Exploring Adaptive Machine Learning Applications in Edge-IDS for Vehicular Systems

Loic Lorente Lemoine

School of Computer Science and Informatics
Cardiff University

This thesis is submitted for the degree of
Master of Philosophy

Cardiff University

October 2025

Abstract

Replacing the eras of horse-drawn carriages, vehicles have shaped the way today's world operates. From planes, that enable frequent continental and international travel, to consumer cars, which provide city and cross-country transportation for employment or leisure commuting. However, the introduction of electronic hardware and corresponding software into vehicles has created cybersecurity risks. This is observed in decades-old technology, such as the defense-less CAN bus, as well as newer EV infrastructure protocols. The resulting social and economic threats are ever more prominent when considering the volume, cost, and dangers of vehicles.

As the latency- and resource-constrained vehicular environments limit the applicable security techniques, machine learning has risen as an effective method for defending the CAN. However, diverged data over time, which may be derived from different drivers, road types, or unseen attacks, can impact model performance in various use cases. Additionally, existing solutions may require support from external computation, such as cloud infrastructure, for inference and adaptation, which introduces additional communication latency. Additionally, such connectivity cannot be assumed given the mobility and longevity of vehicles.

To address this, this thesis explores edge-based intrusion detection with adaptive methods for vehicular networks. Initially, continual learning on EV infrastructure is presented, where a reverse autoencoder learns to reconstruct a new attack while maintaining reliable initial-task performance. This then platforms the main contribution of Tiny Machine Learning and Tiny Online Learning for the CAN bus. Individually trained and evaluated on three CAN IDs, the models demonstrate reliable attack detection with the main fully-connected model achieving F1-scores of 0.993, 0.984, and 0.975 respectively. The online learning method is evaluated locally with an early-stage model, where output-only updates enable improvements over time, such as a peak F1-score increase of 0.04, while deployed computation times present a viable system for in-vehicle use. Lastly, a test-rig provides further insights into model quality.

Table of contents

List of figures

List of tables

Acknowledgements

I am deeply grateful to my supervisors, Dr Amir Javed and Dr Nick (Nhat) Pham, for their guidance over the course of my MPhil. Their continuous support in their supervision has enabled me to successfully complete this work and ultimately allowed me to proceed further into my research career..

I would like to extend thanks to Kha Huynh for his support. I would also like to thank my colleagues in the AgileCPS lab, at Cardiff University, and other collaborators for their contributions and additional support to my journey and this work. They include Hakan Kayan, Amanjot Kaur, and Dr Omer Rana. Additionally, I would like to thank my MPhil funders, Thales Group.

I would like to thank my friends and family for their continued support throughout my entire academic journey and beyond.

Chapter 1

Introduction

According to Transport for London's Road Task Force Technical Note 12, 54% of households in the city have at least one car [1]. More recent statistics from the Department for Transport's National Travel Survey [2] and Vehicle Licensing Statistics [3] first state that only 22% of households in Great Britain don't have a car with the latter calculating a total of 41.7 million licensed vehicles in the UK at the end of June 2024. The numbers back up what we observe in society on a day-to-day basis: there are a lot of vehicles.

According to Autocar [4], the average weight of a new car has risen by roughly 400kg within the past seven years, 1553kg to 1947kg. The World Health Organization (WHO) [5] approximates 1.19 million road traffic collision-related deaths per year, and between 20 and 50 million more people suffer non-fatal injuries, where it is the leading cause of death for children and young adults. In 2023, the road casualties report by the Department of Transport [6] highlights 1,624 fatalities, 29,711 killed or seriously injured casualties, and a 132,977 casualties of all severities. These numbers show that the combination of vehicle mass, alongside capacity for speed, allows a misued vehicle to effectively function as a weapon. Paired with this, economic risks also exist. Moneybarn [7] highlight an increase in the cost of new vehicles, with the Renault Zoe presenting a 123.41% increase from £13,650 to £30,495 within a decade (2012-2022). The high cost of these machines imposes extreme financial burden on individuals, with the Zeo's 2022 price representing 117.4% of the median salary.

Proving risks to both social and financial complications, vehicles must be as safe and as secure as possible. However, a significant challenge remains: the majority of vehicles rely on the insecure and decades-old CAN bus. Unauthorized access to this system would allow a hacker to take control of the vehicle, even while in motion, posing a sever and critical safety risk with grave consequences.

As efforts increase to progress infrastructure into automated smart cities, the global smart car market is observed to have reached $67.3 billion in 2023 with a market report predicting a rise to $256.7 billion by 2032 [8]. This has resulted in new vehicles and infrastructure. As manufacturers continue to produce this smart infrastructure with government-backed incentives, these vehicles will continue to replace their traditional counterparts, shifting us into a new generation of the automotive industry. New technologies to manage this infrastructure, such as Open Charge Point Protocol (OCPP), can present new attack surfaces for attackers to exploit. Furthermore, the large number of existing vehicles and and increased turnover to newer models may present a new challenge: old vehicles risk getting left behind. As these vehicles age, manufacturer support for maintenance and upkeep may diminish, which may introduce new vulnerabilities. This can result in increased risk to the highlighted issues.

Based on the above findings, key problems exist:

1. Insecurities exist in protocols of vehicular-related systems. This primarily includes the CAN bus, which by default has no security features, but also expands to newer protocols such as OCPP.

2. With most households having a vehicle, there exists many possible targets for attackers.

3. Vehicles have many social and economic factors, such as high financial burden on individuals and are often irreplaceable. Additionally, features of a vehicle, such as speed and weight, can cause harm and even fatalities.

4. Modern-day consumerism and a possible lack of schemes for handling older vehicles during transitions to newer vehicles increase the quantity of older vehicles on the road. These vehicles are more likely to experience insufficient maintenance and upkeep over time, providing hackers with additional avenues and vulnerabilities to exploit.

From the more recent electric vehicle (EV) infrastructure communication protocols such as OCPP and ISO 15118, to the decades old CAN bus which enables communication between components, vehicle-based applications are susceptible to injection attacks. As described, the controller area network (CAN) bus lacks security features resulting in it being susceptible to an array of attacks; such as Denial-of-Service (DoS) and spoofing. These attacks allow malicious entities to gain control of vehicles, which has been observed in existing vehicles in both in-the-wild theft [9] and lab-controlled vehicle manipulation [10, 11]. In cases of real-world theft, multiple reports and discussions exist on vulnerabilities that enable thieves to gain physical access to vehicles, such as an exposed CAN bus connection. Meanwhile, as in lab-controlled environments, the risk of danger increases as an attacker can take control of an entire moving vehicle.

Requirements of latency and resource constraints limit the security available for the vehicular CAN. To address this, researchers have studied Intrusion Detection Systems (IDS) with machine learning and deep learning techniques. Classification algorithms are capable of distinguishing between benign samples and crafted attacks using binary or multi-label supervised frameworks. These methods allow for precise predictions that enable further diagnostic efforts. However, real attack data is typically not available, which has popularized anomaly detection methods that rely solely on benign data in order to identify anomalous behavior. More specialized and personalized approaches focus on driver identification, using the CAN bus to identify individual driver behavior. Such an IDS solution does not detect specific attacks directly, but rather extrapolates patterns for an alternative security and monitoring task. Some researchers emphasize the importance of dynamic models that leverage future data to mitigate the impact of novel unseen attacks or the risk of deviations in data patterns within a system's life time. For example, federated learning [12–15] can enable an adaptive system with broader data coverage, as future updates involve aggregating data from other vehicles. However, most studies do not sufficiently consider hardware resource limitations and deployment, instead typically serving as a benchmark to demonstrate IDS performance against attacks. In addition to this, it is not considered how the nature of vehicles, such as mobility and required upkeep, create physical and logistical challenges for cloud connectivity, highlighting the need for local models accompanied by update methods.

This thesis explores the challenges of an adaptive IDS solution that considers local computing requirements, demonstrating systems that are suitable for low-power hardware in two vehicular-network applications. For this, two pieces of work are produced, with the latter forming the main contributions of this thesis: (1) demonstrating continual learning for defending electric vehicle infrastructure; and (2) enabling locally-compatible stream-based online learning for the CAN bus.

For the main contributions, the following is explored:

1. Small TinyML deep learning models for three different CAN IDs which detect their respective attacks through anomaly detection. With four-digit number of parameters, the autoencoders learn to accurately reconstruct normal CAN traffic, resulting in reliable and high quality models.

2. Deployment techniques of the models to three heterogeneous edge devices, alongside a post-training quantization-based deployment. With on-board computation, one model observes inference times as low as 0.85ms and as high as 4.89ms, showcasing the feasibility and flexibility of deployment across diverse in-vehicle hardware environments.

3. Online learning which enables effective learning on edge devices. Evaluation workflows on a early-stage model presents improvements exceeding of 0.5 in F1-score and 5 percentage points in accuracy, highlighting the capacity for progressive adaptation and refinement.

4. CAN and IDS behavior on a vehicle-simulation test-rig. Translating simulated vehicle actions into CAN messages, the system is further validated with local real-time inference and observations to justify TinyML + TinyOL [16] capacity.

While the applications include EV infrastructure and the CAN bus intrusion detection, the techniques can be applied to a wide range of learning objectives in vehicular-networks. For example, network monitoring for diagnostics and analysis or personalization-focused systems such as driver identification.

1.1 Challenges

The following are the identified challenges and problems in existing approaches to vehicular intrusion detection systems:

1. **Hardware Constraints.** The hardware within the infrastructure of vehicular networks may not be designed for resource-intense applications such as modern deep learning. The ECUs within the CAN bus often involve microcontrollers which may have limited power and memory. This can limit the suitability of various model architectures. For example, larger or more complex models may not meet memory requirements or may introduce increased latency due to the weaker hardware.

2. **Network Requirements.** The CAN bus facilitates the communication of a large amount of messages per second, often at the frequency of a message within milliseconds. Therefore, an IDS must be capable of inferring samples at a rate that does not cause delays. While this ties into the prior challenge (item 1), an IDS must not only be appropriate for deployment, but also operate in a timely

manner. For this, the IDS's computation must be completed in time for the next CAN message to be processed. Additionally, detection time should be as low as possible, as once an attack has been detected, attack response must take place. The faster the speed of detection, the more time allotted and quicker the response can be.

3. **Change in Data.** In both our intended applications of intrusion detection systems, and other possible applications (such as driver identification), data itself or its patterns may change over time. This may be due to data drift from degrading or replaced components or new environments, such as different drivers, different road types, or time of day. In this thesis, we explore paradigms for non-static and adaptive deep learning models to address the possibility of this issue in a vehicle's lifespan.

4. **Outliers.** Even when a model is reliable at its task, any failure can become fatal in the use case of vehicular-networks. Outliers may not greatly impact performance metrics, such as false positive rate or sensitivity, as they may be infrequent within their given test set. However, a vehicle operates for large periods of time and the stream the IDS processes may be unpredictable. Therefore, it is possible for it to encounter these outliers on a regular basis. For example, extreme false positives may occur less than 0.5% of the time within a test stream that was recorded on a typical driving session. However, this would still cause a number of false flags within a typical driving session, especially when considering the frequency of CAN messages.. Therefore, in this thesis, outliers are taken into consideration when evaluating model performance with discussion provided on the challenge.

1.2 Contributions

This section states the contributions of each work, which are presented in chapter 4 and chapter 5 respectively.

1.2.1 Continual Learning for Defending Electric Vehicle Infrastructure

In this initial study, continual learning is explored for anomaly detection in an autoencoder. In this, a small autoencoder with on-device continual learning is implemented for the Raspberry Pi 5 with PyTorch, addressing challenge number 1. Addressing challenge number 3, the model learns new tasks in electric vehicle infrastructure attacks in a scenario where the model is initially trained on one attack. Further addressing challenge number 1, the continual learning is evaluated on two separate workflows referred to at "cloud-edge" and "edge-only". In the former, a cloud server controls the model updates. The latter computes all operations locally on the device. With this, we see how the edge device compares to the cloud setup, determining local suitability for the task. Additionally, this project acts as a starting point for the subsequent piece of work. This study focuses on OCPP and ISO 15188 data, which is commonly found in EV charger infrastructure and can be prone to different types of attacks.

1.2.2 Adaptive Intrusion Detection on the Edge for the CAN bus

In this work, we explore an IDS which features both TinyML and TinyOL [16] methodologies to protect the CAN bus. Addressing challenge number 1, three low-parameter models are trained on individual CAN IDs. TinyOL enables on-device adaptation of future CAN streams, addressing challenge number 3 and number 4. To further complete addressing challenge number 1, the model architectures are deployed onto edge devices, with a TinyOL deployment featuring on-device backpropagation and a Arduino deployment with 8-bit integer post-training quantization. Evaluated locally with an early-stage model, simulating and "out-of-tune" model, the online learning's adaptation capacity is demonstrated with both stochastic and mini-batch update workflows. This study focuses on the CAN bus, the major communication network for vehicle components which is insecure in its default form.

1.3 Publications

The work in this thesis has resulted in two publications:

1. Edge-Based Anomaly Detection in Electric Vehicle Charging Infrastructure with Continual Learning
   <u>Loic Lemoine</u>, Amanjot Kaur, Nhat Pham, Omer Rana
   MobiUK 2025 - Seventh UK Mobile, Wearable and Ubiquitous Systems Research Symposium
   This extended abstract explores a strategy for adapting to new attacks in the scenario of electric vehicle infrastructure at the edge using continual learning. I

am the co-first author and main contributor to the ideas and experiments of the paper. My collaborators help with literature review, paper writing, and ideas. This work will be presented in Chapter 3 of the thesis. It was published to the MobiUK 2025 Symposium.

2. Edge-Efficient Online Learning for In-Vehicle Security
Loic Lemoine, Amanjot Kaur, Hakan Kayan, Charith Perera, Amir Javed, Nhat Pham, Omer Rana
IEEE SECON 2026 - 22nd Annual IEEE International Conference on Sensing, Communication, and Networking
At the time of submission, this publication has been submitted and is awaiting response. This paper explores on-device online learning for the vehicular CAN with a small and efficient model deployed onto three edge devices. I am the lead author and the main contributor to the ideas, experiments, and implementation of the paper. My collaborators help with literature review, paper writing, and ideas. This work will be presented in Chapter 5 of the thesis and has been submitted to IEEE SECON 2026.

1.4 Thesis Scope

This thesis investigates the problem and application of efficient on-device inference and training of deep learning-based intrusion detection systems aimed at deployment within vehicular networks. To present the proof of concept in both works, the scope is restricted to maintain focus on the central ideas, leaving further evaluations and broader considerations to future research.

In the first work, two out of three possible attacks are selected from the dataset. This is due to continual learning requiring a minimum of two tasks, the original task and a new task. Utilizing additional tasks, and its accompanied challenges, can be considered out of scope and part of future work. Additionally, the reverse autoencoder enables new attacks to be considered new tasks, while adaptation on a traditional anomaly detection autoencoder would better resemble traditional online learning. Utilizing a classifier instead of this method would require incremental-based continual learning which features the modification of the model architecture for each new task. This method, alongside its accompanied challenges, are considered out of scope.

In the second work, which represents the main contribution of this thesis, the models are trained on three different CAN IDs, creating a model for each CAN ID. The selection of three CAN IDs demonstrate that the model can be applied to different parts of the CAN. Further expansion to other CAN IDs would introduce repetition, and meaningful inclusion of many CAN IDs would broaden the scope excessively as it may necessitate addressing new challenges that fall out of the scope and can be considered as future work. Additionally, the online learning is evaluated on only one CAN ID, serving as a proof of concept. The scope of the online learning evaluation is limited to one CAN ID to focus on the key ideas and demonstrate the feasibility of the approach. The reduction of model parameter precision with quantization is applied to only the Arduino-deployed models with TensorFlow Lite. Applying quantization to the other deployment methods is considered beyond the scope of the work, as doing so may require a separate investigation focused on quantization techniques and their deployment, which introduces further challenges to address, both hardware and software related. Lastly, both convolutional architectures are not deployed to the Pi Zero device due to compatibility. The fully-connected model is demonstrated with a NumPy-based forward pass and backward pass for inference and learning. Convolutional architecture requires further engineering considerations which are not considered due to additional challenges which are out of scope. Therefore, the deployment of the convolutional models utilize existing frameworks.

1.5 Thesis Structure

This thesis is split into 5 chapters. Background and literature are organized into chapter 2 and chapter 3 respectively. They describe the relevant key concepts to this thesis and presents a detailed literature review. The two main works are presented in chapter 4 and chapter 5 respectively. Wrapping up the thesis, chapter 6 presents future works and conclusions.

Chapter 2

Background

Relevant concepts and and definitions are presented in this chapter to provide the necessary background for this thesis. This introduces the CAN bus, alongside the electric vehicle infrastructure protocols OCPP and ISO 15188. Then, edge computing and online learning methods are presented. Lastly, the evaluation metrics used in this thesis are explained.

2.1 Controller Area Network (CAN)

This section presents the Controller Area Network (CAN) bus, defining it, describing relevant attacks, as well as demonstrating CAN data and existing case studies of CAN attacks.

2.1.1 What is the CAN?

The CAN is a protocol for efficient, robust, and reliable communication. It was introduced in 1985 by Bosch with the purpose of reducing wiring complexity and system cost. Due to the increasing number of components in vehicles, it has been adopted as the standard for in-vehicle networks, and is found in almost all modern vehicles. In the protocol, a single two-wire bus with a twisted pair of CAN low and CAN high connects the nodes together. By replacing direct wiring with this, physical complexity is reduced, enabling the increase of components with reduced overheads such as financial cost and weight.

Electronic Control Units, or ECU, are components that are commonly found as the nodes of the network. Each ECU may control a specific functionality of the vehicle. No host device is required as all devices in a CAN receive all messages. Therefore, rather than direct communication, ECUs transmit their CAN frames to all other devices. Alongside this, frames are transmitted one at a time with priority-based transmission. Inside vehicles, it is common to find sub-CAN systems with gateways that can connect multiple CANs together. Each of these CAN may be separated based on purpose or proximity. For example, ECUs relating to the the speedometer, engine, and pedal may be clustered together. The ECU typically contains three components [17]: microcontroller, CAN controller, and CAN transciever. The CAN controller is typically found within the ECU's MCU and handles the CAN frame protocol. A CAN transceiver allows the CAN controller to receive and transmit CAN frames.

Different variations of the CAN bus include the low-speed CAN, high-speed CAN, CAN FD, and CAN XL [17]. Most vehicles feature the CAN FD, where the FD stands for Flexible Data-Rate.

Transmission Example. ECU A transmits a frame to the CAN bus with the arbitration ID of 0x423. If no other frames are transmitted at this point, the frame proceeds. If other frames are transmitted, the frame proceeds if it has an arbitration ID of higher priority. While the frame is present on the bus for all ECUs to see, the ACK field is adjusted accordingly once it has been received by an ECU. The ECU that requires the frame accepts it through its CAN controller and processes the information within the payload.

2.1.2 CAN Frame Features

The CAN consists of different sections. Each section is represented by a set number of bits which include the Start of Frame (SOF), CAN ID (or arbitration ID), Data Length Code (DLC), Payload, Cyclic Redundancy Check (CRC), Acknowledge (ACK), and End of Frame (EOF). It is important to note that many studies consider the message timestamp as a feature, however, this is not a part of the CAN frame but rather it is a common part of the logging method when sniffing the CAN.

1. **Start of Frame (SOF).** The start of a CAN frame is represented by a SOF bit which houses a dominant 0 bit.

2. **CAN ID (or Arbitration).** The message identifier is the CAN ID. An ECU will accept a CAN frame based on this ID. If multiple CAN frames are sent at the same time, the CAN frame with the lowest ID is considered to have the highest priority. The other CAN frames must be resent. This ensures that actions that are more important, like those that have a higher risk, happen as soon as possible.

Table 2.1 The parts of the CAN frame and the number of bits typically allocated to them.

Feature	Bits
SOF	1
CAN ID	11 or 29
DLC	4
Payload	64
CRC	16
ACK	2
End of Frame	7

Two variations of the CAN ID exist, with the standard frame utilizing an 11-bit ID while an extended CAN frame increases this to 29 bits.

3. **Data Length Code (DLC).** The number of bytes in the payload is represented in the 4-bit DLC section. For example, a payload may use six out of the available eight 8 bytes.

4. **Payload.** The important information that an ECU wants to communicate to another ECU is found in the payload part of the CAN frame. It may present another ECU with an action to take or a piece of information, such as the current speed of the vehicle.

5. **Cyclic Redundancy Check (CRC).** The data integrity is checked by the CRC.

6. **Acknowledge (ACK).** Two acknowledgment bits (1 bit ACK slot and 1 bit ACK delimiter) provides confirmation to the sender that their message has been successfully received.

7. **End of Frame (EOF).** The end of the CAN frame is marked by seven EOF bits.

2.1.3 Attack Types

The CAN was intended for isolated systems. Therefore, security via authentication or encryption is not a native feature. This makes it vulnerable to attackers. These attackers can access the CAN through physical surfaces such as the On-Board Diagnostics (OBD) port or wireless communication protocols Bluetooth or Wi-Fi. The main threat to the

CAN is considered to be intrusion attacks which include impersonation attacks, such as spoofing and replay. While these there are different variations of impersonation attacks, they may have overlapping themes. In this work, we focus on these impersonation attacks, instead of (or in conjunction with) fuzzy and denial-of-service attacks. These attacks can be considered as more random and can be described as flooding attacks, while the impersonation attacks can be considered more stealthy.

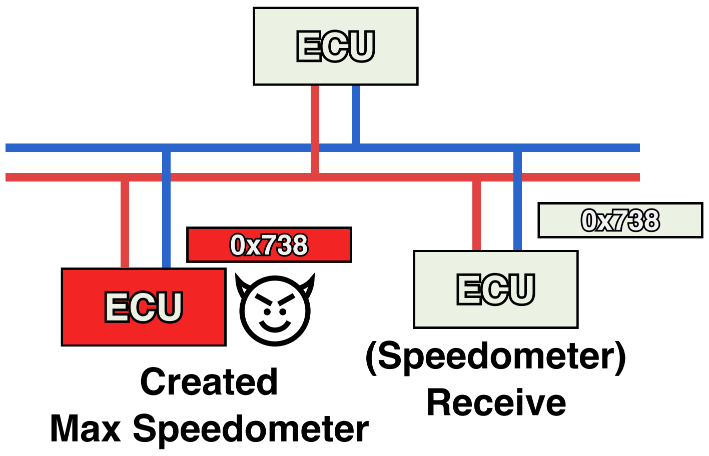


Fig. 2.1 An example of a spoofing attack on the CAN bus.

Spoofing involves injecting fake messages into the CAN with the intent of deceiving the system by pretending to be a legitimate source. This type of attack can be observed within the masquerade attack. Figure 2.1 demonstrates an attacker transmitting a hand crafted message maximum speedometer message where they attempt to maximize the speedometer information which the recipient ECU handles. In that current moment, the real ECU transmitting the speedometer information messages may have been transmitting a frame with a low speed. Therefore, the malicious message may cause malfunction from corresponding components that rely on the speedometer information.

