy is determined that leads to the minimum (or a lower) sum of saliency of weights to be pruned due to permanent faults in the computational array of the DNN accelerator. In the end, the parameters of the DNN are rearranged according to the mapping policy (wherever possible) to avoid any run-time data

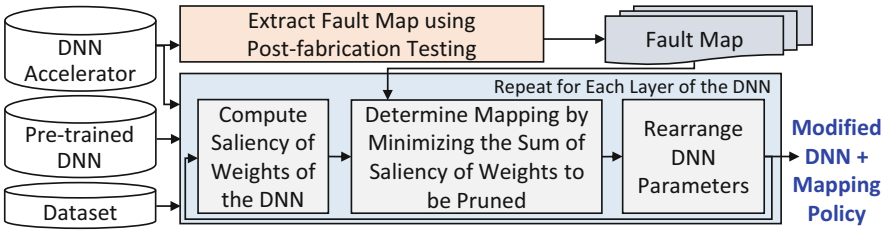


Fig. 10 Fault-aware mapping methodology

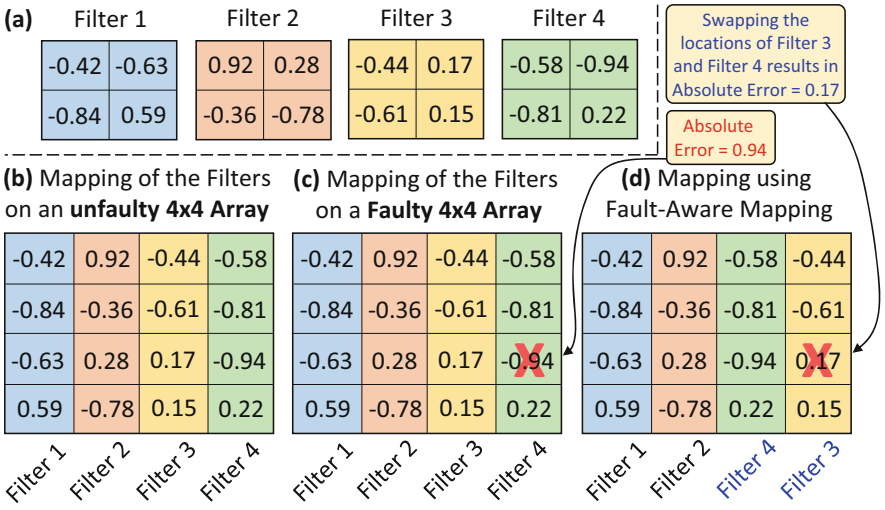


Fig. 11 An illustrative example of fault-aware mapping on a 4×4 systolic array

rearrangement operations. The output DNN and the mapping policy are then used together with FAP for reliable DNN inference. Figure 11 presents an example of how FAM can help in reducing the impact of permanent faults when used with FAP, and Fig. 12 highlights the effectiveness of the approach when used for the VGG11 network trained on the ImageNet dataset. Figure 12 clearly highlights that FAM can be employed without retraining specifically for low-to-moderate fault rates to get better results than only FAP.

3.4 Timing Error Mitigation

Aging in CMOS devices manifests as timing errors. These errors can have a drastic impact on the performance of a DNN interference system, as highlighted in Sect. 3.2.2. Conventional techniques such as aggressive voltage and frequency guard-banding result in significant energy and/or latency overheads. Therefore, it is

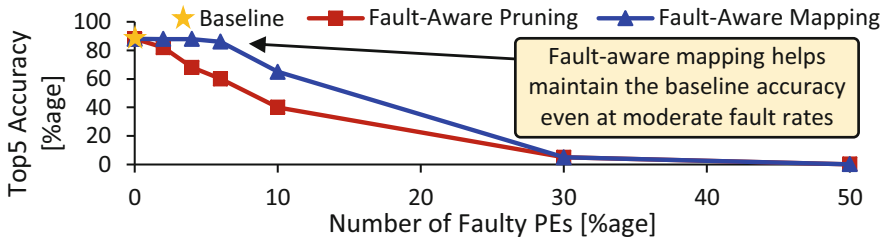


Fig. 12 Impact of fault-aware mapping on the classification accuracy of the VGG11 network trained on the ImageNet dataset