A replay attack involves the attacker injecting real CAN frames. To obtain these frames, the attacker may have had prior access to the network where they captured existing messages in the real-time CAN stream. Figure 2.2 demonstrates events at two time points. In the before time point, an attacker sniffs the network for transmitted CAN messages. The attack will eventually capture a message or a series of messages. In the later time point, the attacker replays the recorded message. While the message was technically benign at one point, the nature in which it has occurred in is not. If the message was of a specific piece of information about the speedometer, the speedometer may be at a different state at the later time point. For example, the attacker may be transmitting a CAN frame informing other components that the speedometer is

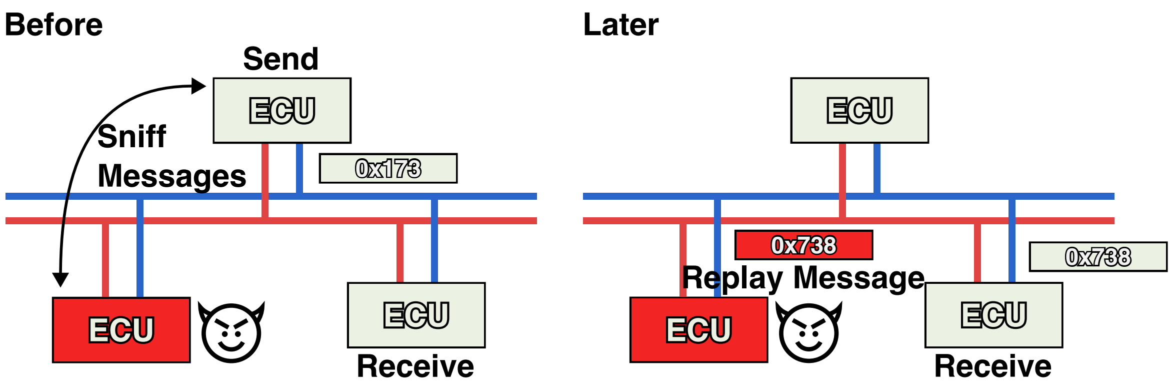


Fig. 2.2 An example of a replay attack on the CAN bus.

at 30MPH, but at later time point the vehicle is accelerating past 40 MPH. Similarly to the spoofing attack, the vehicle may experience malfunctions due to the incorrect information.

2.1.4 Existing CAN Data

To advance the research of CAN data, CAN attacks, and intrusion detection systems, there has been an effort from researchers to create CAN bus attack datasets. Manufacturers typically do not disclose information about their CAN systems, therefore, unless there is collaboration between industry and research, deep access to CAN systems is not typically accessible. The first reason for this is the cost. Vehicles have a high financial cost and may require various approvals to enable its use. Secondly, even if a real vehicle is employed, it may require additional effort, time, and man power from researchers to derive useful analysis and understanding. In this work, we utilize both simulated data from a vehicle test rig and an open dataset from a real vehicle.

Simulated Data. A CAN can be simulated via various tools. For example, the Instrument Cluster Simulator (ICS) [18] is a simulator for SocketCAN which allows you to locally simulate a CAN. It features virtual components such as a dashboard with RPM and speedometer as well as door locking. In the case of this work, we utilize a vehicle simulation test rig which translates vehicle actions from within vehicle-based video games into CAN packets for a real vehicle instrument. For example, accelerating in the game with the accompanying Logitech peripherals is reflected on the dashboard with the speedometer and RPM increasing. Data is observed and collected through an On-Board Diagnostics (OBD) port.

CAN-MIRGU. The open data used in this thesis is the CAN-MIRGU [19] dataset. It contains CAN frames from a fully electric modern automobile manufactured in 2016. The dataset is suitable for anomaly detection autoencoder workflows as it includes 17 hours of benign data. For attack classification or evaluations, it features multiple physically verified real injection attack and 10 simulated masquerade and suspension attacks.

2.1.5 Case Study

The vulnerability of the CAN is not unknown to attackers. While attacks can be considered rare when the amount of cars in the world are considered, the threat still exists and can be fatal. Here, examples are provided of documented occurrences of CAN vulnerabilities and attacks in both lab-controlled settings and in the real-world.

Stolen Vehicle. Thieves can use the CAN to gain access to and steal a vehicle [9]. This example starts with a twitter user posted about their vehicle being tampered with, specifically one of the front headlights. Over a period of a few months, this continued at different points which the user continued to log on his account. Then suddenly, the vehicle was no longer there and had been stolen. After some analysis with a manufacturer-provided remote diagnostics app, they came to the conclusion that the vehicle was on the receiving end of a CAN bus intrusion attack where the attacker accessed the network through the diagnostics port available at the headlights. They were then able to cause the vehicle to malfunction, allowing them to enter. The author of the article stated that tools for these types of attacks exist on the blackmarket. Some of these attack devices are designed for a specific vehicle type, highlighting how accessible and feasible it is to conduct such an attack.

Controlled Attack While Driving. Charlie Miller and Chris Valasek targeted a 2014 Jeep Cherokee [10] [11]. They initially used Wi-Fi to hack the Harman-Kardon head unit entertainment system but eventually got into it through its cellular network. Through the entertainment system, they were able to reach the vehicle's CAN bus. While such a critical attack is expected to be difficult, the researchers described how easy it was. Not only was the specific model vulnerable, they accessed 2013 to 2015 models of Jeeps and other vehicles such as Dodge Viper, Ram, and Durango. The entertainment system featured a "serial-peripheral interface (SPI) connection between the OMAP processor in the head unit and a V850 microcontroller that in turn could talk to the CAN bus". This did not allow instant communication with the CAN bus.

To address this, they re-flashed the microcontroller through the SPI, which was the most difficult part. With this control, they were able to control critical components such as the brakes, steering, etc. While this was eventually fixed, an attacker could have caused serious financial or social damage, including death, with this vulnerability.

Possible Occurrences. Based on the real-world theft and lab-controlled hijacking examples, the threat of a vulnerable CAN bus is made clear. While stories like this, which involve CAN bus attacks, are not in the news everyday, a large focus of cyber security is preparing for what could happen tomorrow, or in the future, so that it may never happen. Therefore, it is important that continuous work is applied to address vulnerabilities and ensure the attacks can not happen. For example, given the Jeep vulnerability, and given the high-speed motorways that frequent thousands of vehicles a day, it would be possible for a hacker to hijack a moving vehicle. This could lead to loss of life and injury.

2.1.6 Intrusion Detection Systems

To protect the CAN against these attacks, an intrusion detection system is necessary. This system must satisfy requirements of minimal latency and minimal resource usage as the CAN is time-critical and is typically comprised of resource constrained devices. Methods such as cryptography and authentication are described as unsuitable for the CAN bus as they do not typically satisfy these requirements. Additionally, use of the cloud can be deemed inappropriate because the external connection unnecessary introduces latency alongside a new requirement of continuous connectivity. This is not always possible with mobile vehicles. Therefore, solutions must be found elsewhere.

These solutions can be found in machine learning, where small-scale and efficient design can be harnessed to produce algorithms that are low in latency with a small hardware footprint without majorly sacrificing performance. Many studies pursue this route, exploring the use of both traditional and statistical machine learning algorithms as well as modern deep learning architectures to learn patterns that allow for attack detection. Literature on machine learning and deep learning is provided in Chapter 3.

2.2 Electric Vehicle Infrastructure

The growth of electric vehicles have required the introduction of new infrastructure to support its features, such as charging. These new infrastructures involve new networks and protocols that can become a target to attackers, presenting new cyber security threats. Attacks may include denial-of-service (DoS), tampering, and impersonation attacks. Both home-based charging and public-based charging present varying vulnerabilities [20]. In homes, EVs serve as IoT devices which may be sensitive to attacks. For example, an attacker may "intercept and modify, or fabricate" features such as voltage and current parameters communicated by the battery management system. This is also possible in public chargers, where there may be a lack of security measures, as well as a wider array of stakeholders and a wider interconnected system.

One study explores an online adaptive Random Forest classifier for electric vehicle infrastructure [21]. Evaluated on the CICEVSE2024 dataset [22], the authors address concept drift with an efficient and scalable solution in both binary- and multiclass-classification where a ADWIN algorithm detects drift through a dynamic sliding window of recent observations. Online learning then adapts the model. The system presents suitable metrics for real-time deployment. Highlighting the importance of an adaptive solution for EV infrastructure data, their method ensures the model remains at sufficient levels of performance even after multiple sets of drift.

2.2.1 Protocols

The implementation of new electric vehicle infrastructure, due to its emergence, has required new protocols to facilitate communication. In this study, the utilized dataset derives its data with the Open Charge Point Protocol (OCPP) and the ISO 1511 standard. While security is considered in these protocols, vulnerabilities which attackers can exploit may still be prevalent.

Open Charge Point Protocol. OCPP coordinates communication and power flow between charging points, the control center, the EVs, and the grid [20]. While it can be combined with transport layer security (TLS), Antoun et al. state that this practice is ignored by manufactures due to unwanted overhead and cellular network costs. The authors state that even with TLS, OCPP is exposed to impersonation attacks. It has also been observed to be insecure against man-in-the-middle (MITM) attacks [23]. Additionally, OCPP does not specify underlying communication technology alongside lacking a definition of strong security services. This results in a dependency of the security provided by other protocols at lower levels [23].

ISO 15118. The ISO 15118 standard [24] is a standard that "outlines the digital communication protocol that an EV and CS should use to recharge EV's high-voltage battery" [20]. It defines the vehicle to grid (V2G) communication interface, coordinating EV chargers to match the grid's capacity. Additionally, it enables Plug and Charge. The first generation is known as IOS 15118-2 and the 2022 second generation is known as ISO 15118-20. While the 2022 version can verify the identity of devices through private keys, it does not handle the verification of the integrity of the devices [25]. Therefore, if a device is compromised, the system may also be compromised.

2.3 Edge Computing

Edge computing is a paradigm focused on executing compute as close to the source of data as possible. In the modern day, a large amount of compute, especially for deep learning, is conducted on cloud data centers that leverage high-end hardware. Due to limitations of proximity and wireless communication, this introduces latency as communication overheads are introduced. By computing at the edge, this overhead is minimized. However, while some edge devices, such as the discrete GPU-powered NVIDIA Jetson, may have more powerful hardware, most edge devices are low-power and do not possess the same computing resources as the cloud alternatives. For example, IoT devices, such as typical microcontrollers and Raspberry Pis, can have memory limitations in the kilobytes. Other devices, such as smartphones and smart watches, may possess better hardware, but can be limited by battery constraints. Therefore, systems must be efficient and feasible on hardware with minimal resources so that both deployment and low-latency at-the-source computing can be achieved.

Machine learning, especially deep learning, is typically trained and deployed on high-end hardware. But, this is not always possible, especially when delving into local-based and more accessible systems. Typical solutions to this involve accelerated designs on hardware that can take advantage of it. Additionally, small-scale design or low-overhead architectural choices when creating models can aid in this issue.

Optimization. To further enable machine learning on the edge, optimization techniques allow models to remove redundancies which result in improved model size and speed. The main methods in deep learning are pruning and quantization. Pruning removes low-impact neurons or synapses from the network to reduce the size of the model. In this work, the latter technique is employed. Quantization reduces the precision of the model. This is typically achieved by converting a model's parameters to a data type with less bits. In PyTorch, models are stored in floating point 32, however, the range and precision of this data type is often not necessary. With quantization, a model's weights and/or activations, is often reduced to smaller floating point types such as fp8 and fp4. Additionally, models can take advantage of integer arithmetic for increased speeds with quantization to int8, int4, etc. By reducing the amount of bits, less memory is required for the model. The speed of the model is also improved as memory access is less costly and quicker arithmetic can be achieved by supported hardware.

Utilizing Edge Devices. Single board computers, such as the Arduino Nano 33, are low-power hardware that can take advantage of optimized machine learning. Machine learning at the edge protects privacy, as the computation occurs at the source of the data. Additionally, as already discussed, it provides improvements on latency. Lastly, the low-cost of these devices means that machine learning applications can be more accessible and widely-used.

Tiny Machine Learning (TinyML) involves efficient machine learning on edge devices, such as microcontrollers. Frameworks such as EdgeImpulse and TensorFlow Lite provide tools to compress, deploy and run the models directly on such devices. TensorFlow Lite, specifically TensorFlow Lite Micro (TFLM), will be utilized in this work for MCU-based deployment.

CAN Hardware. The CAN bus consists of ECUs, gateways, or other components. The electronic control unit (ECU) are the typical CAN nodes and can be seen as the components of the vehicle. An ECU typically contains a microcontroller with an integrated CAN controller. Processing within the ECUs occurs at the MCU, enabling messages to be interpreted and actions to be conducted. Andreica et al. detail three real-world ECUs: (1) Infineon Tricore TC224 produced in 2015, (2) Infineon Tricore TC397 produced in 2018, and (3) S12X3 produced in 2006. While the TC224 and S12X3 are both single-core, the TC397 contains a quad-core CPU at 300Mhz. The devices have memory constraints of 96KB, 2528KB, and 64KB respectively. From these descriptions, the utilized hardware in this thesis can be considered as comparable prototypes to real-world ECUs.

2.4 Online Learning

Online learning (OL) is a method for adaptive machine learning where a model, that has already been trained, learns once it has been deployed. At this deployed stage, the model and its processes are described to be online. Typically, online learning involves streaming-based learning where samples from an incoming real-time data stream are selected for instances of training. These samples may be used for training sequentially in a stochastic method or grouped together for a mini-batch based approach. In both of this approaches, the model may conduct inference as normal, and store samples for training later. Or training may occur in real-time. OL is often found in applications where it is not possible or feasible to train the initial offline model on an entire training set. Alternatively, not all types of data in a training set may be available at the point of initial offline training. Furthermore, tasks with changing data may require a dynamic model that continues to update itself with new samples.

2.4.1 Catastrophic Forgetting

During the new training sessions, a problem can occur in the model's learning. As the model is optimized for the new and unseen training samples, the previous weights are overwritten and the model's parameters may "deviate" from being optimized for the original training data. This is called catastrophic forgetting. This concepts describes when a trained model starts to "forget" its previous training data as its original weights are overwritten in place of new weights that minimize the loss of new samples. In online learning, catastrophic forgetting is not usually intended and must be mitigated. In order to prevent catastrophic forgetting, common methods include replay of old samples during training, as seen in the first contribution (see chapter 4). Replay involves providing new training sessions with stored sets of seen samples from previous training sessions. This can address the catastrophic forgetting problem by ensuring the model stays optimized for existing tasks. If the problem of catastrophic forgetting is not addressed, the model may become unsuitable for its application.

2.4.2 Continual Learning

A variant of online learning is continual learning. This method is utilized when a trained model must learn a new task post-deployment. For example, a multi-image-classification problem may attempt to classify dogs, cats, and parrots. However, now the model must be able to classify otters. In what can also be called incremental learning, the model's output layer is incremented to support the new class. Then, the model is trained on samples of the new class. These samples can be attained from the incoming data stream or from human-controlled operation.

Continual learning is prone to catastrophic forgetting. This is especially more prevalent as the model optimized for a new task which may come from a different distribution or may exhibit different patterns. As the new data may purposefully be different in this way when compared to what the model is already optimized for, continual learning is very prone to catastrophic forgetting.

2.5 Evaluation Metrics

This section of the background details the metrics used in the evaluations of this thesis. Split into two sections, the first contains the metrics used to determine model quality and performance on the data itself, while the second measures the model architecture's computational performance.

2.5.1 Performance Metrics

A primary performance metric to determine the model quality is the F1-score. Precision (Equation 2.1) measures the proportion of correct positive predictions out of all positive predictions. Recall (also known as sensitivity) (Equation 2.2) measures the proportion of all positive samples that were predicted as positive. Both of these metrics are combined to form the F1-score (Equation 2.3), which represents the harmonic mean of precision and recall.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.1}$$

$$\text{Recall (sensitivity)} = \frac{TP}{TP + FN} \tag{2.2}$$

$$F_1\text{-score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2TP}{2TP + FP + FN} \tag{2.3}$$

While the least important out of the performance metrics, the overall model accuracy (Equation 2.4) provides an insight at the overall correct predictions. The false positive rate (Equation 2.5) is identified as a key metric as it provides important insights into whether a model is suitable for the use case. Additionally, the overall confusion matrix (see Equation 2.6) can be presented to provide a more granular look at the model's predictions.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.4}$$

$$\text{False Positive Rate (FPR)} = \frac{FP}{FP + TN} \tag{2.5}$$

	Predicted Positive	Predicted Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

(2.6)

The the score provided by the area under the ROC curve (ROC AUC), which measures the relationship between the specificity (true positive rate) (see Equation 2.7) and sensitivity (false positive rate). In short, the ROC measures the TPR against the FPR at various thresholds and its AUC determines how well the model can distinguish between positive and negative classes.

$$\text{Specificity} = \frac{TN}{TN + FP} \tag{2.7}$$

Lastly, the loss of the model on the validation set, a subset of the full training dataset, is used to aid in evaluating the quality of the model based on how well its predictions match the target value. This is computed with mean-squared error (Equation 5.1). As autoencoders are used, the loss is considered the reconstruction loss, as it represents how well the model is able to reconstruct the original input. Computing this loss on the output, of the same size as the input (e.g. 40 in the fully-connected model), produces a matrix where each value is the loss of the corresponding feature. This is then reduced to one single loss value by calculating the mean.

2.5.2 Computation Metrics

Computation metrics measure the computational costs required to conduct the operations of the intrusion detection systems. This primarily includes both inference time and training time. This is then paired with message frequency statistics to determine whether the time metrics are suitable for the use-case. Inference time measures the time it takes for the machine to conduct an instance of the forward pass of the model. This can occur in both mini-batch and singular (stochastic) inputs. On the other hand, training time, which is specifically the TinyOL [16] time in the main study, measures the time it takes for the machine to conduct one instance of backpropagation. This can occur in both mini-batch and singular inputs. As stated, the main study specifically tracks the training time of the TinyOL method, which does not apply full backpropagation. Given the time intervals CAN messages function with, and the times of the model operations, the metrics are presented in milliseconds.

Chapter 3

Literature

In this chapter, relevant literature is highlighted, demonstrating various related work and their research gaps. This focuses on literature in machine learning vehicular intrusion detection systems.

3.1 Related Work In CAN Intrusion Detection Systems

In this subsection of literature, we detail a general overview on CAN intrusion detection systems with machine learning. More specifically, we identify existing anomaly detection methods. This category of intrusion detection systems can involve the use of machine learning and deep learning methods. Other intrusion detection systems, such as those that are considered signature based [26], are not considered.

3.1.1 Traditional Machine Learning Solutions

Traditional machine learning typically rely on statistical algorithms, with many functioning as "white-box" models. These methods are considered more interpretable and explainable. Additionally, these methods typically require less data when compared to deep learning's neural networks. Common methods include K-Nearest Neighbors (KNN), Support Vector Machines (SVM), alongside decision trees and its variants.

In CICIoV2024 [27], they create a dataset and subsequently evaluate multiple algorithms, which include Logistic Regression, Random Forest, and Adaboost. While their methods were not suitable for multi-class classification, their Random Forest model is able to achieve perfect scores in categorical classification (benign, DoS and spoofing) when using both binary and decimal features. When utilizing binary classification, with classes of benign and attack, all algorithms present near perfect performances. Their models can be improved with training that considers imbalanced data, as benign data is naturally more common, and as such, their multi-class classification results may improve.

Random Forest (RF) is explored again in Alfardus et al. [28], alongside K-Nearest Neighbors (KNN) and Support Vector Machine (SVM). With impersonation, RPM and DoS attacks, their methods obtain perfect detection results with no false positives. On the fuzzy dataset, RF and SVM achieve 100% true positive rate and a 0% false positive rate. The KNN model achieves a tpr of 98.04% with a false positive rate of 0.1%.

Decision Trees (DT), Random Forest, XGBoost (XGB), and CatBoost (CB) are evaluated on the car hacking dataset in Dangwal et al. [29]. They produce a multiclass-classification system where the model predicts between five classes of four attacks and one benign class. Alongside this, they "debias" the data. The algorithms achieve F1-scores above 0.98 across all classes, even on a version of the dataset which is considered "non-modified".

Deriving an additional feature from the time stamp of messages, Gundu and Maleki [30] append the time interval between messages to each sample. With this modified dataset, they evaluate classification with supervised methods, including Random Forest, XGBoost, and K-Nearest Neighbors. The RF and XGBoost method outperforms the KNN method. The two tree methods achieve F1-scores of 0.93 and accuracies of 0.94 and 0.92 respectively.

Overall, these studies demonstrate reliable detection of attacks, however, transitioning to the use of deep learning methods is also considered a viable approach that can address issues found in the traditional machine learning techniques. It can be observed that the varying results, from 100% accuracies to low-90s%, demonstrates that the task of defending the CAN is difficult, especially given that real-world attacks are often unseen and may not resemble an attack in a test set. Additionally, different CAN systems may be more difficult to defend. Therefore, deep learning methods may be more appropriate. Despite statements that describe deep learning approaches as time-consuming and slow [30], neural networks are typically more scalable, do not require intensive feature extraction, and can handle more complex information, such as data with temporal and spatial dependencies. Additionally, while machine learning can work better with limited data, CAN bus datasets often contain a high number of samples, therefore, the improved fitting capacity of deep learning can be leveraged. Lastly, small models, such as in TinyML, alongside optimization techniques, like compression through quantization, can make deep learning methods suitable for low-resource and time-critical applications.

3.1.2 Deep Learning Solutions.

In this subsection, deep learning-based intrusion detection systems are presented. These are split into different focuses, with more general approaches in 3.1.2.1. This is followed by intrusion detection systems with an alternative objective of driver identification in 3.1.2.2. Lastly, more relevant intrusion detection systems that focus on adaptation and edge-based computing are highlighted in 3.1.2.3 and 3.1.2.4.

3.1.2.1 General Approaches

Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), and Gated Recurrent Unit (GRU) are capable of binary- and multi-class classification of CAN bus attack types in Rai et al [31]. Suitable for the "overtime-based" information found in some CAN systems, these architecture choices are specialized for temporal data. They utilize the CAN frame timestamp, CAN ID, and payload as input for these models. Additionally, to introduce spatial features, they extract the CAN bus data into image representations for an image-classification based approach. For this, they utilize the state-of-the-art VGG-16 model. While the models are reliable for attack classification, it cannot be predicted how the model will behave to unknown attacks. Additionally, their deployment on resource-constraint edge devices are unknown or not feasible as the chosen architectures have high complexity and computational demands.

Similarly, Song et al. [32] trains a deep convolutional neural network which comprises of a reduced Inception-ResNet model. They apply this model to their own data collected from a real vehicle. This data is extracted into samples by their framebuilder which extracts the 29 CAN ID bits from 29 messages into a 29x29 2D grid data frame. They utilize pure bits from the CAN stream to avoid the computational costs of preprocessing. With this grid data frame set of features, the model optimizes itself for the temporal pattern of CAN sequences. Specifically, as bits are used in form of an image, the CNN model learns the spatial patterns of the grid data frame. They consider cooperation with vendors as future work in order to take advantage of semantic features alongside adversarial training techniques to detect unseen attacks.

IDS-DEC [33] proposes the first deep embedded clustering in in-vehicle network security intrusion detection with a deep LSTM Convolutional Autoencoder with Entropy

Regularization embedded into fuzzy clustering. This method utilizes anomaly detection and jointly trains the clustering module and the autoencoder module, achieving improved accuracy and F1-scores on DoS, fuzzy, spoofing gear, and spoofing RPM attacks when compared to baseline methods. However, the authors highlight that when in more complex scenarios involving multiple attacks, performance is degraded.

Zhang et al. [12] present a two-stage graph-based IDS for anomaly detection. The first stage handles binary classification between benign and attack with a CAN message graph and a GNN. If an attack is detected, a second stage utilizes an anomaly message graph where a GNN with an openmax output, which replaced the softmax used in training, to enable the handling of zero-day attacks. As softmax picks one of the available classes, it would not be suitable for the task. Openmax, on the other hand, is able to pick outside of the known class set. Their GNN-based IDS can simultaneously handle various attack types with high reliability, such as 100% detection of DoS attacks and 97.26% detection of replay attacks.

Xu et al. [34] combines a Multi-dimensional Long Short-Term Memory (MD-LSTM) with Self-Attention Mechanism (SAM) to create MULSAM, achieving an accuracy of 98.98% on a classification task which includes a benign class and five attacks (DoS, fuzzy, spoofing, replay, delete). Their MLP and CNN comparison models achieve 80.08% and 82.07% respectively, performing well on flood-based attacks but poorly on classifying normal messages. Their LSTM-based models, such as the stacked LSTM, MD-LSTM, and the main MULSAM, all achieve significantly higher accuracies, with a lowest of 97.07% in the stacked LSTM. This demonstrates the superiority of specialized architecture on data with temporal properties.

Ashraf et al. [35] discovers anomalous events with a LSTM autoencoder where the CAN bus is used to extract two network features: total number of packets sent by a unique CAN ID in a given time window and the size of outbound messages from unique IDs. Trained for 500 epochs, the LSTM autoencoder outperforms baseline methods, such as ANN and SVM, with a precision of 0.99 and recall of 1.0. The SVM baseline, which outperforms the ANN, achieves precision and recall of 0.92 and 0.89 respectively.

While not deployed, Aljabri et al. [36] demonstrate a lightweight classificaton-based IDS. Evaluated on various attack types, the inclusion of a feature engineering method achieves almost perfect metrics of 99% accuracy alongside 0% false positive rate. This method involves treating the CAN bus data field as separate byte features rather than as one individual feature. Their fully-connected classifier includes 10117 learnable parameters and a total memory footprint of 41256. The model requires 20064 FLOPs, a stark improvement when compared to their baselines.

While results are generally reliable, many of these methods do not consider the next step: deployment. Therefore, their suitability for this is either unknown, due to lack of consideration, or assumed to not to be optimal, as their IDS choices may have large sizes. Lightweight methods are capable of achieving comparable results to larger and/or more complex methods [36], however, feasibility in deployment must be further evaluated.

3.1.2.2 Driver Identification

An alternative application is driver identification via the CAN. This is suitable for both alternative "intrusion detection", such as physical theft, or general personalization applications.

To defend vehicles against car theft, Khan et al. [37] uses an LSTM- and convolution-based model for unauthorized driver detection at the CAN level. This LSTM-FCN model takes input data down two stages; a combination of 1D-convolution and squeeze and excite blocks and a LSTM block. The outputs of both stages are concatenated together and processed through an output softmax layer. With two datasets leveraging important features from real-vehicular CAN data, the system demonstrates improved performance when compared to other studies as it reliably classifies 14 drivers almost 100% of the time. However, classification requires data to be optimized for. Therefore, the model may be ineffective for detecting if the driver is not one in the training set. It would not be feasible nor possible to use thief data in the model's training as such data is not available.

Tseng et al. [38] conducts anomaly detection in driver identification with a Generative Adversarial Network (GAN). When incorporating PCA, the CLGAN model achieves an accuracy of 98.5% accompanied by an f1-score of 0.991. However, when compared to the other GAN evaluation models, their proposed model requires the highest memory with the smaller models still producing comparable results when using all types of road data. Additionally, the authors call for more effort in diverse datasets for vehicle theft detection based on OBD data. This should include different road types as it can strongly impact performance. This presents a problem of data availability at training time, which can be solved with an adaptable system to expose the model to more data allowing for better per-driver personalization.

Similarly, [39] trains a RNN-LSTM-based GAN for anomaly detection in car theft. With 32 essential features from the CAN, reduced from an original 51, they achieve 88.4% accuracy and f1-score of 0.789 on average across four different drivers. Their system only considers one driver at each instance, limiting its use-case when a vehicle may have multiple drivers. Additionally, the system could be improved by considering different road conditions, such as highways or parking lots. This highlights the importance of research towards adaptive deep learning-based CAN intrusion detection systems.

3.1.2.3 Adaptive Systems

In the research of defending in-vehicle systems, federated learning has been presented as an appropriate solution to evolving data, such as unseen attacks. Federated learning describes an approach where distributed training is leveraged to obtain model updates which are then aggregated into a main model. This technique enables an adaptive model that respects privacy, as data is not shared, while enabling an adaptive model.

Zhang et al. [12] implements federated learning into their GNN-based IDS. The authors state that different vehicle states can cause variations in CAN messages and CAN message frequency, leading to data drift. Additionally, limited exposure during training to different drivers, vehicles, and driving scenarios can "restrict the comprehensiveness of the derived models". To address this issue, two federated learning techniques, FedAvg [13] and FedProx [14], are evaluated. When compared to a model trained on only one vehicle, which they simulate by limiting the training data, the federated learning approaches improve detection performance with an increase from 32.01% to 80.03% (FedAvg) and 99.04% (FedProx) when using an interval of 100 messages.

In Althunayyan et al. [15], a multi-stage IDS classifies samples in the first stage, with a second stage leveraging anomaly detection in an LSTM-autoencoder to catch any missed attacks. When the second stage detects an attack, the classifier is updated. Leveraging the multi-stage adaptability, hierarchical federated learning is proposed. HFL utilizes various edge aggregators that combines different sets of models. This is then progressed to a global aggregator that produces the new global model. The staggered system improves scalability which is crucial considering the number of vehicles it may continuously serve. The authors consider streaming learning as future work to allow the vehicle to adjust dynamically in real time. The effects of adversarial attacks should also be considered.

While federated learning enables an adaptive model, it assumes participation and cloud connectivity, which are both not guaranteed. Additionally, it may require further consideration into how update aggregation can affect local specialization, especially when vehicles may encounter different conditions. Ensuring sufficient local-only updates addresses these issues.

Ahmed et al. [40] looks at continual learning for CAN bus data to adapt a model to different driving scenarios. For each new driving scenario, a new output module is added which uses a transfer learning-based weight cloning scheme. Training involves the replay of weighted subset of previously learned driving scenarios to strengthen connections. A supervisor model selects which output module to utilize during inference. Evaluating with real data, 8 of 113 recorded driving scenarios are recorded are selected. The scenarios collect multiple sensor information, which is extracted from CAN frame payloads. Their method achieves better average forgetting ratios when compared to their baselines as well as lower loss on the new task. The problem of catastrophic forgetting in this use case demonstrates how data can change depending on driving scenarios, highlighting the importance of a model that can adapt.

3.1.2.4 Edge-Focused

Continuing from the general approaches (3.1.2.1), the importance of edge-based intrusion detection systems has been noticed, with efficiency and deployment presented as a focal point in various studies. This brings IDS research closer to real-world-ready use as realistic constraints and issues are addressed.

The survey "Deep Learning in the Fast Lane" [26] describes IDS architectures. Onboard self-detection involves a vehicle utilizing its own onboard compute. On the other hand, remote detection involves offloading detection to a remote service. This introduces latency that may exceed the suitable amount for the CAN. Alternatively, collaborative detection leverages vehicles working "in tandem", such as in truck platoons. In this study, an edge-based CAN IDS is considered self-detection, where all compute occurs on-device, such as on an ECU.

Im and Lee [41] utilize TinyML and TensorFlow Lite for an Arduino-deployed CAN IDS. They employ a CNN model called LC-IDS with two routes for two different sets of features: (1) a grid of CAN ID sequence and (2) a grid of data field of the most recent CAN frame. After the dense layer in both routes, a feature fusion layer feeds into an output sigmoid function. With an output size of two, the model is a binary classifier between attack and normal. Compared to alternative studies which leverage GPUs, such as an RTX3090 and a Tesla K80, or full desktop-level CPUs, the 4,045 parameter model produces comparable detection performance and inference latency despite harnessing low-power hardware and a smaller model. This is highlighted by a large reduction in FLOPs, where only 11,242 FLOPs are required compared to the Inception-Resnet approach, which sees 100,132 million FLOPs for its 1,708 million parameters. This further proves the suitability of lightweight approaches. TinyML is capable of providing efficient prototypes, therefore in this work, we aim to further explore TinyML for the CAN bus with alternative methods as well as the inclusion of on-device TinyML online learning through the TinyOL approach.

Andreica et al. [42] compares the power of Android Head Units against in-vehicle ECUs to assess performance for deploying intrusion detection systems. The study looked at automotive-grade microcontrollers, such as the 32-bit Infineon AURIX TC224, 32-bit Infineon AURIX Tricore TC397, and the low-end 16-bit S12XEP. For the android-based devices, they included mobile devices such as Samsung smartphones, and two Android head units. Evaluated with various intrusion detection systems, the Android head units slightly outperform the ECU's under specific conditions. Their work highlights the deployment of IDSs directly onto existing vehicle hardware. This helps bridge the gap towards real-world use with realistic evaluations.

Xu et al [43] deploy their MULSAM system onto a Ultra96-V2 FPGA with Vivado high-level synthesis (HLS) and compare the deployment to a Jetson TX2. The latter they describe as a "commercial-off-the-shelf GPU-based embedded platform". Their results show that their FPGA-based deployment is more efficient than the GPU-based Jetson and more effective for intrusion detection. Their stacked LSTM variant presents the FPGA hardware accelerators to be about twenty-one times as energy-efficient as the TX2. With about 2W of power consumption on the FPGA, all models exceed 5W on the TX2.

SecCAN [44] is an FPGA-implemented extended CAN controller featuring an embedded intrusion detection system that "incurs limited resource overhead" when compared to a standard controller. To further push the boundary of efficiency, a Quantized Neural Network (QNN), featuring Quantization-Aware Trained (QAT) linear layers, is embedded into the controller. With reliable detection results, which are comparable to other IDS schemes, the integrated IDS can infer samples before the reception window is completed.

These studies highlight the opportunities and directions available for an edge-deployed IDS alongside appropriate computational times. For example, a model must not only be deployable on the resource constraint hardware, but also conduct inference within time constraints, line in [44].

3.2 Research Questions

Based on the described literature, as well as the summary in section 3.3, the following main research questions are identified:

RQ1 How can we enable timely operation of a reliable Intrusion Detection System (IDS) in the CAN with an efficient system?

RQ2 How can on-device online learning enable an adaptive model without impeding CAN bus latency?

RQ3 How can an adaptable TinyIDS be effectively deployed and enabled on the edge with resource constrained hardware?

Research questions 1 and 3 address the hardware constraints and network requirements challenges (see number 1 and number 2 in 1.1). Meanwhile, research questions 2 and 3 help address the challenges of changing data and outliers (see number 3 and number 4 in 1.1). In the research questions, "timely operation" means that inference, and/or online learning, is conducted in a manner that "keeps up" with the frequency of CAN messages.

3.3 Summary of Motivations and Research Gaps

This section serves as a brief summary of the contributions and gaps this work addresses which has been justified based on the background and literature.

If an IDS cannot be deployed locally, it may require a cloud or fleet connection to handle computation. As they are mostly mobile when in use, a vehicle may lack reliable and consistent connection to the external computation. This becomes a problem as unnecessary latency is introduced, which can be detrimental in such a high message frequency and low latency use case. Therefore, efficient models, whether that be through small architectures or accelerated design, should be continuously researched to enable reliable on-device inference. Such an IDS would mitigate the Jeep vulnerability (see subsection 2.1.5), as when the hacker gains control of the ne ttwork they can still be stopped, when they would typically have full control. For example, their message injection may deviate from the benign patterns of the CAN which can trigger a detection

While an on-device deep learning-based intrusion detection system may help solve real issues, like those described in subsection 2.1.5, such a system could still fail without further considerations. Based on the statement from [12], it is clear that CAN models may suffer, now or in the future, the problem of data drift. This may be due to different drivers, different road types, and/or different setups/states. Static models may no longer be effective due to these effects. This is also true in driver identification, where a vehicle may have multiple drivers or training data is limited due to a lack of diverse training data (e.g. road types). This may result in increase false flags within existing intrusion detection systems. Alternatively, given the example of the Jeep vulnerability (see subsection 2.1.5), a model may not detect the malicious disabling of the vehicle's brakes as it has not been refined sufficiently in the case of an autoencoder or incrementally-trained in the case of a classifier.

To adapt an IDS, federated learning techniques may not always be appropriate. This may be for the reasons mentioned earlier in this section. For example, [15]'s hierarchical federated learning requires a road-side server for training. Therefore, federated learning methods, or other upkeep methods, may typically assume constant connection to an external server. Or, it may be due to the longevity of vehicles, as they are on the road for a long time, sometimes decades. Therefore, as a vehicle gets older, and new vehicles are released, the upkeep and quality of updates may not be guaranteed due to a lack of federated learning volunteers, or the manufacturer's maintenance and upkeep no longer being available. Additionally, if the IDS is designed to run independently, with its own on-device updates, the vehicle may not be able to take advantage of federated learning. For example, if the independent learning focuses on personalization, the updates are only relevant to the specific vehicle.

Ultimately, the literature highlights a need for the exploration of a small, efficient, and adaptable intrusion detection system for the CAN bus. With this, timely inference and updates can be achieved, presenting a system that is appropriate for real-world use.

Chapter 4

Continual Learning for Defending Electric Vehicle Infrastructure

This chapter presents the first contribution.

4.1 Introduction

With various government-backed incentives and the constant advancement of technology, EVs are increasingly becoming more common. Alongside this, there is a corresponding rise of EV chargers to support it. With this new infrastructure, new communication protocols have been introduced, such as the Open Charge Point Protocol (OCPP) and ISO 15118 standard. While some security considerations are present, vulnerabilities still exist. Creating new surfaces for attacks, the vulnerabilities within these communication protocols can compromise data integrity or lead to unauthorized control over vehicles/charging components. Additionally, attack detection or classification methods in deep learning rely on training data, however, types of attacks are never definitive and it is also unlikely that data patterns remain unchanged over time. Therefore, an initial static model may no longer be suitable for its task as data drifts or new unseen attacks are encountered.

To address this, we explore continual learning for EV infrastructure attacks, presenting a system where a deep learning model learns the representation of new attack types after its initial training session. This chapter presents a demonstration of continual learning for electric vehicle infrastructure attacks. Through two competing workflows, we show that on-device local continual learning is suitable for the use-case. We evaluate the model's continual learning with and without techniques for mitigating catastrophic forgetting on two types of attacks. When incorporating a replay method, results show that the model successfully learns the representation of both attacks despite being trained in different sessions. Comparative workflow evaluations present a local based continual learning on an edge device for EV charging infrastructure is suitable with acceptable training and inference times on a Raspberry Pi 5.

4.1.1 Research Questions

Adapting from the main research questions (see section 3.2), the following research questions are identified for this initial study.

- **RQ1:** Can an autoencoder successfully reconstruct attacks that are trained in sequential training instances for anomaly detection?
- **RQ2:** Can an edge-only based system enable this continual learning when compared to a cloud-edge system?

These research questions are answered with the following contributions:

- **EV Power Consumption Anomaly Detection.** An autoencoder is trained to detect EV charger attacks based on patterns present in power consumption.
- **New Task Detection.** A reverse autoencoder undergoes continual learning to learn the representation of attacks considered as "new task". A replay technique mitigates catastrophic forgetting, enabling successful reconstruction of two different tasks with reliable attack detection.

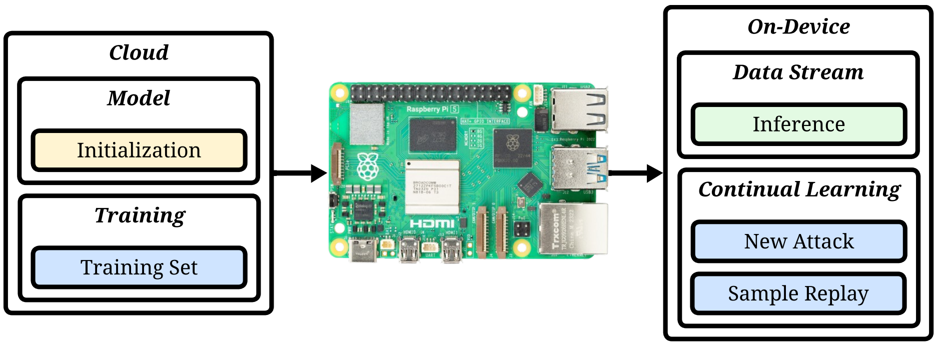


Fig. 4.1 Overview of the proof-of-concept continual learning system.

4.2 Anomaly Detection in Power Consumption

In this research (Figure 4.1), the system comprises of three sections: (1) initial model initialization, (2) model deployment, and (3) continual learning workflows.

4.2.1 Power Consumption in Electric Vehicle Charging Environments

Attacks on electric vehicle charging environment can be reflected in the power consumption of components within the environment. We explore this with the CIC EV charger attack dataset 2024 (CICEVSE2024) dataset [22]. Specifically, we utilize the power consumption data of the component "EVSE-B". In their setup, this component is represented by a Raspberry Pi 4 model B (Quad-core Cortex-A72 (ARM v8) 64-bit processor, 2GB RAM, running Ubuntu 22.04.3 LTS (Jammy), and utilizing the Linux 5.15.0-1044-raspi kernel). A Raspberry Pi is used as its features align with the essential requirements of typical EVSE hardware alongside having a track record of successful implementation in various commercial EVSEs and for its accessibility and replicability.

Table 4.1 Power consumption features of EVSE-B from the CICEVSE2024 dataset.

Feature	**Description**
Time	Timestamp of the sample
Shunt_voltage (mV)	Voltage drop across the shunt resistor in the I2C wattmeter
Bus_voltage (V)	Measured DC supply voltage
Current_mA	Current consumption of EVSE-B
Power_mW	Power consumption of EVSE-B

Typically, the power consumption of the SECC is not tracked by the CSMS, however, the authors state that it may still indicate compromise. Table 4.1 lists the features in the power consumption set. From this, we extract the shunt voltage, bus voltage, current, and power then combine it with the state the vehicle is in (charging or idle) and the interface (e.g. OCPP, ISO15118). Therefore, each sample contains 6 features, producing an input of $\mathbf{x} \in \mathbb{R}^6$.

Three attacks are included in this dataset: Reconnaissance, Denial-of-Service, and Host-Attack.

Principle Components. To better understand the data and its features, we analyze the data with Principle Component Analysis (PCA). In order to plot the data, allowing it to the visualized, we reduce the dimensionality of the data from 6 dimensions to 2 dimensions. In Figure 4.2, we observe that the host-attack and normal samples may have similar representation in power consumption features when compared to normal and recon with what can be considered some overlap in the cluster(s). On the other hand, DoS and Recon have high separation when compared to the normal data.

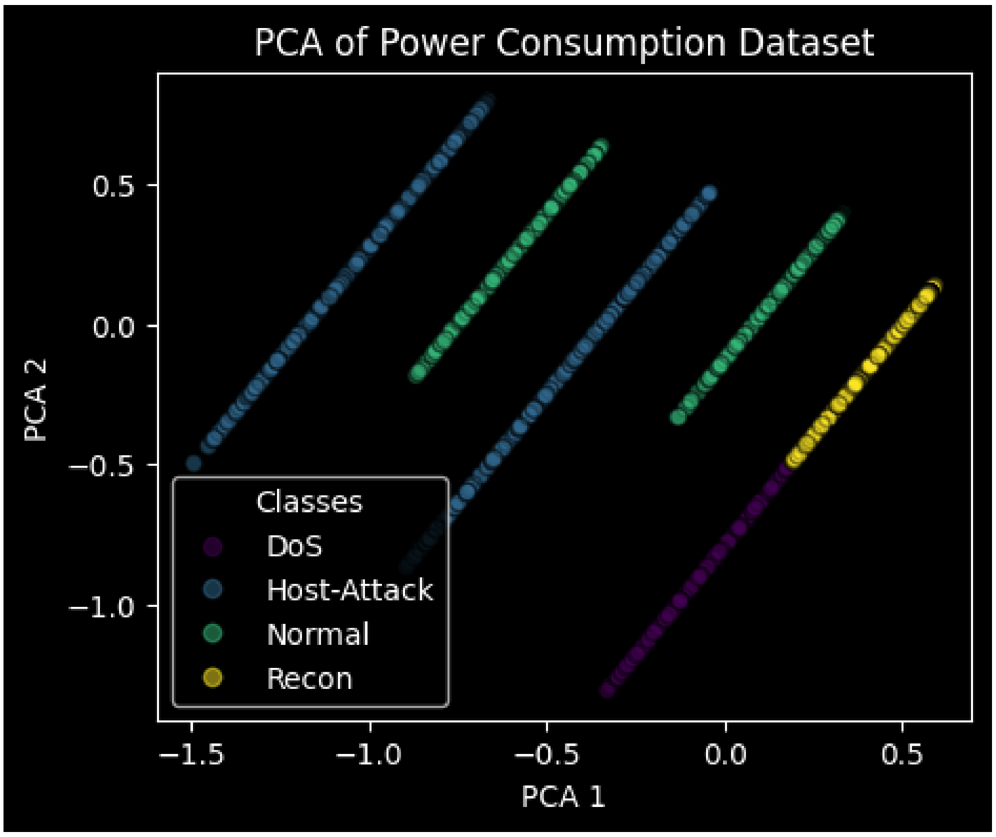


Fig. 4.2 Principle component analysis of the dataset.

Chosen Attacks. To evaluate the proof of concept, two out of the three attacks are chosen. These two attacks resemble the original task and the new task. In addition to the scope defined in section 1.4, the PCA analysis shows that the Recon and Denial-of-Service attacks have similar representations while the Host-Attack may provide a more difficult task as it overlaps with some of the benign data. Therefore, DoS samples are chosen as the original task the model is trained on while the Host-Attack samples provide a new task to enable continual learning.

4.2.2 Anomaly Detection

To explore and demonstrate continual learning in this context, we train an autoencoder on attack data. With this, it can be analyzed how the new task affects the model's learned representations based on the reconstruction losses.

Reverse Anomaly Detection. In a typical autoencoder used for anomaly detection, only the normal data is used for training. Therefore, when a calculated threshold is applied, any inferred samples that produce a reconstruction loss above the threshold is an attack. To enable training on new attack tasks, we reverse this and train only on attack data. Therefore, anything below a given threshold is an attack. While the traditional benign-only autoencoder approach has its benefits, such as not requiring hard-to-acquire attack data, this reverse approach allows us to demonstrate continual learning with an alternative method. Training only on benign data may not warrant a new task-based continual learning strategy, but may rather represent online learning adaptation. Therefore, by training on attacks, the new task learning can be demonstrated. While a classifier may also be suited for this, it may involve incremental learning where new classes are created by increasing the number of output neurons for each new task. This can be considered both out of scope and future work.