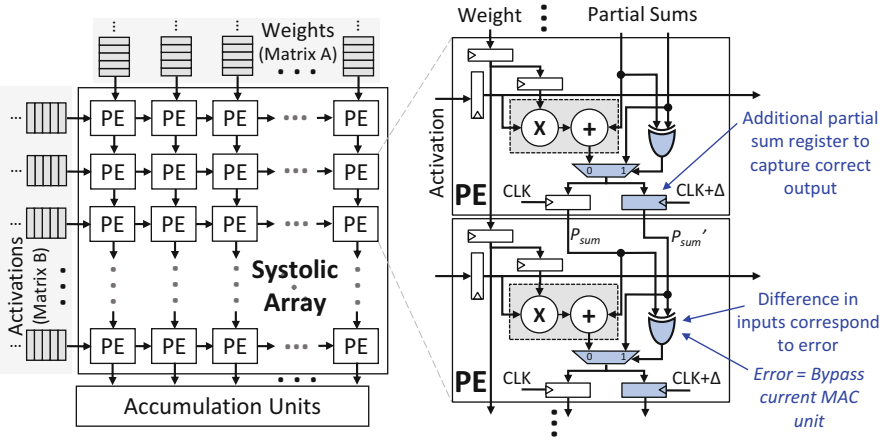


Fig. 13 Architectural modifications required in PEs of a systolic-array-based DNN accelerator to realize TE-Drop

crucial to address these errors at low cost to ensure reliable and resource-efficient DNN execution.

To address timing errors in the computational array of a systolic-array-based DNN accelerator at a low cost, Zhang et al. proposed TE-Drop [14]. TE-Drop works on the principle that the contribution of each individual MAC operation to the overall output of a DNN is very small. Therefore, a small percentage of MAC operations can be dropped without affecting the application-level accuracy of the system. To detect timing errors in the computational array, TE-Drop utilizes Razor flip-flops; however, instead of re-executing the erroneous MAC operation, it captures the correct MAC output in an alternate partial sum register operating on a delayed clock. Then, the succeeding PE is bypassed to feed the correct MAC output back into the computational flow. Figure 13 presents the architectural modifications required to realize the concept.

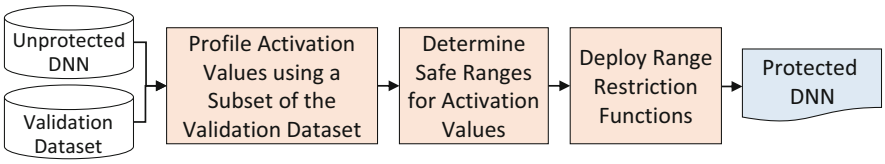


Fig. 14 General flow of range-restriction-based soft error mitigation techniques

3.5 Soft Error Mitigation

As highlighted in Sect. 3.2.3, soft errors at critical locations in a DNN-based system can significantly degrade the application-level accuracy of the system [16, 27]. Therefore, it is crucial to address these errors to ensure reliable DNN execution. Although conventional redundancy-based techniques (e.g., DMR and TMR) are highly effective against soft errors, they result in extreme overheads due to the compute-intensive nature of DNNs. Therefore, specialized low-cost techniques are designed to improve the resilience of these systems against soft errors.

To mitigate soft errors in SRAM-based on-chip memory, Azizimazreah et al. proposed a zero-biased SRAM cell design that has a higher tendency to switch to '0' in case an error occurs in the cell [28]. The intuition behind this design is that 0-to-1 bit-flips in DNNs result in a higher accuracy loss compared to 1-to-0 bit-flips. To mitigate soft errors in the computational array of a DNN accelerator, researchers have proposed different range-restriction techniques, e.g., Ranger [27], that bound the range of the intermediate activation values to a pre-computed safe range. The intuition behind these techniques is that soft errors can result in large activation values that may propagate to the output and impact the classification result. Therefore, abnormally large activation values that fall out of the normal range can be classified as erroneous, and dropping such values can mitigate soft errors due to the inherent resilience of DNNs to pruning. Figure 14 presents the general flow of range-restriction techniques. A similar technique is proposed in [29] for mitigating soft errors in the on-chip memory of DNN accelerators. Apart from the above-mentioned techniques, algorithm-based fault tolerance, such as checksum-based error detection and correction have also been proposed to mitigate soft errors in DNN systems at a low cost [30].