The autoencoder leverages a fully connected architecture of which transform the 6-dimensional input to a latent space of size 2 and then to a 6-dimensional reconstruction of the input. Each intermediary fully-connected layer outputs into a Rectified Linear Unit (ReLU) activation function. The model outputs through a sigmoid activation function. Overall, the model's forward pass is $(6, \mathrm{ReLU}) \rightarrow (4, \mathrm{ReLU}) \rightarrow (2, \mathrm{ReLU}) \rightarrow (4, \mathrm{ReLU}) \rightarrow (6, \mathrm{Sigmoid})$. A fully-connected architecture is chosen as the data lacks temporal dependencies between samples. Additionally, the specific width and depth is chosen for suitability as the data can be considered easily separable. Alongside this, a low parameter count with minimal computation, such as FLOPs, promotes light deep learning.

Autoencoder Training. Initially, the model is trained on one attack type for 10 epochs a a learning rate of 0.001 using MSE loss and the Adam optimizer. The dataloader uses a batch size of 124 where the training and validation set have a 50/50 split, leading to a 17442 sized training set and 17442 sized validation set. To find the reconstruction loss, the mean-squared error (MSE) is taken between the input sample and the model output. If the input is a batch, the resulting batch of reconstruction losses is averaged. To find a suitable threshold for the model, we apply a grid-search across the range of reconstruction losses present in the test set.

Learning New Tasks. Given two attacks, attack A and attack B, learning new tasks works as follows. The initial model is trained on only attack A. In a new training session, the same model that is optimized for attack A is trained on an additional attack, attack B. With this, we aim for the model to successfully learn the representations of both attack A and attack B. In our setup, the model is initially trained on the DoS attack set, as previously described. It is then trained on a new attack, for example, host-attack. However, as the model optimizes for host-attack in the new training session, its weights for DoS will be overwritten in place of the new weights designed for host-attack. The model may no longer be suited for reconstructing DoS attack. For this, the replay of previous samples must be implemented. At each epoch, the continual learning training loop goes through the mini-batches of the new training set. Before the forward and backwards pass, a random replay set batch is loaded in and merged with the current mini-batch. The remainder of the training loop follows the standard training methodology where inference is conducted and the loss is calculated and backpropagated with respect to the entire mini-batch.

The continual learning hyperparameters match the initial training's hyperparameters with a batch size of 124 for the dataloader and a learning rate of 0.001 where MSE loss and the Adam optimizer are used. Again, 10 epochs are used. While the dataloader's batch size is 124, this is for the new task data. The total batch size for the continual learning includes 41 new task samples (1/3 of the batch size). Therefore, the total batch size for continual learning is 165.

Replay for Catastrophic Forgetting. As the continual learning optimizes the model for a new task, the original weights that were optimized for another task are overwritten. Therefore, the model is no longer suitable for the first task. To mitigate this, we employ the replay of old samples seen in prior training sessions. The replay technique essentially "reminds" the model of what it has already seen, ensuring that it stays optimized for the initial task while improving for the new one. A set of the initial training samples are stored within a "replay set". These samples are not selectively chosen and are a subset of the training set. During continual learning, a random subset of size 41 is chosen from the replay set and concatenated onto the new task training mini-batch.

Deployment. The model is deployed onto a Raspberry Pi 5 which features 8GB of RAM and a quad-core ARM Cortex-A76. For deployment, the full Python PyTorch library is used. The device presents sufficient resources for the full model and the PyTorch interpreter. In this deployment, all layers remain trainable.

4.3 Continual Learning Workflows

Two workflows are implemented in order to evaluate and justify a local-only system. They both compare a standard continual learning sequence on two devices. These workflows are referred to as "Edge-Cloud" and "Edge-Only". With these methods, we evaluate the continual learning in a scenario where a deployed model is trained on a set of attacks. A subset of training data is deployed with the model onto an edge device. During operation, the system collects non-benign samples with an undefined method, e.g. cloud intervention or a verification model. Once sufficient samples are collected, continual learning is triggered with the new attacks and the replay of old samples stored in the system. After continual learning, a subset of the new training samples are stored in memory alongside the original training samples, creating an updated replay set.

Edge-Cloud. To reduce the need for consistent cloud communication, which introduces increased overhead, latency, and privacy concerns, the edge device conducts inference locally. The cloud then handles the continual learning, reducing communication to training sample upload and model deployment. Both of these actions would not typically be negatively affected by increased overhead. In this workflow, a pre-determined "new task" training set is communicated from the edge device to the flask server. Mimicking the cloud, the flask server stores these samples. Alongside this, a set of pre-determined old seen samples are loaded into the replay set. After the uploading of the "new task" training set, the edge device triggers training on the flask server. The cloud device conducts its GPU-powered training. While this occurs, the edge device is free to continue inferring on new samples.

For the "cloud", a flask server is setup on an RTX3070 system, which enables GPU-powered PyTorch training. Communication between this flask server and the edge device includes uploading samples, triggering training, and deploying the new model. On the edge device, PyTorch forward pass process conducts inference on a stream of samples. The reconstruction loss is calculated with the MSE loss function, and compared to a threshold. In this, the inference time is calculated. The backwards pass leverages PyTorch's automatic differentiation engine with the gradient descent optimizer

Edge-Only. In this workflow, the cloud server is removed from the system and the Raspberry Pi 5 handles both inference and training locally. Inference is conducted sequentially on a set of samples. The device collects samples and triggers CPU-powered training. Inference is continued on another thread while the local training occurs. A Raspberry Pi 5 presents an edge device which can handle a deployment setup using PyTorch due to its RAM constraints of 8GB. Therefore, it may even handle larger networks for more complex tasks in non-time critical use cases.

4.4 Evaluations

4.4.1 Local Model Performance

Two model results are collected with the first coming from the initial model trained on the first attack and the second coming from the "new task" model where it undergoes continual learning on a new attack set.

Single-Attack Performance. When trained on an initial attack of DoS, the distribution of attack sample reconstruction losses and distribution of benign sample reconstruction losses do not overlap and are clearly separated. A threshold on this model is capable of detecting all attacks with no false negatives or false positives. This can be observed on the PCA analysis, where DoS and Benign data show clear separation on the principle components of their features. Based on the learning curves (Figure 4.3), the model converges after the 6th epoch. Additionally, the learning curves do not diverge from each other and the validation loss closely follows the training loss.

CL No Replay Performance. To demonstrate the necessity of mitigating catastrophic forgetting, the continual learning is evaluated with and without the replay of old samples. Catastrophic forgetting can be observed in Figure 4.5 as the model's weights based on the original training set has been overwritten in turn for the new training set. It is no longer suitable for detection DoS samples, but it can detect Host-Attack samples as it successfully learns its representation.

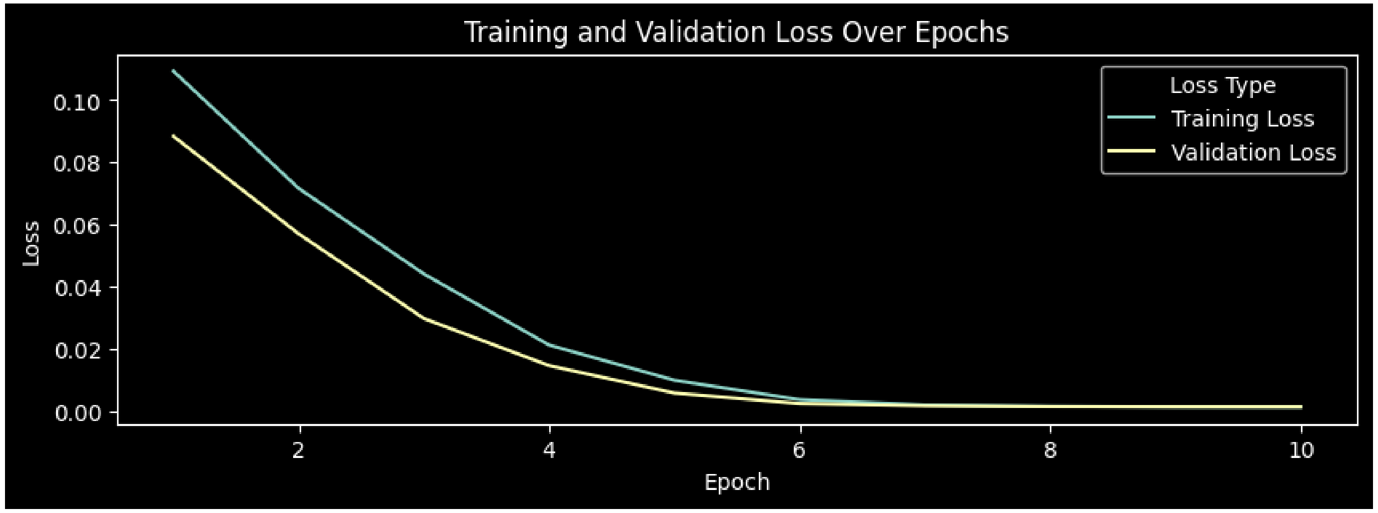


Fig. 4.3 Learning curve of the initial DoS training.

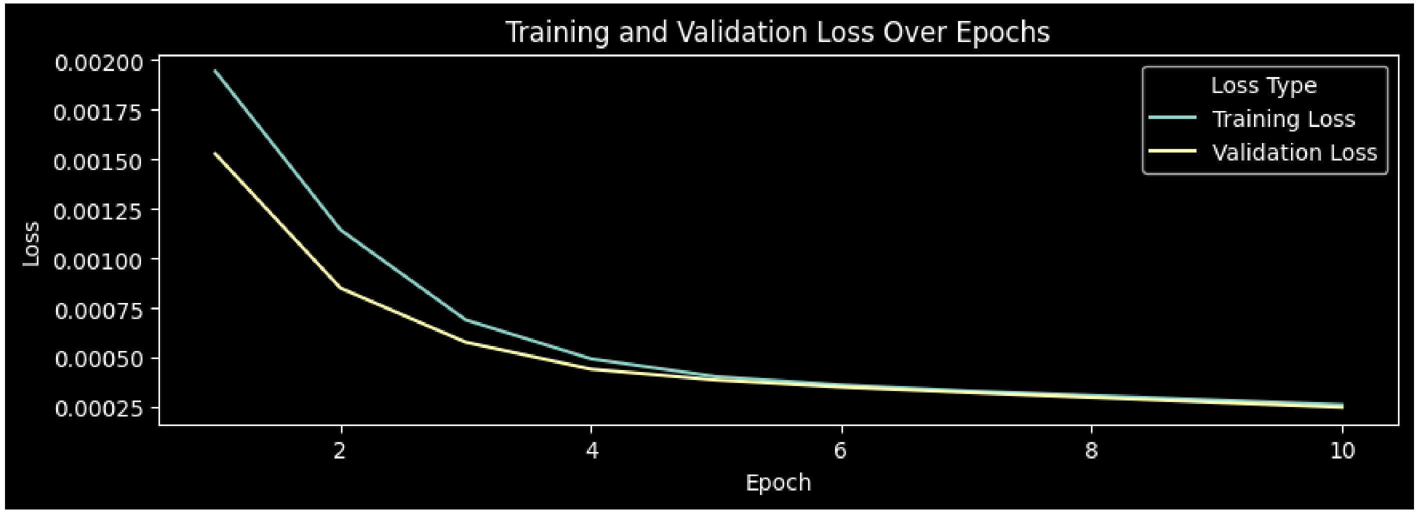


Fig. 4.4 Learning curve of new attack CL with no replay.

When applying continual learning with no replay, the loss decreases (Figure 4.4) as the model learns to reconstruct the new samples. The training and validation loss remain similar until training stops at above 0.00025 loss after 10 epochs. From this, it can be judged that the model successfully learns how to reconstruct host-attack samples. However, the reconstruction losses on the test sets (Figure 4.5) show that the model has effectively forgotten how to reconstruct the DoS samples. This can be seen as the DoS samples are above the threshold and on the right side of the plot with high reconstruction losses. Additionally, it can be observed that there is an increase in false positives, matching up with what was judged by the PCA. Here, there is a cluster of host-attack samples that fall above the threshold alongside normal samples.

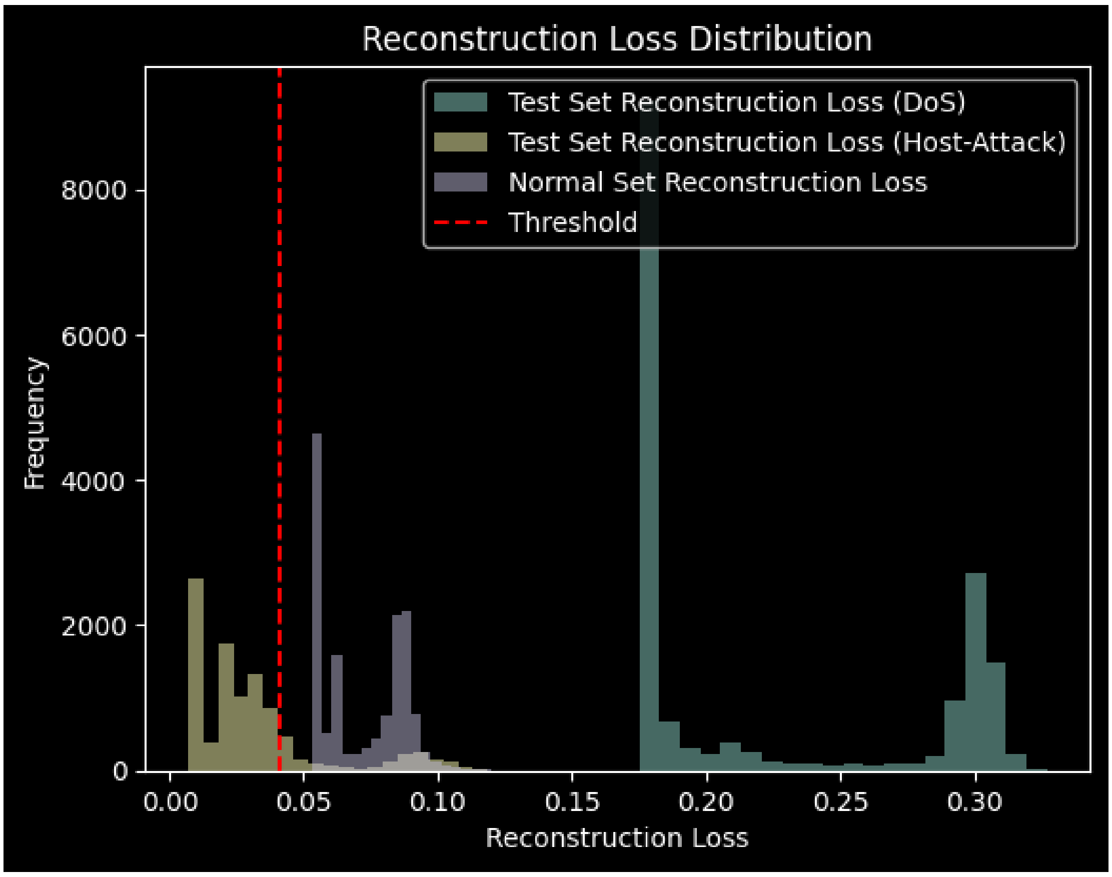


Fig. 4.5 Reconstruction losses of new attack CL with no replay.

CL With Replay Performance. With the replay of samples from the initial training set, catastrophic forgetting is mitigated. The model can be considered optimized for both DoS and Host-Attack. This is despite a lower proportion and variety of DoS samples in the replay set when compared to what would be included in the Host-Attack set and when compared to compared to the initial training set.

4.4.2 Workflow Computational Performance

Due to the resource constraints of edge devices, training time, inference time, and model size are key computational metrics for suitable operation. As inference is conducted on the edge device in both workflows, the inference time remains consistent in both workflows. To not provide repetitive results, the inference times during the edge-cloud workflow is generally omitted, as seen by the missing data in Table 4.2.

Table 4.2 Average inference times, training times, and inference while training times for both workflows.

	Average Time (ms)			
	Inference	Training (batch)	Inference w/ Training	Training w/ Inference
Edge-Cloud		0.783 ± 0.042		
Edge-Only	0.537 ± 0.039	2.141 ± 0.561	0.788 ± 0.339	2.831 ± 0.831

4.4.2.1 Edge-Cloud

The edge-cloud setup, with the cloud-based GPU, conducted training at a processing time of 0.783ms on average per batch with a batch size of 124 in the main training loop. As training is conducted on the cloud, inference during training on the edge device remains consistent with the general edge inference time. This inference time is described in the next section (see 4.4.2.2 Edge-Only).

4.4.2.2 Edge-Only

On the same training set size and batch size as the previous workflow, the CPU-based training took roughly 2.141ms per batch, which can still be considered suitable for the task as continual learning updates can be conducted during down time and the higher training time is still relatively low. Additionally, downtime can be considered quite frequent in vehicles, especially when it comes to EV chargers. When conducting inference at the same time, the training took 2.831ms per batch.

The inference of single samples on the edge-only flow, 0.537ms on average, increased to 0.788ms on average when training was running on another thread. The former inference time, which is conducted independetly without any resources allocated for training, can be considered suitable for real-time use. However, more exact evaluations must be conducted to determine how much of an increase in time would exceed the message frequency requirements within a real network. This is especially true for the inference during edge-based training, as if the increase in inference time no longer allows the model to operate appropriately for the network, then the training must be conducted during less critical times, such as during down time. Additionally, this would determine if the hyperparameters, such as batch size, can be modified accordingly.

Ultimately, times demonstrated by the model signify that a larger and more complex model could be employed while still remaining at an operational level. Such a model could achieve better performance results as it better captures the relationships in the data, as well as have the capacity to handle more new tasks over time.

4.4.2.3 Post-Workflow CL Performance

The continual learning in both workflows utilized identical hyperparameters, training set, and replay set. Therefore, the models produced by both workflows followed the same learning path. This can be observed in Figure 4.6, where the reconstruction loss figures of both models are identical. The model was able to detect 99.21% of all attacks correctly alongside a false positive rate of 0.75%

4.5 Observations and Discussion

Based on the results, the model may benefit from a more optimal replay set selection technique to ensure sufficient representation of prior tasks. With this, catastrophic forgetting would be mitigated more effectively. Additionally, this would become more important if there are many prior tasks or if the prior tasks involve data with high variation. For example, a replay set may not sufficiently represent the features of a set which has variance if it has a small sample size. This may result in catastrophic forgetting of certain peripherie(s) of the class distribution.

In terms of the new training set, samples may not be collected in real-time but rather constructed and communicated to the vehicle for training by a cloud supervisor system. For example, if a new attack pattern was discovered. If the edge system is to be fully independent, improved sample selection for the new task training set would

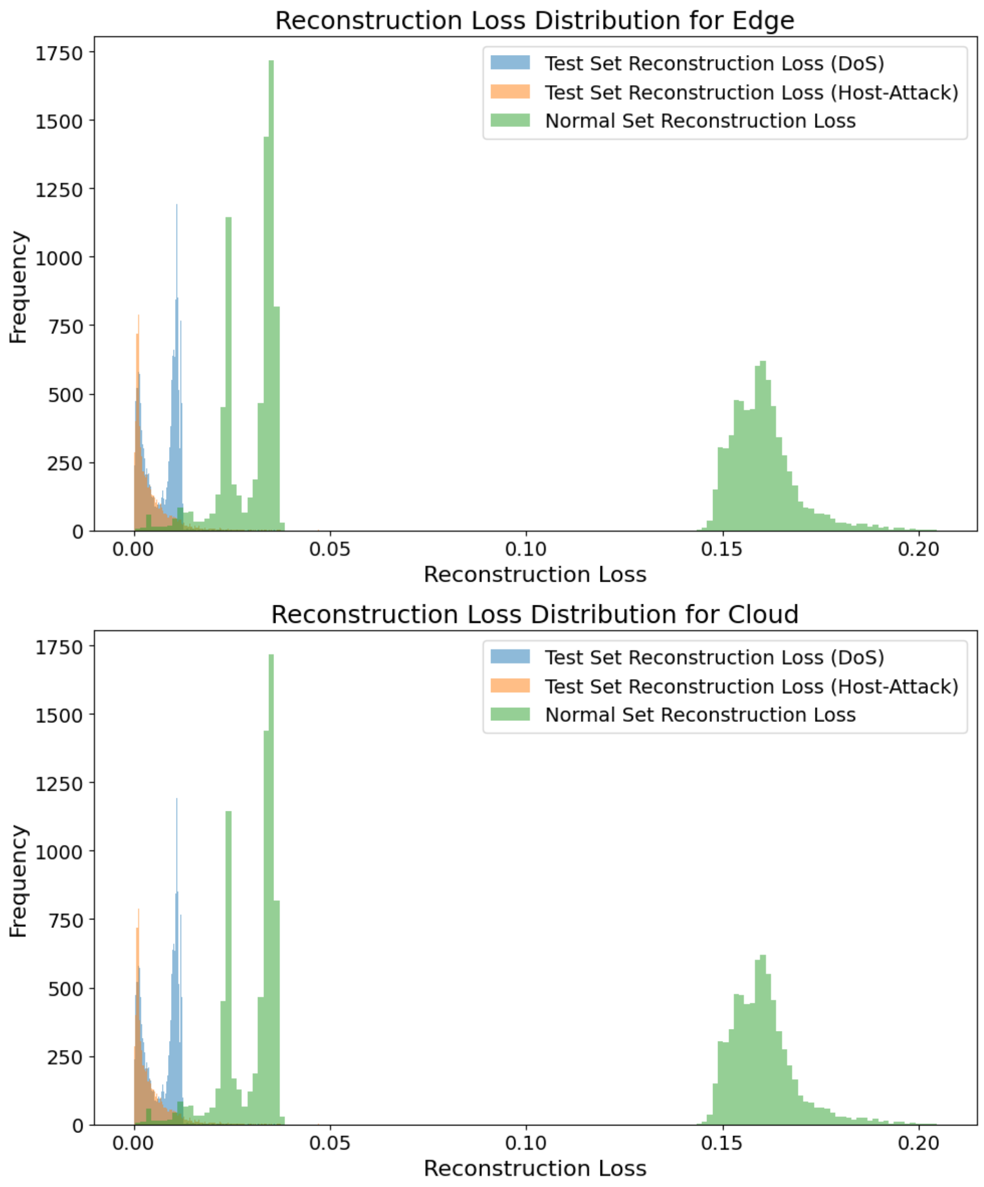


Fig. 4.6 The identical reconstruction losses of the model produced by both workflows.

improve model performance. This may include clustering methods to statistically determine what is not in the distribution of benign samples.

The adaptation process can be enhanced by involving updates from other infrastructure systems. For example, the continual learning can be combined with federated learning. Alternatively, a more diagnostic approach may be more beneficial, therefore, the continual learning framework can be modified to support classification. This may then involve incremental learning where new output neurons are added. The timing of training can also be adjusted, for example, EVs and their infrastructure are not always charging, allowing for a lot of downtime. Therefore, down-time based training, rather than real-time, can allow for the use of more complex models or more strategic training.

The continual learning poses three questions: (1) How many more attacks can the model adapt to? (2) Will the performance begin to deteriorate when the parameters must be optimized for reconstructing too many types of attacks? (3) Will a more complex model then be better suited and have more space for improvement? Future work should explore the trade-offs within these questions, such as model complexity, efficiency, and performance across many tasks over time.

Lastly, the Raspberry Pi 5 can be considered more than sufficient for the task. Therefore, future work should involve more resource constrained devices, such as an Arduino platform, to determine the applicability of the system under different circumstances. Additionally, it would allow an evaluation to determine with more depth how continual learning only on the edge compares to the GPU-enabled cloud. For example, in low-resource hardware, the current batch size may not be suitable. Therefore, stochastic based CL or very small mini-batch sizes would be required, which would impact the capability of model learning. Furthermore, the size of the replay set and collected sampled for the new task may be limited due to memory limitations.

Chapter 5

Adaptive Intrusion Detection on the Edge for the CAN bus

This chapter presents the second contribution. This is the main contribution of this thesis.

5.1 Introduction

Modern machine learning and computing often finds itself in repeating issues of privacy, computation power, and latency. This is especially true in naturally remote and mass produced physical use cases. Vehicles are always on the move and consist of a time-critical network, the CAN bus, which enables the communication of almost all the components within most modern vehicles. However, the CAN bus is not secure and is vulnerable to external threats. Vehicles, which can be viewed as dangerous weapons when wielded incorrectly, therefore, can have fatal consequences at worst and financial issues at best if targeted by an attack. Jeep initiated a vehicle recall after researchers discovered vulnerabilities when targeting a 2014 Jeep Cherokee [10] [10] where a compromised CAN allowed them to control critical components. Thieves can also use the CAN to gain access to vehicles with blackmarket-bought technology [9].

To address this apparent issue, researchers have investigated machine learning intrusion detection systems (IDS). From driver identification with GANs [38, 39] and LSTM-convolutional hybrid models [37] to attack detection with traditional machine learning [27–29] or deep learning [33, 34, 32], it has been proven that ML-based IDS solutions are capable at defending the CAN bus. However, the CAN is time-critical and typically contains devices, such as ECUs, which have resource constraints. Therefore, many proposed solutions are not suitable for locally deployed systems due to model size and therefore require the cloud, which introduces latency. More efficient and deployable IDS solutions have been explored [43, 41, 44, 42] to bridge the gap and get closer to real-world-ready local systems.

However, initially trained models are not guaranteed long-term suitability, as changes in data are possible due to road types, different drivers, or change in components. Alongside this, some tasks like driver identification, require continuous personalization. This requires online learning (OL), a method where a model continues to be trained once deployed. Resource constrained hardware are typically not suitable for model training, and may require even smaller models. To address this, we demonstrate a TinyOL-based [16] IDS for the CAN (see Figure 5.1). By introducing this existing online learning framework into CAN bus intrusion detection, the system can conduct both anomaly detection and adaptive updates on device, fortifying the vehicle against current and future threats.

In this chapter, we explore on-device online learning at the edge for the CAN bus to enable an adaptive intrusion detection system. For this, the system consists of a tiny autoencoder, capable of efficient inference on resource-constrained hardware. To demonstrate this, the model is deployed onto three heterogeneous edge devices. We evaluate the OL with different OL workflows with both benign data streams and attack-contaminated data streams.

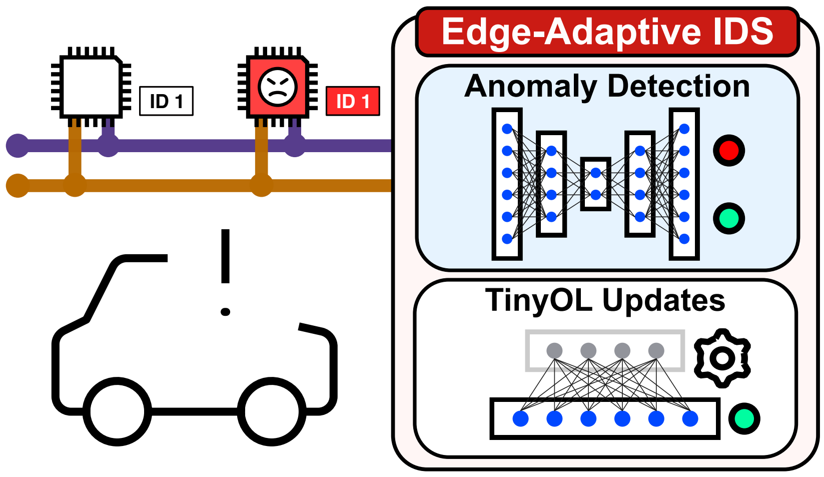


Fig. 5.1 A brief overview of the concept.

5.1.1 Contributions

To answer the main research questions presented in section 3.2, this main work presents the following contributions:

- **TinyML for Temporal CAN Patterns.** Three small autoencoders with different architectures (dense, conv1d, and conv2d) are trained on three different CAN IDs (speedometer, gear shifter, and steering angle) that would typically have temporal features.

- **Edge Deployment.** With Arduino and Raspberry Pi platforms, the models are deployed with different methods. Through this, we evaluate model deployment suitability with computational performance metrics.

- **Online Learning.** An efficient online learning (OL) technique designed for TinyML is adopted for the CAN bus use case. With an early-stage model variation, the OL is conducted locally to the evaluate the capacity for model adaptability and improvement post-training.

- **Test Rig Observations.** Additional insight on deployment is observed through test rig-based vehicle simulations. For this, the original model architectures are trained on the test rig's speedometer data to obtain relevant observations on real time inference and message frequency to determine model suitability.

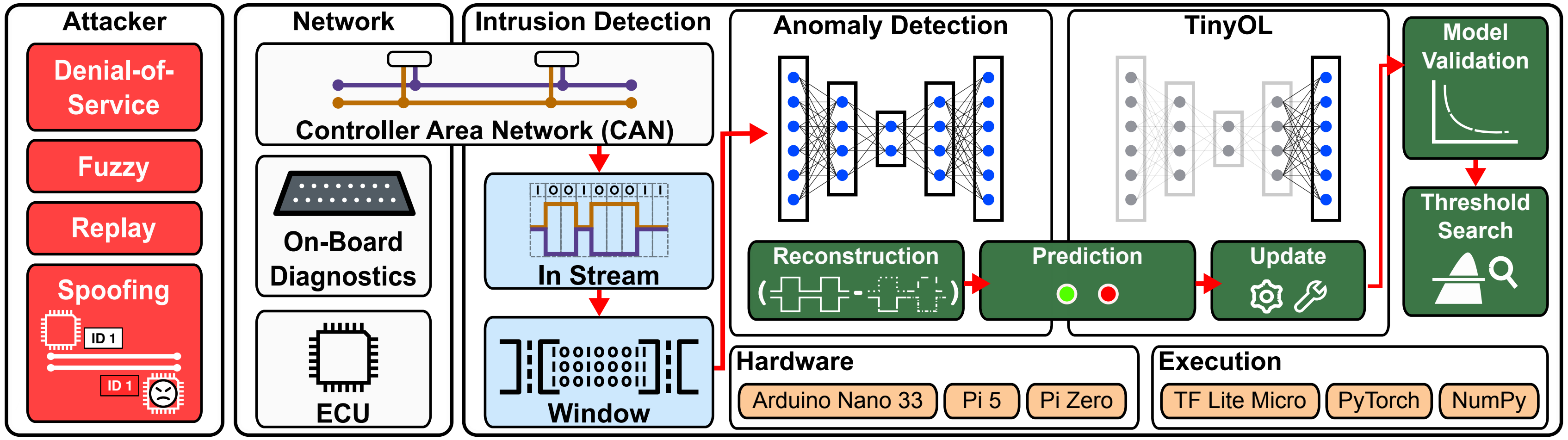


Fig. 5.2 An overview of the chapter.

5.1.2 System Overview

This research is split into three sections: (1) preprocessing CAN bus data and training a small model, (2) deploying the models onto three heterogeneous devices with three different methods, and (3) a local evaluation of OL on the model and its on-device computational metrics. Figure 5.2 provides an overview of the chapter and the proposed proof-of-concept system. From left to right, it presents possible attacks CAN attacks that can target the network. This is then followed by the flow of the system where the incoming CAN stream is windowed. Inference and the TinyOL framework is then applied. Deployment hardware and their execution are also listed.

It is important to note that this piece of work is not aiming to provide benchmark results in comparisons with other CAN IDS approaches that propose machine learning methods in an attempt to achieve optimal performance metrics. Rather, we demonstrate a proof of concept and discuss the possibilities of resource-constrained deployment alongside on-device OL.

5.2 CAN Bus Data

This section describes the approach used to handle the CAN data, such as information on the utilized data and any feature extraction.

Open Dataset. Accurate CAN bus data requires a real vehicle. For this, we use the open CAN-MIRGU [19] dataset which comprises of real-world CAN traffic collected from a fully electric vehicle equipped with autonomous driving capabilities. CAN messages were recorded at 500 Kbps using SocketCAN utilities and a Kvaser Memorator 2xHS v2 device connected directly to the vehicle's CAN gateway, ensuring comprehensive access to high-speed CAN communications. The dataset encompasses 36 attack scenarios, including 26 real injection attacks and 10 simulated suspension and masquerade attacks, targeting 13 out of the 56 unique CAN IDs present.

Extracted Datasets. Many parts of the CAN bus provide temporal patterns over a series of messages. For example, a window of speedometer information shows the change in speed over time. The same can be applied to the gear shifter, as a window of gear shifter information shows the change in gears over time. Of course, the presentation of this information in different CAN systems can vary, however, the temporal patterns would still remain present. We specifically target the defense of the

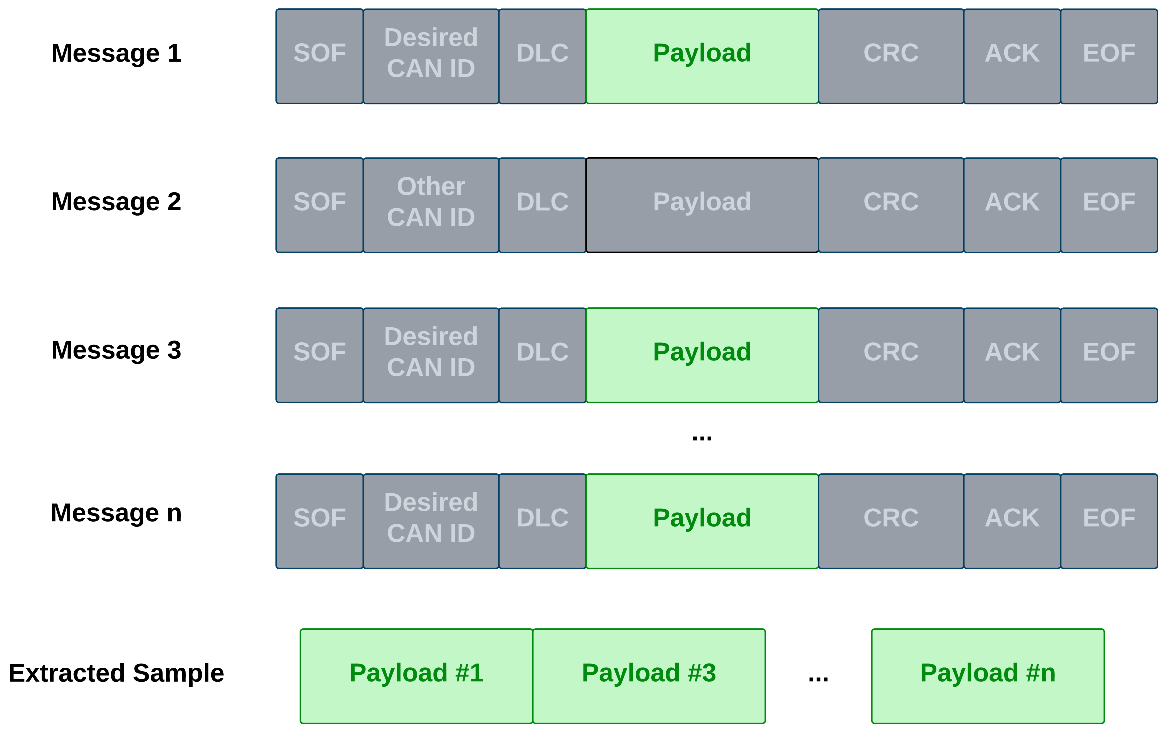


Fig. 5.3 An example CAN stream and the feature extraction process. All frames not of the desired CAN ID are ignored to create the specific dataset of a CAN ID. For frames of the desired CAN ID, the payload is collected and appended to prior payloads to form a window of length 5.

temporal-based CAN features including the speedometer, gear shifter, and steering angle (Table 5.1). To capture the changes over time for our model, each CAN ID's payload is extracted into its own dataset. Then, to capture the change over time for our models, the datasets are organized into overlapping windows. For example, this results in a dataset with only speedometer payload data where each sample is a window of payloads. With this method, we pursue a model-per-CAN ID approach. Table 5.1 presents the three CAN IDs used in this work. Each one contains its own set of attacks. The speedometer CAN ID features attacks that attempt to minimize or maximize it. The gear shifter is attacked with replay attacks. The steering around is attacked with both replay attacks and a "steering attack".

Data Preparation. The CAN-MIRGU dataset provides sets of attack-free fully benign data streams. We combine two sets of benign data stream, recorded on the same day, for the model's training. To provide more interpretable and optimized features for the model, the payloads, which are initially presented as bytes, are converted to decimal. As there are 8 bytes, this results in 8 features per message. Therefore, windowed

Table 5.1 The CAN ID components extracted from the CAN MIRGU dataset.

System Component/CAN ID	Attacks
Speedometer	Max speedometer, Min speedometer
Gear Shifter	Replay Attack
Steering Angle	Replay Attack, Steering Attack

samples are of shape $w \times 8$ where the window size w is set to 5 in this study. Following this, the 5×8-shaped decimal samples are normalized using a MinMaxScaler which is fitted to the benign training sets. A 80/20 split is applied to create a training set and a validation set. For testing, the corresponding CAN ID attack files are organized into their own test sets. The same preprocessing is applied with conversion to decimal followed by splitting into windows and transformed by the scaler which has now been fit to the training data.

5.3 Anomaly Detection

In deep learning intrusion detection systems, real attack data for training is typically not available for the following reasons. First, attacks can be considered rare when considering how many vehicles exist. Secondly, attack data from an attacker would be difficult to collect as attacks are unpredictable. It would be required to constantly store CAN data and then find which messages correspond to the attack. Lastly, CAN systems vary depending on the vehicle type or manufacturer so attacks on one vehicle are unlikely to work for another vehicle, which makes it more difficult to have attack data for a desired vehicle. Taking this issue into account, we adopt an autoencoder for anomaly detection. As an autoencoder learns to reconstruct its input, it can be trained only on benign data. When deployed, if an attack, or any abnormal sample, is encountered, the model would not have been trained to reconstruct it. Therefore, its reconstruction loss would be outside the normal range for benign samples. If a detection threshold is calculated, the attack sample would be above the threshold, triggering a detection alert.

In this study, three architectures are looked at and trained on three separate datasets, one of each CAN ID. This produces nine total models. The architectures include a fully-connected autoencoder, which can also be referred to as the "dense" model, alongside 1D- and 2D-convolutional autoencoders. To enable such a system where there can be multiple models, deployed models must be small and efficient. This is especially true on resource-constrained edge devices. Therefore, each model is kept to a low parameter count.

5.3.1 Fully-Connected Autoencoder.

We utilize a tiny fully-connected autoencoder (see Figure 5.4) for anomaly detection of CAN bus attacks. As the architecture consists of fully-connected layers, we flatten the incoming windowed samples into a 40-dimensional vector. While the data is temporal, a fully-connected architecture is used as it may still capture useful features due to the simplicity of the sequential data.

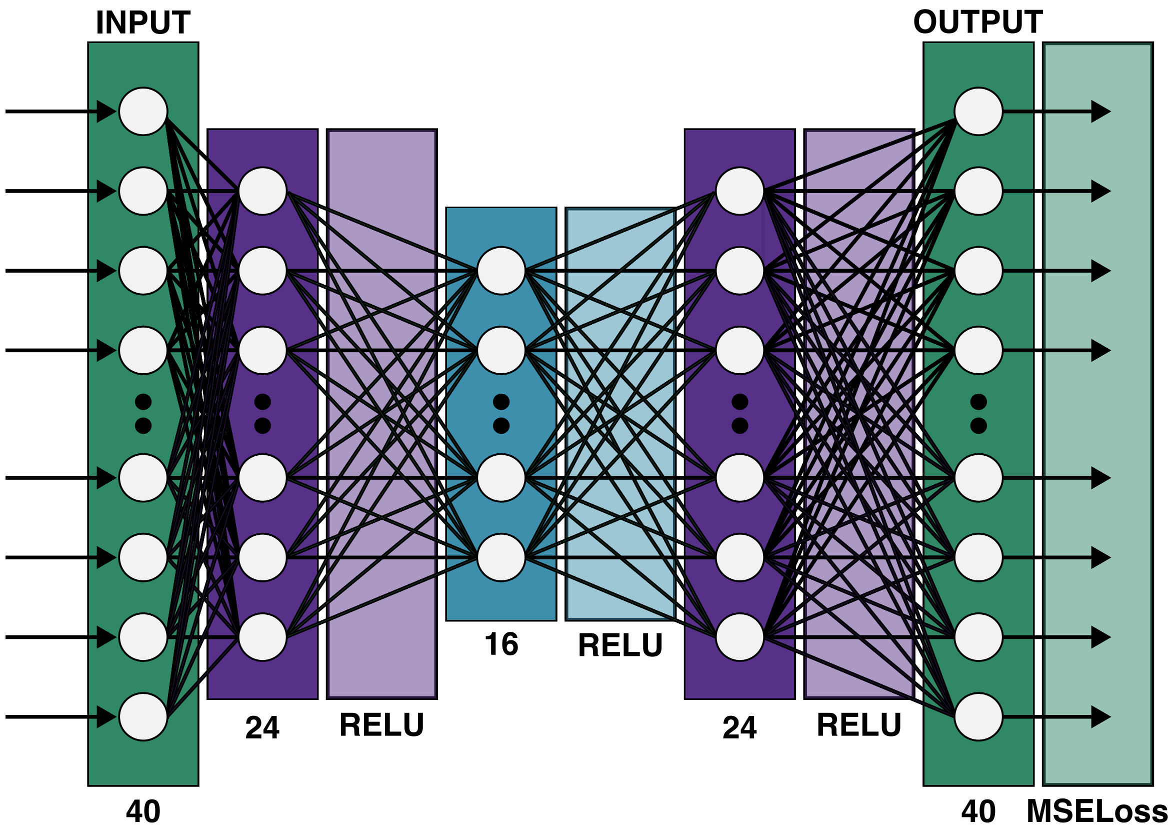


Fig. 5.4 The architecture of the fully-connected autoencoder.

With a 16-dimension latent space, the model transforms the 40-dimensional input vector to a size of 24. As the autoencoder is symmetrical, it follows another dense layer of size 24 after the latent space before being transformed back to the original size of 40 in the output layer. Therefore, it follows transformations of $(40, \text{ReLU}) \rightarrow (24, \text{ReLU}) \rightarrow (16, \text{ReLU}) \rightarrow (24, \text{ReLU}) \rightarrow (40)$. The exact architecture of the model

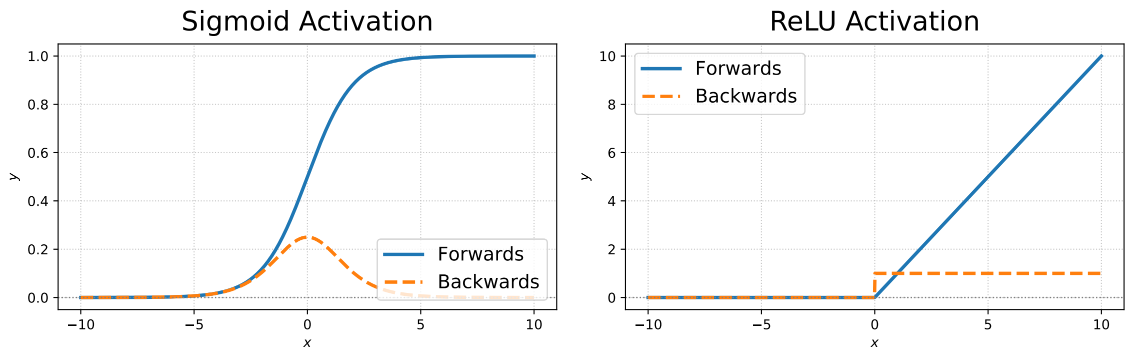


Fig. 5.5 Sigmoid and ReLU activation functions and their derivatives.

is described in Figure 5.4. As described, the intermediary layers output through the Rectified Linear United (ReLU) (Figure 5.5) activation function while the output layer does not find its product with a non-linear activation function, but rather outputs its pure logits.

Training Hyperparameters. We train the model with mean-squared error (MSE) loss where the difference between the input and output values are calculated and then squared to eliminate negative values. We train the initial model with the Adam optimizer which provides faster convergence and improved stability when compared to vanilla gradient descent. This features a learning rate of 0.0001.

Table 5.2 Fully-connected autoencoder model training hyperparameters.

Epochs	Learning Rate	Batch Size	Optimizer	Loss Function
50	0.0001	124	Adam	MSE

5.3.2 Convolutional Autoencoders.

Fully-connected models assume each input feature is independent, therefore, shared information, such as corresponding bytes across frames, are lost. To address this, and to present additional evaluations with alternative low-parameter architecture models, we train a 2-dimensional convolutional autoencoder and a 1-dimensional convolutional autoencoder.

5.3.2.1 2D-CNN

The standard 2D convolutional layer applies 2D kernels over its input. As a result, the layer expects input channels that have three dimensions, most commonly width, height, and channels. For example, an RGB image of size 32x32 would consist of three channels (red, green, and blue) where each channel has 1024 elements that are organized into a 32x32 grid.