4 Conclusion

The state-of-the-art performance of DNNs for complex AI problems has led to their adoption for safety-critical applications as well. However, these systems have strict robustness constraints that are challenged by the hardware-induced reliability threats introduced due to the use of specialized DNN accelerators. The compute- and memory-intensive nature of DNNs prevents the use of redundancy-based techniques

for mitigating these threats. Towards this, this chapter covered different low-cost techniques for improving the resilience of DNN inference systems against soft and timing errors. The chapter also covered different techniques for mitigating permanent faults. Moreover, the chapter also discussed a holistic methodology for mitigating all types of reliability threats at low overhead costs.

References

1. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)
2. Fink, M., Liu, Y., Engstle, A., Schneider, S.A.: Deep learning-based multi-scale multi-object detection and classification for autonomous driving. In: Fahrerassistenzsysteme 2018, pp. 233–242. Springer, Berlin (2019)
3. Esteva, A., Robicquet, A., Ramsundar, B., Kuleshov, V., DePristo, M., Chou, K., Cui, C., Corrado, G., Thrun, S., Dean, J.: A guide to deep learning in healthcare. Nat. Med. **25**(1), 24 (2019)
4. Chen, Y., et al.: Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. IEEE J. Emerg. Sel. Topics Circuits Syst. (2019)
5. Kwon, H., Samajdar, A., Krishna, T.: Maeri: Enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnects. In: Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 461–475. ACM, New York (2018)
6. Jouppi, N.P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al.: In-datacenter performance analysis of a tensor processing unit. In: 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), pp. 1–12. IEEE, Piscataway (2017)
7. Hanif, M.A., Putra, R.V.W., Tanvir, M., Hafiz, R., Rehman, S., Shafique, M.: MPNA: A massively-parallel neural array accelerator with dataflow optimization for convolutional neural networks (2018). arXiv preprint arXiv:1810.12910
8. Baumann, R.C.: Radiation-induced soft errors in advanced semiconductor technologies. IEEE T-DMR **5**(3), 305–316 (2005)
9. Kang, K., et al.: NBTI induced performance degradation in logic and memory circuits: how effectively can we approach a reliability solution? In: ACM/IEEE ASP-DAC, pp. 726–731 (2008)
10. Raghunathan, B., Turakhia, Y., Garg, S., Marculescu, D.: Cherry-picking: exploiting process variations in dark-silicon homogeneous chip multi-processors. In: 2013 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 39–44. IEEE, Piscataway (2013)
11. Vadlamani, R., Zhao, J., Burleson, W., Tessier, R.: Multicore soft error rate stabilization using adaptive dual modular redundancy. In: Proceedings of the Conference on Design, Automation and Test in Europe, pp. 27–32. European Design and Automation Association (2010)
12. Lyons, R.E., Vanderkulk, W.: The use of triple-modular redundancy to improve computer reliability. IBM J. Res. Dev. **6**(2), 200–209 (1962)
13. Lu, W., Yan, G., Li, J., Gong, S., Han, Y., Li, X.: FlexFlow: A flexible dataflow accelerator architecture for convolutional neural networks. In: 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 553–564. IEEE, Piscataway (2017)
14. Zhang, J., et al.: Thundervolt: enabling aggressive voltage underscaling and timing error resilience for energy efficient deep learning accelerators. In: ACM/IEEE DAC, pp. 1–6 (2018)
15. Limbrick, D.B., Mahatme, N.N., Robinson, W.H., Bhuva, B.L.: Reliability-aware synthesis of combinational logic with minimal performance penalty. IEEE Trans. Nucl. Sci. **60**(4), 2776–2781 (2013)