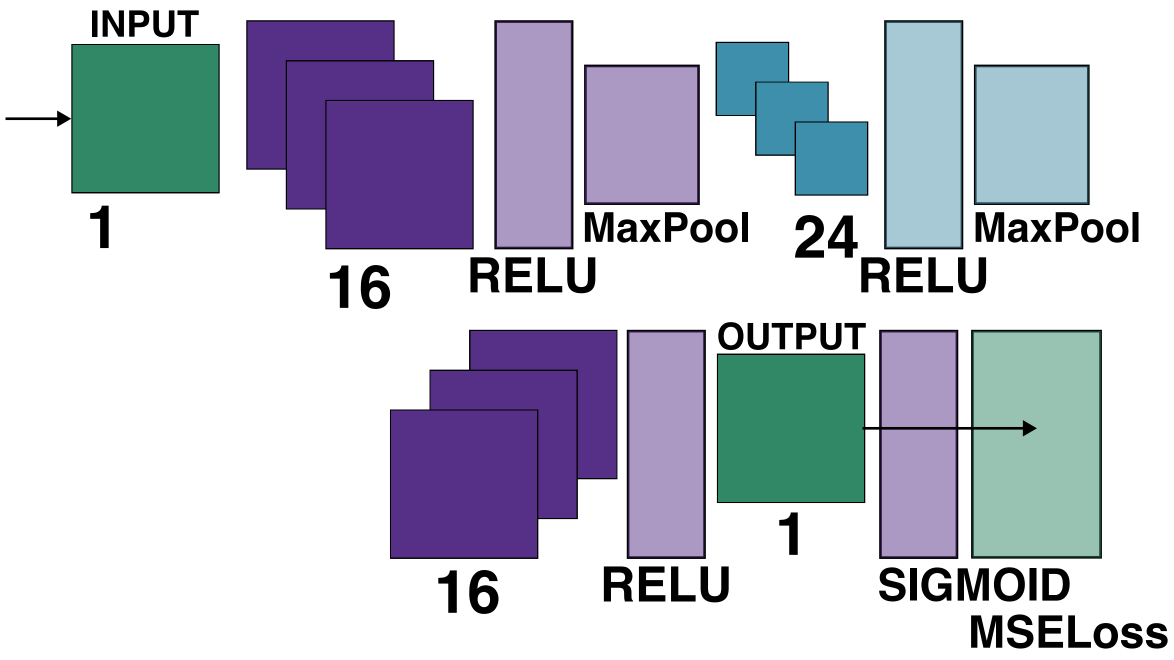
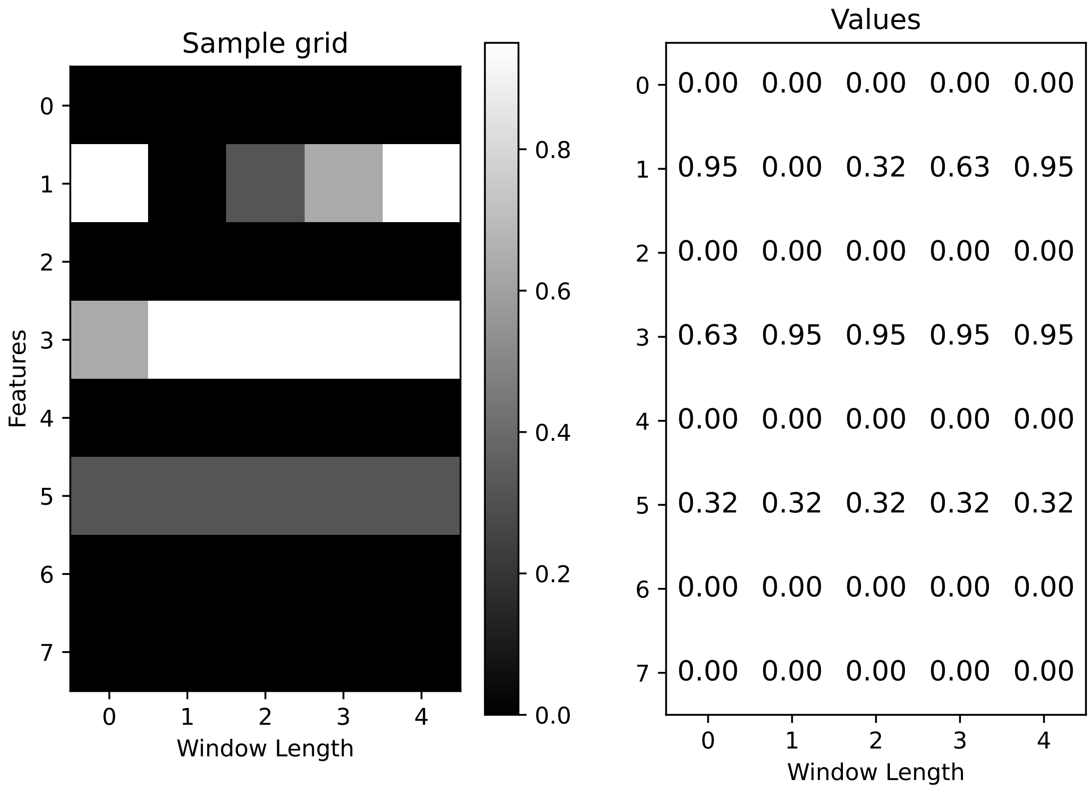


Fig. 5.6 The architecture of the 2D-convolutional autoencoder.

As we collect windows of CAN payloads, these windows can be represented as a 8×5 grid (or image), rather than a flattened to a 40-dimensional vector. To process the features of this "image", the small 2D-convolutional autoencoder contains standard 2D convolutional layers, maxpool layers, and 2D convolutional transpose layers. This is formatted in the following structure: $(Conv2d(1,16), \text{ReLU}) \rightarrow (MaxPool2d) \rightarrow (Conv2d(16,24), \text{ReLU}) \rightarrow (MaxPool2d) \rightarrow (Convtranspose2d(24,16), \text{ReLU}) \rightarrow (Convtranspose2d(16,1), \text{Sigmoid})$. The arguments in the convolutional layers represent the number of channels, whereas they represent the feature dimensions in dense layers. Overall, this model has 4457 parameters.

With this more specialized architecture, the CNN autoencoder may better extract the CAN payload's patterns through improved spatial feature extraction. As denoted by an input channel size of 1, the data is formatted into a one channel grayscale image with width and height sizes of 8 and 5. In this grid, the intensity of the pixels depend



	0	1	2	3	4
0	0.00	0.00	0.00	0.00	0.00
1	0.95	0.00	0.32	0.63	0.95
2	0.00	0.00	0.00	0.00	0.00
3	0.63	0.95	0.95	0.95	0.95
4	0.00	0.00	0.00	0.00	0.00
5	0.32	0.32	0.32	0.32	0.32
6	0.00	0.00	0.00	0.00	0.00
7	0.00	0.00	0.00	0.00	0.00

Fig. 5.7 A grayscale grid speedometer sample.

on the value at the specific position. For example, in the context of the speedometer, if a higher value is a higher speed at a corresponding byte, then if a specific payload byte's position has a lighter pixel then it is a message with higher speedometer information. Figure 5.7 shows an example of a windowed sample displayed as a grid (or grayscale image). To provide further examples, multiple consecutive samples are displayed in Figure 5.8.

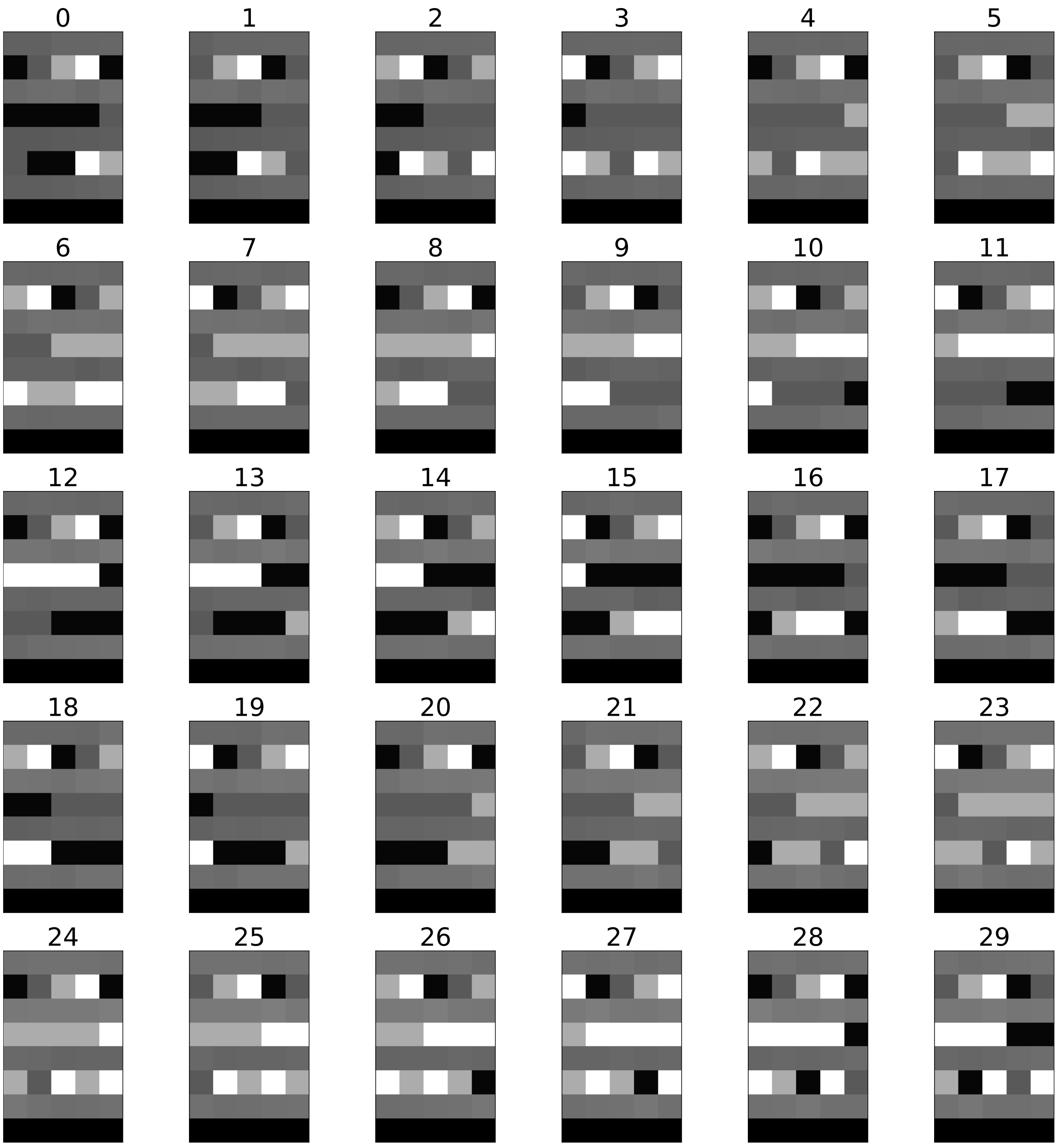


Fig. 5.8 A set of 2D-convolutional input samples.

5.3.2.2 1D-CNN

1D convolutional layers are specialized for sequential data, such as time-series data. It leverages 1D kernels to extract temporal patterns. This makes it suitable for the format of the data, where each sample is a window of 5 timesteps. Each of these window samples contains short-term temporal dependencies where changes in bytes are observed over time. For example, given an accelerating vehicle, a window of the speedometer data may demonstrate corresponding acceleration characteristics in the payload at each time point. In this case, it may be an increase in values at specific bytes.

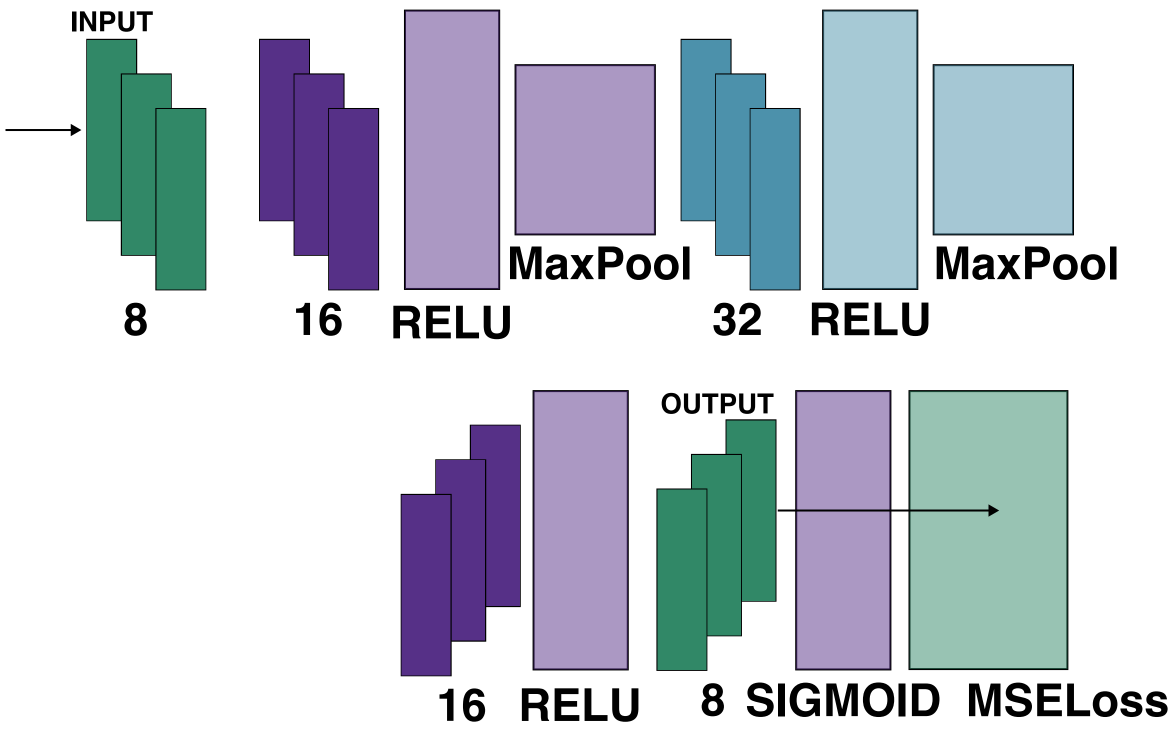


Fig. 5.9 The architecture of the 1D-convolutional autoencoder.

Our 1D-CNN accepts an input of 8 channels where the channels represent each byte. In each channel, there are five values which correspond each time point in the window. Therefore, the data can be described as 8 byte channels where each value in the channel is a point in time. If the data was instead five channels with eight time points, the model would learn the pattern from byte to byte, which would not typically contain temporal patterns as this set of bytes belong to one message at one time point. In this described situation, a maximum speedometer attack at time point 3 (of 5) in the window may not be be considered anomalous in the sequence, but rather only be detected if such a combination of bytes in one specific payload has not been seen by the model.

Overall, the model encodes the data with the following: $(Conv1d(8,16), \text{ReLU}) \rightarrow (MaxPool1d) \rightarrow (Conv1d(16,32), \text{ReLU}) \rightarrow (MaxPool1d)$. This then feeds into the decoder: $(Convtranspose1d(32,16), \text{ReLU}) \rightarrow (Convtranspose1d(16,8), \text{Sigmoid})$ This model architecture is visually presented in Figure 5.9. The encoder block's convolutional layers feature kernel sizes of 2 and padding sizes of 1 while the decoder block's convolution layers featuring kernel sizes of 2 and 3 and strides of 2. The max pool layers have kernel sizes of 2 and strides of 2. Overall, this model has 3400 parameters.

5.3.2.3 Training Hyperparameters.

The hyperparameters for training both types of convolutional autoencoders on all three CAN IDs are presented in Table 5.3.

Table 5.3 Convolutional autoencoders model training hyperparameters.

Model	Epochs	Learning Rate	Batch Size	Optimizer	Loss Function
Conv1d	50	0.0001	124	Adam	MSE
Conv2d	50	0.0001	124	Adam	MSE

Table 5.4 Convolutional autoencoders model training hyperparameters.

5.3.3 Determining Performance

Autoencoders reconstruct the inputted sample and do not naturally output any prediction. Instead, the reconstruction loss between the output and input is calculated and compared to a threshold. In this work, the model output when deployed is passed through MSELoss Equation 5.1 to find the reconstruction loss.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{5.1}$$

In this work, the threshold for each model is calculated using a linear search of a range of values from the minimum to the maximum reconstruction loss in the test set. To determine the performance of a threshold, an overall score is judged based on two weightings. The first weighting is the percentage of attacks detected, also known as the true positive rate or recall. The second weighting is the false positive rate. These two metrics are chosen as it can be argued they heavily influence the quality of a model in this use case. For example, a high false positive rate and high recall may detect many attacks, however, the model is not suitable for real-world use as there would be frequent incorrect detections, hindering a vehicles usability. On the other hand, a low false positive rate and low recall may ensure a vehicles usability, however, stealthy attacks may not be detected.

All models utilize identical hyperparameters in this search. This includes a step size of 0.00001, false positive rate weighting of 0.8, and recall weighting of 0.2. The false positive rate is given higher priority as attacks are typically rare when considering the frequency of benign messages. Additionally, false detections can be just as dangerous as attacks when considering the response that is caused by it. If appropriate, the option of setting the threshold to the lowest attack reconstruction loss also exists.

5.4 Online Learning

It has been discussed how data drift (or concept drift) from the initial training distribution is possible in vehicular systems in various ways. Based on the literature, and in preparation to assumed possible future effects, this can be caused by varying vehicle states (different drivers, road types, vehicle components). We address this by demonstrating how TinyOL [16] can be utilized for updates in a local-only IDS. In this, a model is sequentially exposed to new samples via the incoming data stream once deployed. In this OL system and the evaluation of it, we utilize our fully-connected autoencoder architecture and the speedometer data. A high level overview alongside a visual demonstration of the OL is presented in Figure 5.10. From the CAN, n payloads are collected then flattened into a vector, forming an $n \times 8$ matrix for input into the model. This model has a fully connected layer at its input. The autoencoder, trained on only benign samples, attempts to reconstruct the input. Following this, the reconstruction loss, calculated via the MSE loss (Equation 5.1 between the input sample and model output, is compared to a pre-calculated threshold. If the threshold determines that the input input sample is benign, then the model predicts that the input sample does not contain an attack message. This sample is then used for updating the last layer of the model.

For this stream-based OL, we utilize two methods: stochastic learning and mini-batch learning. In the stochastic learning, samples are inferred and their loss is backpropagated individually. However, as future samples are not predictable, allowing

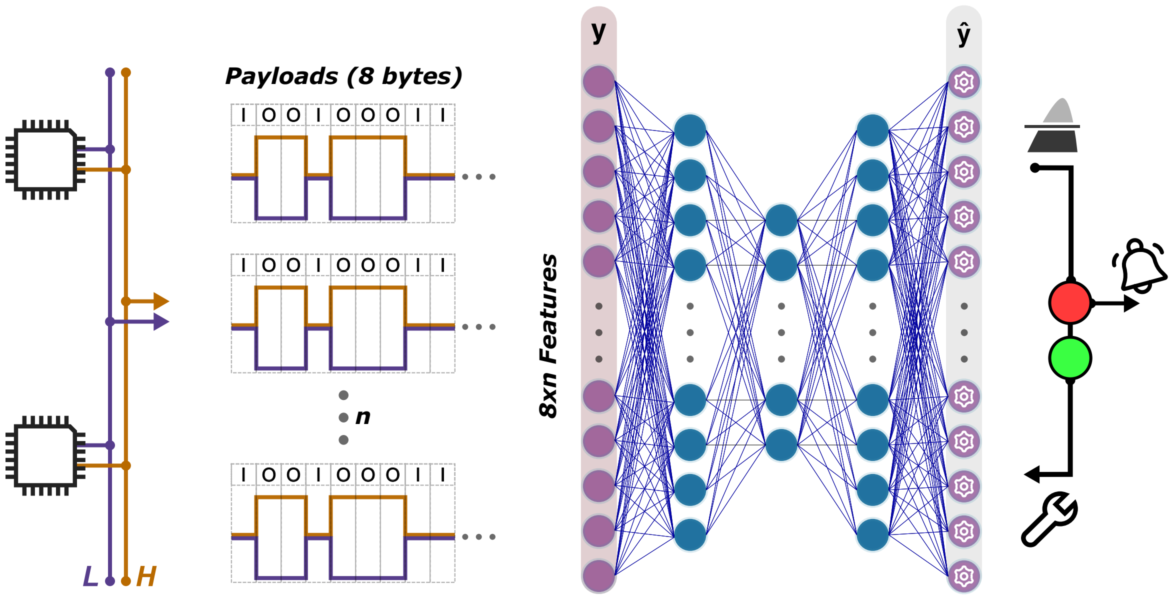


Fig. 5.10 The architectural flow of the adaptive fully-connected autoencoder.

for sensitive updates may lead to a lack of stability in the short-term. To ensure better stability from the online updates, mini-batch learning collects a set of inferred samples. Once a sufficient amount of samples are collected, the entire batch is used for training by averaging the reconstruction losses and backpropagating it through the network. For the OL, we train the last layer with the gradient descent optimizer.

However, one problem is identified with traditional OL in our use case: calculating and storing the gradients of the model may not be feasible on resource-constrained edge devices. For example, for each parameter value, a corresponding floating point gradient value must be stored during backpropagation. If the model has 2000 parameters, then it must store an extra 2000 gradients resulting in a total of 4000 stored values. On typical full desktop systems, this is negligible, as training often can contain tens of thousands to millions of parameters. However, edge devices may not have the memory limitations. As stated, we address this by demonstrating TinyOL [16] in this use case (continued in 5.4.1).

Figure 5.11 highlights the evaluation process for the OL. The fully-connected autoencoder is deployed onto the devices to measure computation metrics. As mentioned in the background (see 2.5.2), this involves the inference and training (TinyOL) times. With an early-stage model, the TinyOL model is evaluated on the local device with the OL workflows.

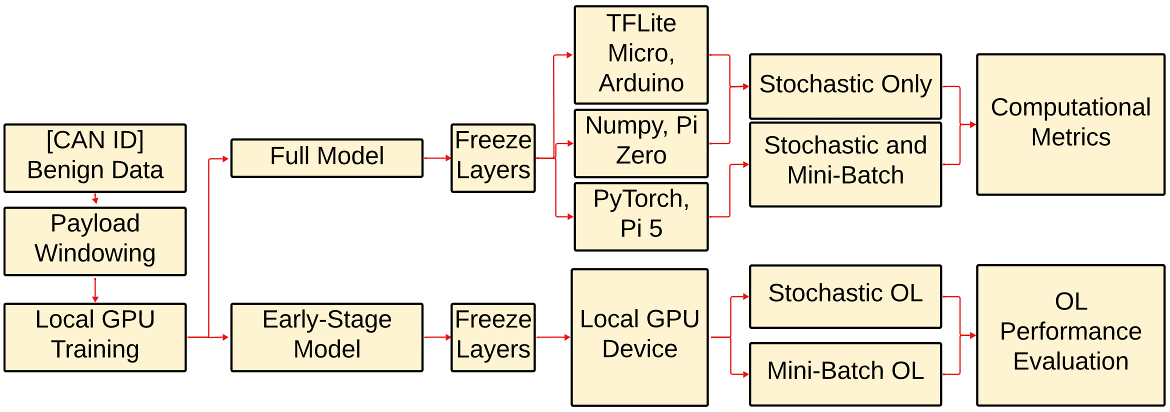


Fig. 5.11 Flowchart of the deployment and online learning setup.

5.4.1 Enabling Resource Constraint On-Device Learning

Edge devices, such as the Arduino Nano 33, have tight resource constraints. These typically include slower, lower-power processors and limited memory, which improve energy efficiency. For example, it has 256KB of flash memory, which in many use cases, is not sufficient for un-optimized models. Calculating and storing the weights during backpropagation, while the model is still loaded into memory, may not be feasible. This is also paired with increased CPU usage. Therefore, even for a small model, full model backpropagation may not be possible, or may present increased latency which cannot be afforded in some tasks. TinyOL [16] presents a solution to this by freezing all layers except for the last layer. Therefore, only the last layer is learnable and is backpropagated to. We adopt this technique and explore it in the use-case of an adaptable edge-based CAN IDS. The remainder of this section explains the OL setup.