16. Hanif, M.A., Khalid, F., Putra, R.V.W., Rehman, S., Shafique, M.: Robust machine learning systems: Reliability and security for deep neural networks. In: 2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS), pp. 257–260. IEEE, Piscataway (2018)
17. Gebregiorgis, A., Kiamehr, S., Tahoori, M.B.: Error propagation aware timing relaxation for approximate near threshold computing. In: Proceedings of the 54th Annual Design Automation Conference 2017, p. 77. ACM, New York (2017)
18. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998). https://doi.org/10.1109/5.726791
19. Ba, J., Caruana, R.: Do deep nets really need to be deep? In: Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, K.Q. Weinberger (eds.) Advances in Neural Information Processing Systems 27, pp. 2654–2662. Curran Associates (2014). http://papers.nips.cc/paper/5484-do-deep-nets-really-need-to-be-deep.pdf
20. Zhang, J.J., et al.: Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator. In: IEEE VTS, pp. 1–6. IEEE, Piscataway (2018)
21. Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.R.: Improving neural networks by preventing co-adaptation of feature detectors (2012). arXiv preprint arXiv:1207.0580
22. Wan, L., Zeiler, M., Zhang, S., Le Cun, Y., Fergus, R.: Regularization of neural networks using DropConnect. In: International Conference on Machine Learning, pp. 1058–1066 (2013)
23. Hanif, M.A., Hafiz, R., Shafique, M.: Error resilience analysis for systematically employing approximate computing in convolutional neural networks. In: 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 913–916. IEEE, Piscataway (2018)
24. Reagen, B., Gupta, U., Pentecost, L., Whatmough, P., Lee, S.K., Mulholland, N., Brooks, D., Wei, G.Y.: Ares: A framework for quantifying the resilience of deep neural networks. In: 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), pp. 1–6. IEEE, Piscataway (2018)
25. Han, S., Mao, H., Dally, W.J.: Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding (2015). arXiv preprint arXiv:1510.00149
26. Hanif, M., et al.: SalvageDNN: salvaging deep neural network accelerators with permanent faults through saliency-driven fault-aware mapping. Philos. Trans. R. Soc. A **378**(2164) (2020)
27. Chen, Z., et al.: Ranger: Boosting error resilience of deep neural networks through range restriction (2020). arXiv preprint arXiv:2003.13874
28. Azizimazreah, A., Gu, Y., Gu, X., Chen, L.: Tolerating soft errors in deep learning accelerators with reliable on-chip memory designs. In: 2018 IEEE International Conference on Networking, Architecture and Storage (NAS), pp. 1–10 (2018). https://doi.org/10.1109/NAS.2018.8515692
29. Hoang, L.H., Hanif, M.A., Shafique, M.: FT-ClipAct: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation. In: 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1241–1246. IEEE, Piscataway (2020)
30. Zhao, K., Di, S., Li, S., Liang, X., Zhai, Y., Chen, J., Ouyang, K., Cappello, F., Chen, Z.: FT-CNN: Algorithm-based fault tolerance for convolutional neural networks. IEEE Trans. Parallel Distrib. Syst. **32**(7), 1677–1689 (2020)

Index

A
Access point attacks, vii
Accuracy, 3, 24, 46, 74, 95, 129, 159, 182, 201, 235, 254, 287, 316, 345, 378, 396, 434, 463, 497, 526, 554
Activity recognition, 48, 50, 56, 57
Advanced driver assistance systems (ADAS), vii, 233, 235–238, 240–242, 244, 253, 265
Adversarial attacks, viii, 316, 380, 433–436, 443–448, 458, 459, 463, 464, 468–471, 473–476, 479–488, 497–514
Aging, 130, 464, 467, 554, 555, 558, 560, 564
Anomaly detection, vii, 32–34, 177–197, 253–281, 325, 332, 333, 335
Automotive systems, 259, 261–263
Autonomous vehicles (AVs), vii, 203, 205, 209, 216, 220, 221, 223, 315–338, 501, 520, 524

B
Backdoor, vii, viii, 316, 321–336, 395–399, 401–413, 427, 428, 470, 471, 524–526, 529, 530, 540, 548, 549
Backdoor attacks, vii, viii, 316, 321–332, 335, 336, 338, 395–428, 470, 471, 519–549
Backdoor detection, 333, 412, 428
Bias, 7, 27, 30, 37, 38, 104, 236, 258, 268, 384, 387–393, 467, 556
Bio-signal, 22–24, 32–40, 96

C
Capsule Networks (CapsNets), viii, 463–488
Clean-label attack, 397, 520, 526, 527, 533–544, 548, 549
Cloud offloading, 151–172
Compression, vii, 21–40, 79, 128, 129, 145, 155, 157, 291, 434, 473, 505
Compute-in-Memory architectures, 457
Computer vision, v, 21, 22, 50, 73, 92, 102, 128, 136, 291, 292, 297, 301, 433, 519–521, 523, 525
Constraints, v, vi, 22, 24, 26, 27, 29, 30, 35, 39, 40, 46, 51, 56, 57, 65, 66, 73, 77, 97, 99, 106, 111, 141, 144, 162, 190, 203, 205, 216, 220, 234, 262, 302, 328, 380, 381, 505, 506, 558, 566
Context, 76, 89, 93, 100, 104, 110, 135, 145, 178, 179, 183–184, 192, 197, 267, 271, 272, 286, 336, 344, 385, 395, 465, 466, 470, 477, 498, 514
Controller, 77, 117, 143, 177, 181, 254, 263, 265, 307, 315, 316, 318, 320, 321, 328–329, 335–336, 338, 509
Convolutional neural networks (CNNs), vi, vii, 3–18, 21, 58, 73, 79–86, 88, 91, 98, 130–132, 135, 137, 138, 204–207, 214, 219, 222–224, 236, 239, 256, 291, 316–318, 323, 345, 346, 351–352, 354–358, 360–363, 365, 371, 372, 403, 404, 407, 408, 457, 498, 500, 502, 507, 509, 511, 528, 533, 536–544, 548, 556
Cyber-physical systems (CPS), vii, 177, 201–205, 211, 213, 214, 219, 220, 223, 224, 233–250, 253–281, 377
Cybersecurity, 253

S. Pasricha, M. Shafique (eds.), *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing*, https://doi.org/10.1007/978-3-031-40677-5

D
Deep learning (DL), 4, 21, 50, 73, 103, 214, 233, 254, 288, 316, 345, 433, 506, 519, 558
Deep neural networks (DNNs), v, vi, viii, 4, 6, 7, 14–17, 24, 27–40, 74, 131–133, 204–206, 208–216, 218, 221–224, 256, 298, 316–319, 345, 346, 351, 353–357, 369, 371, 372, 395–428, 433–459, 463–488, 497, 498, 501, 505–507, 509–512, 521, 553–567
Deep reinforcement learning (DRL), vii, 218–220, 222, 223, 316–322, 325–329, 335–336, 338, 348
DNN reliability, 553–567

E
Edge computing, v, vi, viii, 49, 96–98, 190
Efficient deep learning, 233
eHealth Applications, 95–121
Embedded hardware, 74, 76, 286, 288, 308
Embedded systems, v–vii, 21, 253, 259, 285–308, 498, 504–511
Energy efficiency, 5, 119, 120, 141, 145, 153, 349, 434, 442, 445, 458, 471
Energy harvesting (EH), 46, 53, 57, 59–61, 141–144
Error resilience, viii, 106, 109, 558
Exploration, vi, 23–30, 33–38, 46, 62, 64–66, 74, 79, 92, 99, 109, 113, 120, 163, 217, 234–237, 240–250, 320, 328, 378, 478–479, 507, 511

F
Fault mitigation, 467, 559, 560, 562–564
Fingerprinting, 4–6, 9, 13, 14, 345–351
Framework, 4, 22, 82, 99, 128, 152, 179, 212, 235, 254, 294, 318, 343, 387, 451, 467, 499, 522
Freezing-of-gait (FoG), 130–132, 145

G
Graph data, 520, 522–523, 529–531, 545–547

H
Hardware, viii, 5, 6, 22, 24, 26, 27, 29–34, 36, 40, 74–76, 79, 81, 106, 107, 110, 111, 127, 151, 177, 211, 223, 278, 286, 288, 291, 301, 306, 308, 434, 436, 443–454, 458, 459, 464–467, 480, 497–514, 519, 553, 555–560
Healthcare, v–vii, 21–40, 65, 95, 96, 100, 106, 107, 113, 119, 127–145, 377, 497, 511, 553
Human activity recognition (HAR), 46, 52, 53, 59, 61–66, 130, 133–135, 145
Human pose estimation, 130, 136–140, 145

I
Indoor localization, vii, 3–6, 8, 10, 14–16, 18, 343–373
Indoor navigation, vi, 3–18
Intelligent systems, 110, 497, 501
Internet of things (IoT), v–viii, 21, 22, 45, 60, 74, 97, 98, 113, 177–197, 377, 497, 514, 521