5.4.1.1 Output-Only-Training

Zhang et al. [12] simulated many vehicles from one dataset by splitting a dataset into multiple parts. Then, the initial model is trained only on the "first" vehicle. Federated learning is then conducted with the "other vehicles". This limits the data that the model is initially exposed to. Given limitations, such time constraints and lack of resources, long-term OL with an initial converged model, or the use of OL for other use-cases, such as driver identification, cannot feasibly be evaluated. Therefore, to allow for the demonstration of the on-device OL, an early-stage version of the model is utilized. Through this, we are able to evaluate the system's effectiveness while providing a foundation for future work.

The training of the early-stage model features an initial training session with a limited starting point. This involves three changes: (1) using a smaller subset of a single benign data stream to limit exposure to samples, (2) a higher batch size to provide less sensitive and less exact gradient updates, and (3) a lower amount of epochs so that there is less time to train. This demonstrates a scenario where a model has not been exposed to enough samples yet in training and has not completely settled itself according to the samples it has been exposed to. While this model may not be deployed in this state in real-world use, and rather a converged model with the best performances would, this scenario represents the capacity for learning. This may then be extrapolated to other use-cases, such as a driver identification that has limited training data and must continue adapting, or an anomaly detection/classification model that must adapt to new data.

Training Samples. For the experiment and evaluation of the online learning, we utilize all benign-predicted samples for the updates within the OL. When online, the model infers the incoming data stream. Samples that are predicted as benign are used for training. Through this, we are able to observe how the model's learning behaves with what it judges to be normal samples. Additionally, we involve two sets of streams:

1. The first stream is a benign only stream extracted from the initial training set. As the initial training set was cut down for the early-stage model, the unused remainder of the set is used as the stream for OL. The set is processed in sequential order as based on the original recording.
2. The second stream involves test sets containing attack samples. This stream will evaluate how the model's learning behaves when attack samples contaminate the training as some attack samples may be predicted as benign messages.

Workflows. Two OL workflows, as shown in Figure 5.11, are defined in the next two sections (see 5.4.1.2 and 5.4.1.3). The first involves stochastic updates and the second involves mini-batch updates. Both are applied to the two types of OL streams.

5.4.1.2 Stochastic OL

The stochastic OL workflow involves OL based on updates of singular samples at a time. This is presented in algorithm 1, where the model infers on the incoming samples continuously. If the reconstruction loss falls below the threshold benign, and is therefore predicted as benign, the sample is instantly used for an instance of backpropagation on the output layer. This enabled by freezing all other layers. Overtime, as the model updates, the threshold may no longer be suitable. To address this, the threshold is recalculated after a defined number of updates. For example, if set to 512, the model will be reevaluated on a provided test set once 512 samples have been inferred.

Algorithm 1: Online Learning Stochastic Workflow

```
Input : CAN interface $\mathcal{C}$, Target ID $ID_{target}$, Model $f_\theta$, Initial threshold $\tau$,
        Update frequency $N$, Validation set $V$
// Phase 1:  Initialization and Layer Freezing
$W \leftarrow$ Empty buffer of size 5;
$c \leftarrow 0$ ;                                   // Iteration counter
Freeze $f_\theta$ layers, except for the output layer;
// Phase 2:  Online Learning Iteration
while $is\_connected(\mathcal{C})$ do
    $frame \leftarrow$ ReadFrame($\mathcal{C}$);
    if $frame.id = ID_{target}$ then
        Update sliding window $W$ with $frame.payload$;
        $x \leftarrow$ Normalize($W$);
        if $W$ is full then
            $\hat{y} \leftarrow f_\theta(\text{flatten}(W))$;
            $\mathcal{L} \leftarrow \text{MSE}(\hat{y}, \text{flatten}(W))$;
            // Update only on benign samples
            if $\mathcal{L} < \tau$ then
                $f_\theta$.backpropagation($\mathcal{L}_{batch}$);
                $c \leftarrow c + 1$;
            // Re-evaluate threshold after calibration period
            if $c == N$ then
                $\tau \leftarrow$ RecomputeThreshold($f_\theta, V$);
                $c \leftarrow 0$;
```

The hyperparameters of the stochastic online learning is defined in Table 5.5. The recalibration occurs after 512 updates, where stochastic updates utilizes a batch size of 1.

5.4.1.3 Mini-Batch OL

This workflow is presented in algorithm 2. With a set batch size (*batch_size*), the system accumulates samples that are predicted as benign in a buffer. Once the batch size as been met, the formed mini-batch is used for an instance of backpropagation.

To address the problem of an outdated threshold, recalibration occurs after every mini_batch update. Defined in Table 5.6, the mini_batch hyperparameters include a batch size of 512.

Algorithm 2: Online Learning Stochastic Workflow

```
Input : CAN interface $\mathcal{C}$, Target ID $ID_{target}$, Model $f_\theta$, Initial threshold $\tau$,
        Batch Size $N$, Validation set $V$

tcpPhase 1: Initialization and Layer Freezing $W \leftarrow$ Empty buffer of size 5;
$MB \leftarrow$ Empty buffer of size $N$ ;                          // Mini Batch
Freeze $f_\theta$ layers, except for the output layer;

// Phase 2:  Online Learning Iteration
while $is\_connected(\mathcal{C})$ do
    $frame \leftarrow$ ReadFrame($\mathcal{C}$);
    if $frame.id = ID_{target}$ then
        Update sliding window $W$ with $frame.payload$;
        $x \leftarrow$ Normalize($W$);
        if $W$ is full then
            $\hat{y} \leftarrow f_\theta$(flatten($x$));
            $\mathcal{L} \leftarrow$ MSE($\hat{y}$, flatten($x$));
        if $\mathcal{L} < \tau$ then
            $MB$.append(flatten(x));
        // Perform batch adaptation and recalibration
        if $len(MB) == N$ then
            $\hat{B} \leftarrow f_\theta(MB)$ $\mathcal{L}_{batch} \leftarrow$ MSE($\hat{B}, MB$) ; // Mean loss over batch
            $f_\theta$.backpropagation($\mathcal{L}_{batch}$)
            $\tau \leftarrow$ RecomputeThreshold($f_\theta, V$);
            MB.clear()
```

5.4.1.4 Updates

Updating the last layer of the neural network involves calculating the derivative of the loss with respect to the weights and bias of the last layer.

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w} \tag{5.2}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b} \tag{5.3}$$

As there is no nonlinearity after the last layer, the gradient of the loss is backpropagated directly to the last layer. With the chain rule, this is represented as equation Equation 5.4 and Equation 5.5. For the weight, its gradient is the gradient of the loss multiplied by the input of the layer. As the bias is added to the forward pass equation as a constant, its gradient is just the gradient that was backpropagated.

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \cdot x \tag{5.4}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} \tag{5.5}$$

Once the gradients are calculated, the parameters are updated with equations Equation 5.6 and Equation 5.7, where the gradient is scaled by a learning rate η and subtracted from the corresponding parameter.

$$w := w - \eta \frac{\partial L}{\partial w} \tag{5.6}$$

$$b := b - \eta \frac{\partial L}{\partial b} \tag{5.7}$$

The OL updates utilizes standard gradient descent with a static learning rate and no momentum. In the mini-batch-based gradient descent, the gradient is the mean of all gradients in the mini-batch. In the stochastic-based OL, each gradient descent update uses the gradient update corresponding to the single sample.

5.4.1.5 Hyperparameters

The OL training hyperparameters are different to the initial training hyperparameters. For learning rate, we utilize a learning rate of 0.001. The batch size for mini-batch OL is set to 512. The number of samples per calibration for stochastic OL matches the batch size of the mini-batch OL.

Table 5.5 Stochastic OL hyperparameters.

	Description
Recalibration	After 512 inferred samples
Batch Size	1
Learning Rate	0.001
Optimizer	Gradient Descent

Table 5.6 Mini-batch OL hyperparameters.

	Description
Recalibration	After each batch
Batch Size	512
Learning Rate	0.001
Optimizer	Gradient Descent

5.5 Model Deployment

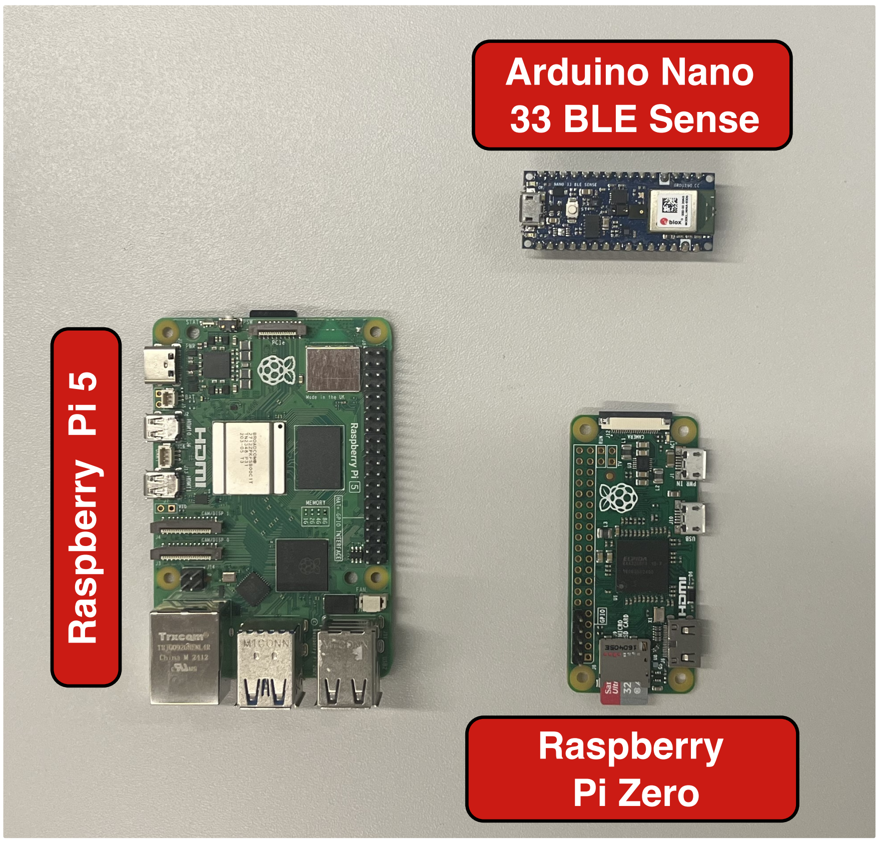


Fig. 5.12 The low-power edge devices used for deployment.

To evaluate computational metrics, we use three heterogeneous devices with three different deployment techniques. The three devices (see Figure 5.12 and Table 5.7) include the Arduino Nano 33, Raspberry Pi Zero, and the Raspberry Pi 5. Each of the three devices represent different levels of resource constraints, demonstrating different deployment opportunities. The architecture of a specific model type is the same regardless of the CAN ID it is trained on. For example, the computation required for the fully-connected speedometer model is the same as the gear shifter speedometer model, as only the values of the parameters change. Therefore, each architecture is only deployed once and its computation metrics then correspond to all models of its architecture. Additionally, the deployment techniques will take into consideration both the fully-connected model's OL computation metrics, which require backpropagation, alongside the inference only metrics of all three model types.

The fully-connected model will be deployed to all three of the devices, while the 1D- and 2D-convolutional models will be deployed to the Arduino platform and the Pi 5 only.

Table 5.7 The hardware specifications of the three deployment devices.

Device	**CPU**	**Clock Speed**	**RAM**
Arduino Nano 33	nRF52840	64MHz	256 KB SRAM, 1MB flash
Raspberry Pi Zero	Single-core ARM1176JZF-S	1GHZ	512MB
Raspberry Pi 5	Quad-core Arm Cortex-A76	2.4GHz	8GB

Arduino Nano 33. Similarly to Im and Lee [41], we utilize TensorFlow Lite Micro (TFLM) to deploy the model onto the Arduino platform. As TFLM deploys a model with static weights, the model's output layer is separated from the full model before it is converted to .tflite. The main block of the model is deployed and utilizes the TFLM interpreter. The extracted final layer has its parameters loaded to the Arduino as a C array header file. Then, a C++-implemented forward pass accesses these parameters.

When the Arduino-based model is to infer a sample, it has two stages. First, the sample is inputted to the TFLM model through the TFLM interpreter. This model ends at the second to last layer of the full model, providing an array of logits of the respective size. This output from the interpreter is then passed to the C++ forward pass. The dot product of the previous output, which can now be considered a new input, and the model weights (plus the bias) is computed. This creates the reconstruction of the input sample. After the MSE reconstruction loss is calculated, a C++ backpropagation step trains only the last layer, updating its imported parameters accordingly. Only stochastic updates are made available for the Arduino deployment.

The Arduino forward pass and backpropagation implementations for the last layer do not leverage parallelization or vectorization. Instead, it uses C++ loops over the flat weight, input and gradient arrays with appropriate indexing to mimic correct matrix sizing.

Raspberry Pi Zero. The model is deployed onto the Pi Zero with the Numpy Python library [45]. The full model's weights are loaded as NumPy arrays and a FP64 model is implemented. The backwards pass of the final layer follows the same and is implemented with FP64 calculations, as is the default. Only stochastic updates are made available for the Pi Zero deployment. With NumPy, vectorized operations to compute many operations to many data elements at once, mitigating the overhead that can be found on explicit Python loops.

Raspberry Pi 5. Similarly to the EV charging infrastructure deployment, the model is deployed to the Pi 5 using the full PyTorch package as the device has sufficient memory. With the PyTorch API, the OL follows the same setup as the local evaluation where all layers, except for the last layer, are frozen. The Pi 5 deployment performs stochastic updates as well as mini-batch updates through PyTorch with the built-in gradient descent and MSE loss functions. No explicit loops are utilized and PyTorch's vectorization and parallelization are leveraged.

5.5.1 Quantization

While the models can already be considered small in size, any improvement in latency can be appreciated in a time-critical use case. In terms of intrusion detection, it provides more time for responding to attacks, which is just as crucial as detection. To evaluate the possible speed up from quantization, post-training quantization (PTQ) is applied to all three of the model architectures. Specifically, we use full 8-bit integer (uint8) quantization to convert both the weights and activations to uint8 representations. To perform this static integer PTQ, we provide a calibration set of samples to allow for the calculation of the activation quantization. The PTQ conversion process is performed with TensorFlow Lite Micro.

In terms of deployment, the quantized versions are imported to the Arduino platform with TFLM. On the device, the input data must be quantized and the output data must be "de-quantized". This ensures the sample input with the model's input as well as the output being processed in its full form. The two equations for this are shown in 5.8 and 5.9 where r and $\hat{r}$ is the real value and the approximated real value, s is the scale, and z is the zero point.

$$q = \text{round}\left(\frac{r}{s}\right) + z \tag{5.8}$$

$$\hat{r} = s \cdot (q - z) \tag{5.9}$$

5.6 Test Rig

While in-lab tests on predefined data can provide an initial look at a system's suitability, the unpredictability of a demo can demonstrate how a system can behave in the real world. For example, performance metrics will struggle to model how a system can behave, or make the vehicle or driver behave, while in use. To further the evaluations and to encourage more real-world ready evaluations, which can take CAN IDS research closer to real-world deployment, we observe the model and its system on a test rig that simulates a vehicle (see Figure 5.13). To accomplish this, we utilize the test-rig to justify the deployed model's suitability, serving as a tool to analyze model behavior. The rig involves a discrete-GPU enabled system and vehicle peripherals, such as a steering wheel and pedals, to support vehicle simulation. This setup supports validation of the proposed system by featuring its own CAN bus, allowing us to approximate the environment found in a real vehicle.

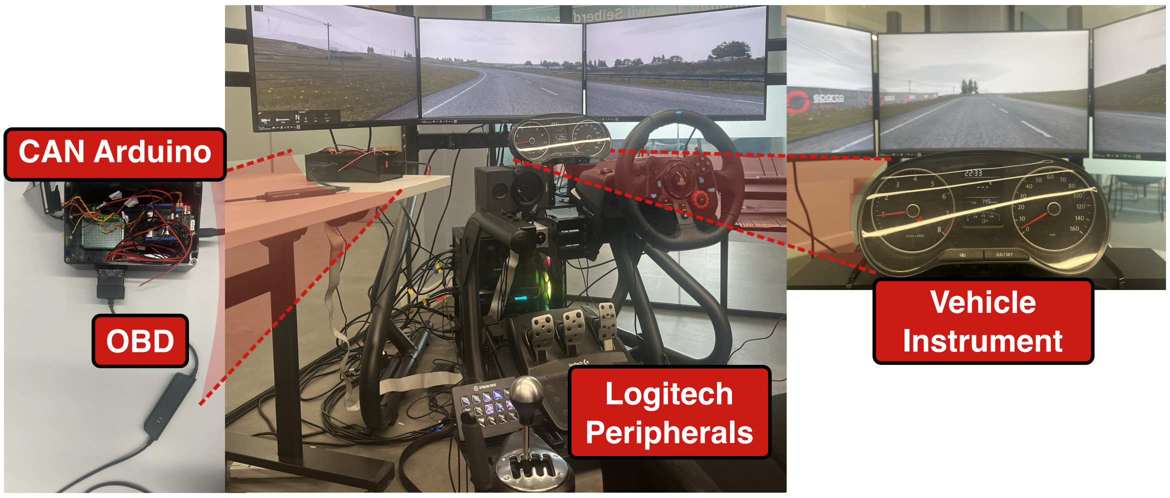


Fig. 5.13 The vehicle test rig controlled by Logitech peripherals.

To detect attacks for this test-rig, new models are trained. Benign data is collected from a typical driving stream, creating our own original dataset. As the CAN setups of the real vehicle in CAN-MIRGU and the test-rig are different, the model must be retrained with the newly collected data. The data is collected from the Assetto Corsa video game. This game was specifically chosen as it provides long stretches of empty roads which enables higher control without having to consider NPC (non-controllable player) vehicles. However, any game in the test-rig would suffice. The dataset is composed from two benign recorded streams. Both streams are recorded in separate driving sessions where a vast array of driving data is aimed to be captured. For example, this includes acceleration and deceleration at various levels of speed, switching gears, and maintaining extremely high and low speeds to capture a large array of payload variations.

To select the CAN ID, the CAN is sniffed to observe the changing CAN payloads and communicated IDs. From this, it was determined that the CAN ID 0x1A0 represented speed information due to its changes that corresponded with the changes in speed. When collecting the dataset, the incoming stream was filtered to only include the payload of messages with the ID 0x1A0.

Creating Post-Collection Attacks. After data collection, attacks are simulated on the CAN data offline. When integrated with a test set, these attacks enable the model to be evaluated. These attacks are created offline by collecting samples of the benign data and introducing attacked messages. For example, a speedometer sample, which contains five frames, is modified to have messages of maximum speedometer at random message indices. As the data has been scaled (between zero and one), the maximum speedometer attack modifies the corresponding byte element to 1.0.

For the chosen CAN ID test rig data, we chose to simulate minimum and maximum speedometer attacks. Given a subset from the test rig test set, two random indexes of the windowed samples are selected. For example, with a window size of five, there are five possible samples to modify. The two selected indexes have their features changed to the maximum and minimum scaled values 1.0 and 0.0 respectively. This modification to the benign test set enables a threshold grid search as data that can be considered to be of the "other" class is now included. In this use-case, the "other" class is everything that has not been trained for, e.g. everything that is not the benign data. However, It must be considered that a larger range of attacks on the speedometer data, and more testing data, would present more accurate model results.

5.6.1 Real-Time CAN Stream Setup.

The test-rig's CAN system contains an on-board diagnostics (OBD) port, allowing communication with the CAN system from an external device. With OBD to USB, we connect the IDS with the local device (RTX3070 system). The CAN python library

[46] facilitates the connection through a Python script. In this Python script, we are able to sniff incoming CAN packets and extract the payload. The payloads are scaled and stacked into a window, producing the windowed payload samples. This window of five samples is inferred. After inference, the last element (the first element that was added of the five) is popped out of the window. This makes room for the next frame, which is then appended. This process repeats continuously as long as the system is active.

The machine learning operations, including the model inference and reconstruction loss calculations, are conducted using GPU-enabled PyTorch on the local RTX3070 system. To compare the usability of the devices on the test rig, we calculate the frequency of messages and the mean interval times between messages of the speedometer CAN ID. Through this, we can measure if the devices can handle the incoming stream of messages for both inference and OL.

5.7 Experiments

To provide clarity in the evaluations, this short section briefly states the experiments conducted in this chapter. The evaluations are presented in section 5.8.

TinyML and Deployment. The converged models are evaluated locally on the attack contaminated test sets corresponding to the specific model's CAN ID. The models are judged with the metrics defined in the background (see 2.5.1).

The fully-connected model architecture is deployed onto all three models (see Table 5.7 and Figure 5.12). The convolutional models are deployed onto two of the devices, the Arduino Nano 33 and the Pi 5. The deployments are judged with the metrics defined in the background (see 2.5.2).

CAN TinyOL. The online learning is evaluated with both workflows, stochastic and mini-batch, on both benign and attack-contaminated streams using an early-stage model. The former recalibrates the threshold after 512 inferred samples while the latter uses a mini batch size of 512. For the evaluation, both the performance metrics and validation loss are utilized to track the performance trends over the course of the online learning (see 2.5.1). At the early-stage model's initial starting point, and at each recalibration step, the weightings of false positive rate and recall and kept equal (0.5 and 0.5) in the threshold search.

Test Rig. The message frequency properties of the test rig's CAN bus are measured and applied to the model deployment computational metrics. Additionally, performance metrics are provided of the models of a given CAN ID in the test rig's CAN bus system. This CAN ID corresponds to the acceleration and deceleration of the vehicle.

5.8 Results Evaluations

This section presents the results and evaluations for the methodologies and experiments found in this chapter.

5.8.1 Model Performances

Model performance evaluations measure the quality of the model architectures on three types of CAN IDs (speedometer, gear shifter, and steering angle). The performance metrics are captured from the models that are trained on the complete initial set of two benign streams (converged models) with no OL. We define these as our "fully-trained" models. Providing an initial starting point, the local device used for initial training features a 3070 laptop GPU. The structure of this subsection presents the performance results of the model architectures individually rather than per-CAN ID. This is followed by a more detailed comparison of the model architectures and their performances.