L
Long short-term memory (LSTM), vii, 27, 32, 33, 131, 132, 182, 187, 189, 190, 194, 214, 223, 256–259, 266, 271, 272, 325, 327, 528, 540–544, 548

M
Machine learning (ML), 4, 45, 103, 127, 152, 181, 202, 254, 297, 319, 345, 377, 428, 463, 497, 519
ML security, 488, 501–514
Mobile devices, 4, 6, 17, 130, 141, 151–163, 165–167, 171, 347, 349–351, 357, 358, 361, 362, 372, 506
Model, 4, 21, 45, 74, 96, 128, 154, 178, 201, 234, 254, 292, 316, 346, 378, 395, 434, 463, 497, 519, 553
Multi-modal ML, vi, 95–121

N
Natural language processing (NLP), 21, 63, 128, 395, 497, 521, 522, 527–528, 534–540
Neural architecture search (NAS), vi, 21–40, 248–250
Neural networks (NNs), v, 5, 6, 48, 50–54, 63, 64, 74–77, 81, 89–92, 98, 115, 117, 121, 134–136, 158, 159, 163, 182, 197, 204, 218, 220–222, 224, 235, 236, 257, 260, 266, 271, 291, 321, 384, 385, 387–389, 396, 400, 402, 414, 415, 427, 428, 449, 477, 498, 505, 527, 543, 548, 555, 556
Node sensitivity, 384, 391

Noise, 3, 98, 133, 181, 219, 234, 263, 293, 322, 360, 377, 424, 433, 465, 501
Novelty detection, viii, 397, 398, 412

O
Object detection, v, 75, 81, 90, 128, 204–211, 222, 224, 233–236, 238–241, 243, 244, 246, 247, 289, 292, 300, 322, 332, 470, 553
On-line retraining, 413

P
Perception architectures, 233–242, 244, 245, 247–250
Permanent faults, 464, 467, 555, 558–560, 562–564, 567
Power line inspection, 288
Privacy, v–viii, 46, 49, 51, 103, 106, 137, 138, 140, 345, 349, 350, 464–466, 497–514, 521
Probabilistic analysis, 132, 182

R
Recurrent neural networks (RNNs), 60, 179, 182, 187, 188, 214, 223, 256–258, 266, 271, 556
Reinforcement learning (RL), vii, 51, 60, 106, 110, 111, 119, 120, 151–172, 202, 216–219, 316, 319, 327, 520
Requirements, 5, 6, 22, 24–27, 29–31, 34, 38, 40, 45–48, 58, 64, 75, 77, 78, 85, 89, 98, 99, 104, 106, 107, 109–111, 119, 136, 137, 140, 141, 143, 152, 213, 220, 234, 237, 238, 263, 292, 297, 308, 351, 405, 415, 434, 498, 506, 509
Reverse-engineering defense, 402
Robotic vision, v
Robot planning, vii
Robustness, vi–viii, 5, 14, 46, 55–61, 66, 73, 115, 140, 194, 344, 382–384, 389–393, 421, 423, 433–459, 464, 469, 471–480, 483, 487, 488, 498–500, 506, 507, 509–511, 514, 566

S
Search, 24, 28, 35–38, 40, 73, 92, 109, 111, 128, 153, 235, 239, 241, 243, 245–249, 302, 323, 331, 466, 471
Secure indoor localization, 343–373
Security, vi–viii, 51, 73, 74, 177, 178, 180, 253, 316, 336, 343–345, 349, 351–357, 369, 372, 396, 458, 463–467, 469–488, 498, 501, 507, 514, 520, 521
Sensor association, 181, 184–187, 189–191, 195–197
Sensor fusion, 203, 233–236, 239–241, 243–248, 250
Smart healthcare, 119, 128, 553
Soft errors, 466, 467, 498, 554, 555, 558, 559, 566
Speech recognition, 128, 395, 519, 521, 523, 528–529, 540–544, 553
Spiking neural networks (SNNs), v, vi, viii, 463–488

T
Telecommunication infrastructure inspection, 287, 288, 296, 299, 308
Timing errors, 467, 555, 560, 561, 564–567

U
Unmanned aerial vehicles (UAVs), vi, vii, 73–81, 91, 92, 285–308

V
Visual disaster recognition

W
Wearable devices, vi, 45–66, 130, 133, 141, 144, 145