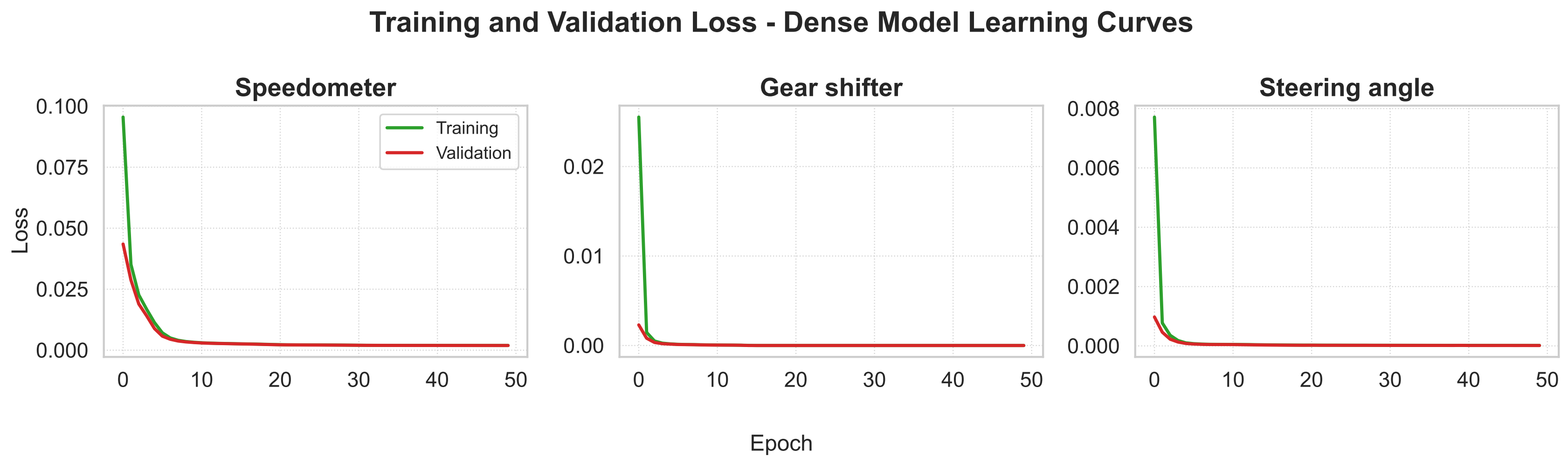


Fig. 5.14 Learning curves of the fully-connected autoencoder.

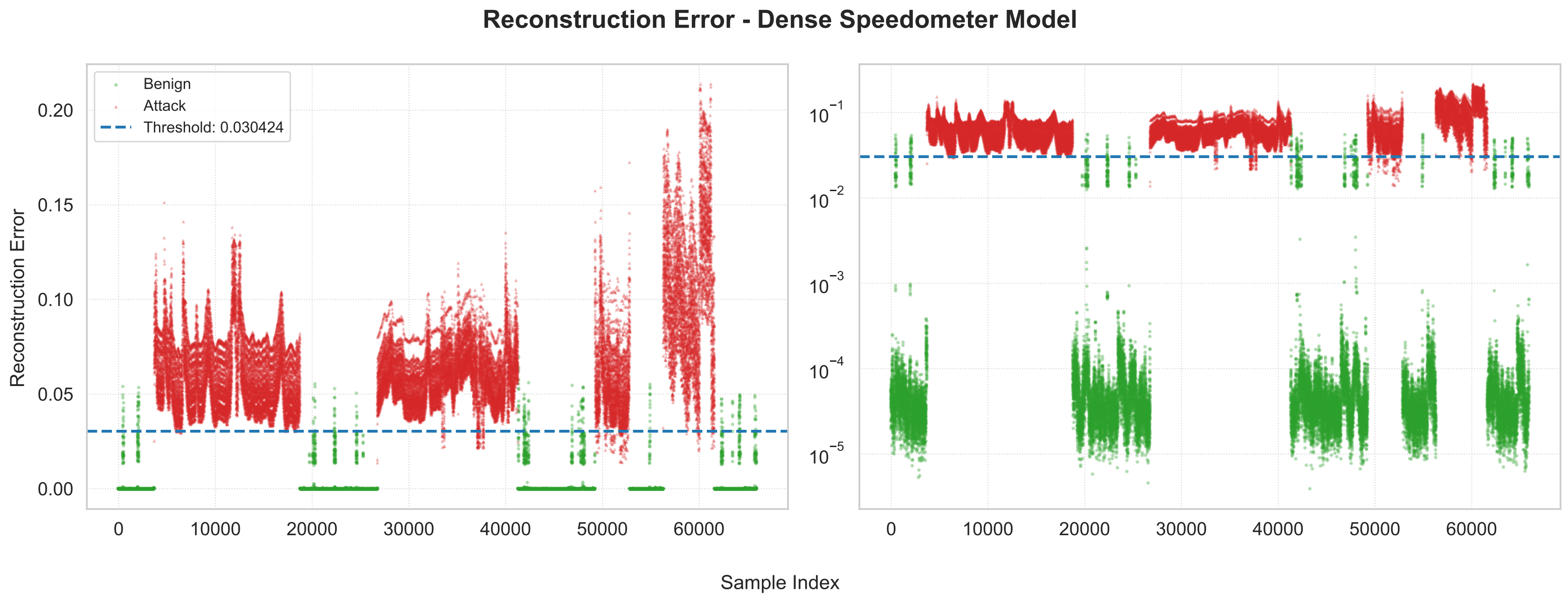


Fig. 5.15 Reconstruction loss of the fully-connected autoencoder's predictions on the speedometer test set.

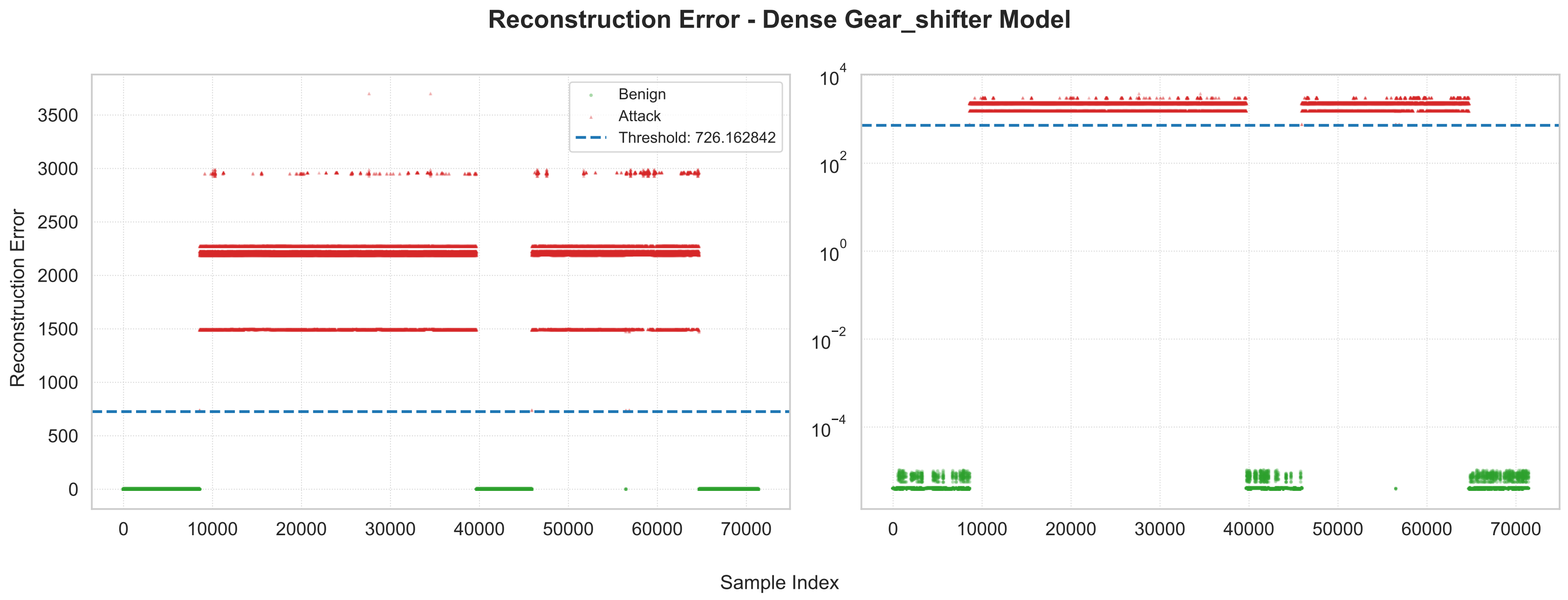


Fig. 5.16 Reconstruction loss of the fully-connected autoencoder's predictions on the speedometer test set.

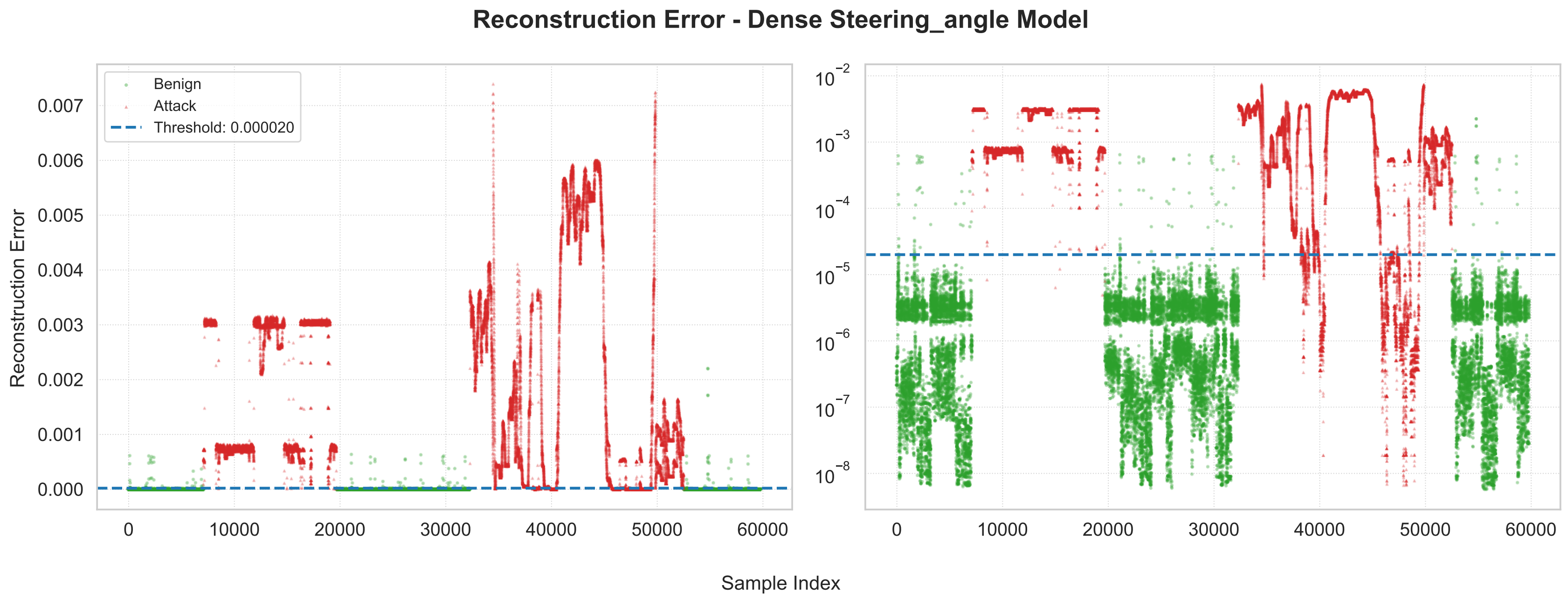


Fig. 5.17 Reconstruction loss of the fully-connected autoencoder's predictions on the steering angle test set.

5.8.1.1 Fully-Connected Autoencoders

Table 5.8 and Figure 5.14 show that the dense architecture is capable of accurately reconstructing the benign windows with consistent performances across all three CAN IDs. The validation loss closely follows the training loss across all epochs as the models learn to reconstruct the overall representation of the benign samples within 5-10 epochs. This is followed by a slow and steady decrease of the validation loss which may reflect continued refinement of the learned representations.

Table 5.8 Converged fully-connected autoencoder model performance metrics.

Model	Accuracy (%)	F1-Score	Precision	Sensitivity (Recall)	Specificity	FPR (%)	ROC AUC
Speedometer	99.195	0.993	0.994	0.992	0.992	0.85	0.998868
Gear Shifter	100.0	1.0	1.0	1.0	1.0	0.0	1.0
Steering Angle	92.81	0.93	0.996	0.872	0.996	0.44	0.960938

Speedometer The fully-trained speedometer CAN ID model are evaluated on both the minimum speedometer and maximum speedometer test streams. The dense model is able to reliably reconstruct benign CAN frame windows despite lacking temporal or spatial feature extraction. It achieves an f1-score of 0.993 on the test sets. This is accompanied by an overall accuracy of 99.195% as well as a precision and recall of 0.994 and 0.992 respectively. 0.85% of benign samples fall above the threshold and as such are false positives. Overall, the ROC AUC score is close to 1.0 at 0.99887, indicating that the model is capable of reliably distinguishing between benign and attack samples.

Gear Shifter The gear shifter dense model achieves an overall accuracy of 100% where all benign samples are correctly classified as seen by the false positive rate of 0%. This is demonstrated again with a precision of 1.0 meaning that all attack predictions are true attack samples. With a recall of 0.99998, a very low amount of attack samples were not picked up by the model and were classed as benign. Overall, this model achieves an f1-score of 1.0. Figure 5.16) shows a large separation between the benign sample's reconstruction losses and the attack sample's reconstruction losses with no overlap in between the two.

Steering Angle Presenting itself as a more difficult task when compared to both the speedometer and gear shifter data, the steering angle fully-connected model achieves an overall accuracy of 92.81%. This is a 6.385 point difference from the speedometer fully-connected model. While the task is more difficult, prioritizing the false positive rate allows it to achieve a lower false positive rate of 0.44% at the cost of less anomalies detected. This is also portrayed in the ROC AUC score of 0.96. Its f1-score is 0.93 produced from a precision and recall of 0.996 and 0.872 respectively

5.8.1.2 1D Convolutional Autoencoders

The 1D convolutional autoencoder is able to learn to reconstruct the benign samples of each CAN ID window creating three models that are reliable at detecting attack samples (see Table 5.9). The learning curves (see Figure 5.18 show that the models are learning how to reconstruct the overall representation of benign samples in each CAN ID within a few epochs as shown by an initially relatively sharp decrease. After this decrease, the reconstruction losses continue decreasing slowly.

Table 5.9 Converged 1D convolutional autoencoder model performance.

Model	Accuracy (%)	F1-Score	Precision	Sensitivity (Recall)	Specificity	FPR (%)	ROC AUC
Speedometer	97.978	0.982	0.99312	0.972	0.991	0.944	0.998947
Gear Shifter	100.0	1.0	1.0	1.0	1.0	0.0	1.0
Steering Angle	94.202	0.945	0.98866	0.905	0.990	1.258	0.967130

Speedometer. The false positive rate for the speedometer CAN ID is slightly higher than that of the fully-connected model at 0.944%. Reliably detecting attack samples as on the given test set, it is able to correctly classify nearly all samples and detects almost all attack samples. It achieves an overall accuracy of 97.978%. However, this figure is the lowest out of all three model architectures on the speedometer dataset. The f1-score is 0.989, produced by the harmonic mean of the 0.993 precision, and 0.972 recall. These scores highlight that of all attack detections, a majority were truly attacks and almost all attacks were detected. Overall, the ROC AUC score of 0.9989 backs up the reliability of the other metrics, as it can accurately distinguish between attack and benign samples.

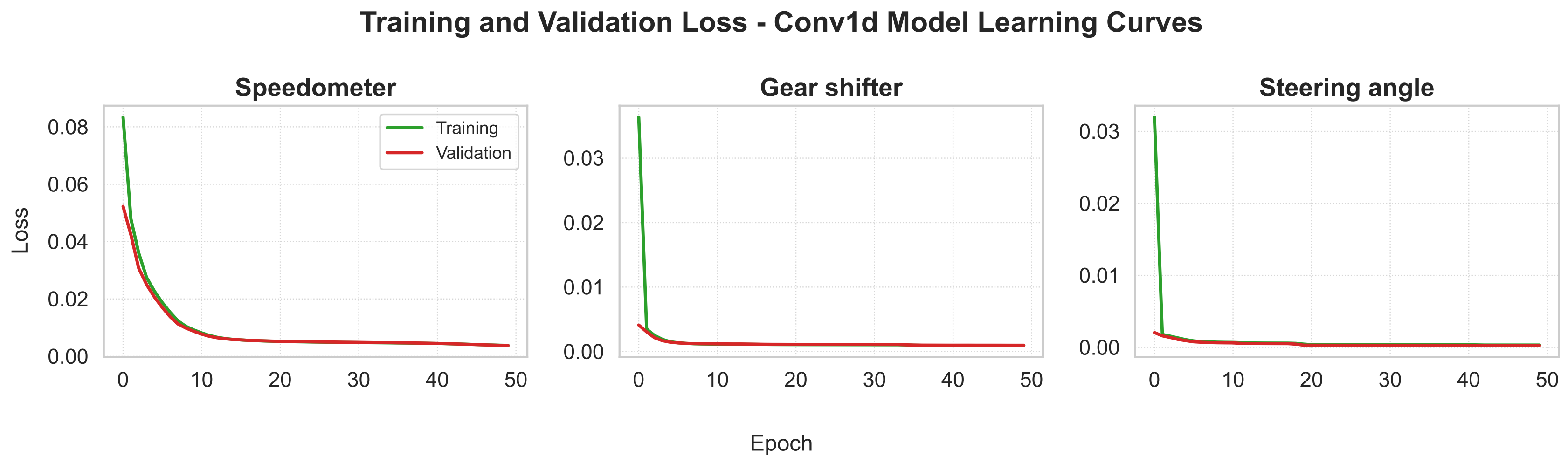


Fig. 5.18 Learning curves of the 1D autoencoder.

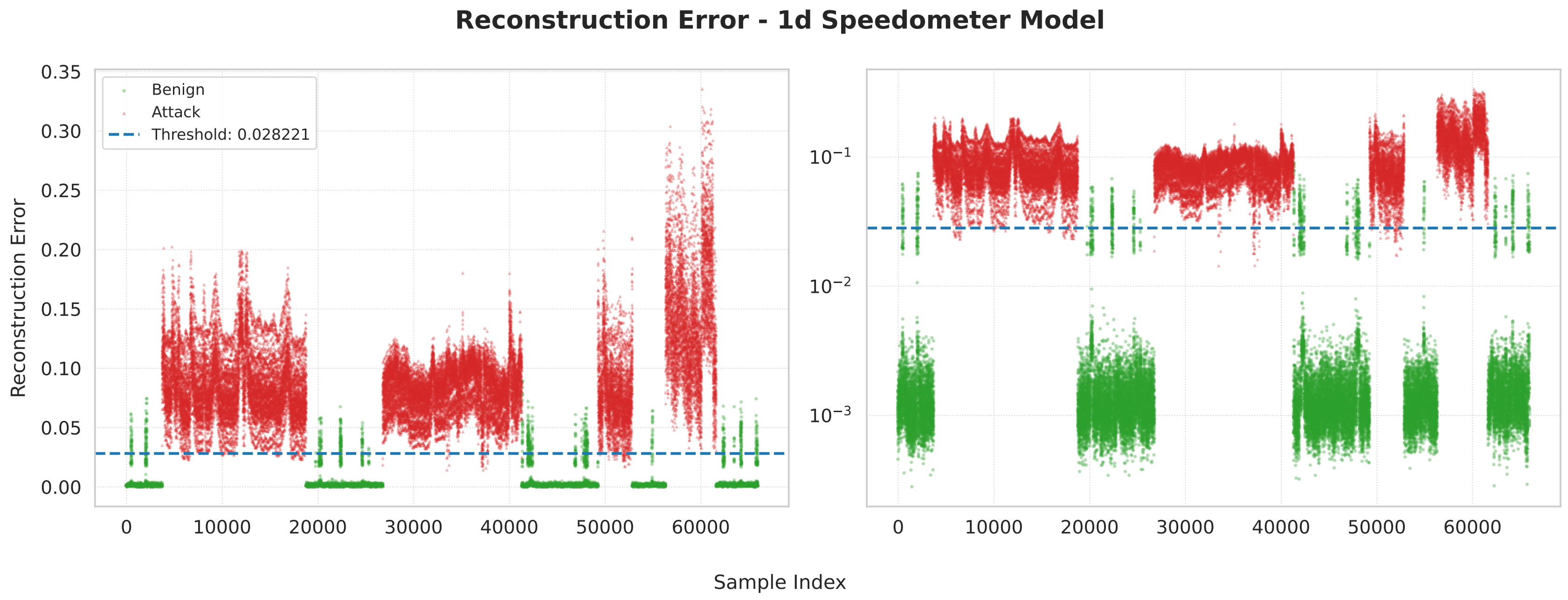


Fig. 5.19 Reconstruction loss of the 1D convolutional autoencoder's predictions on the speedometer test set.

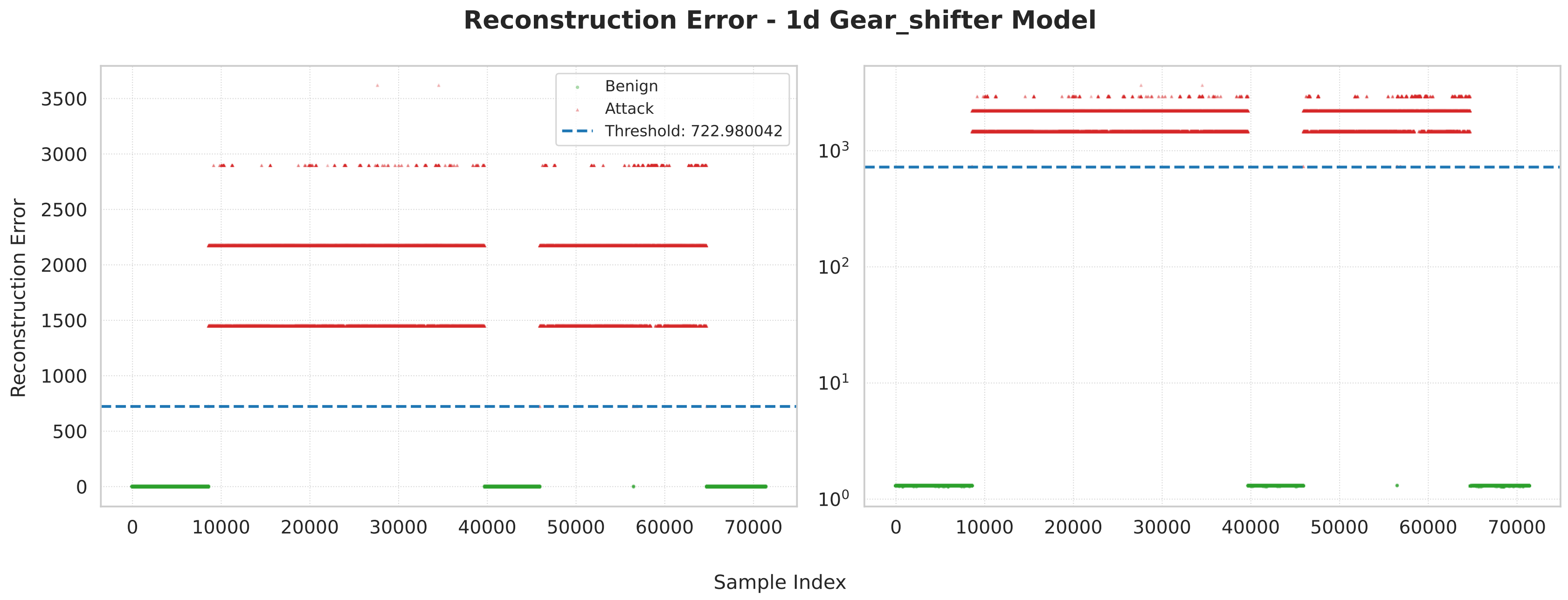


Fig. 5.20 Reconstruction loss of the 1D convolutional autoencoder's predictions on the gear shifter test set.

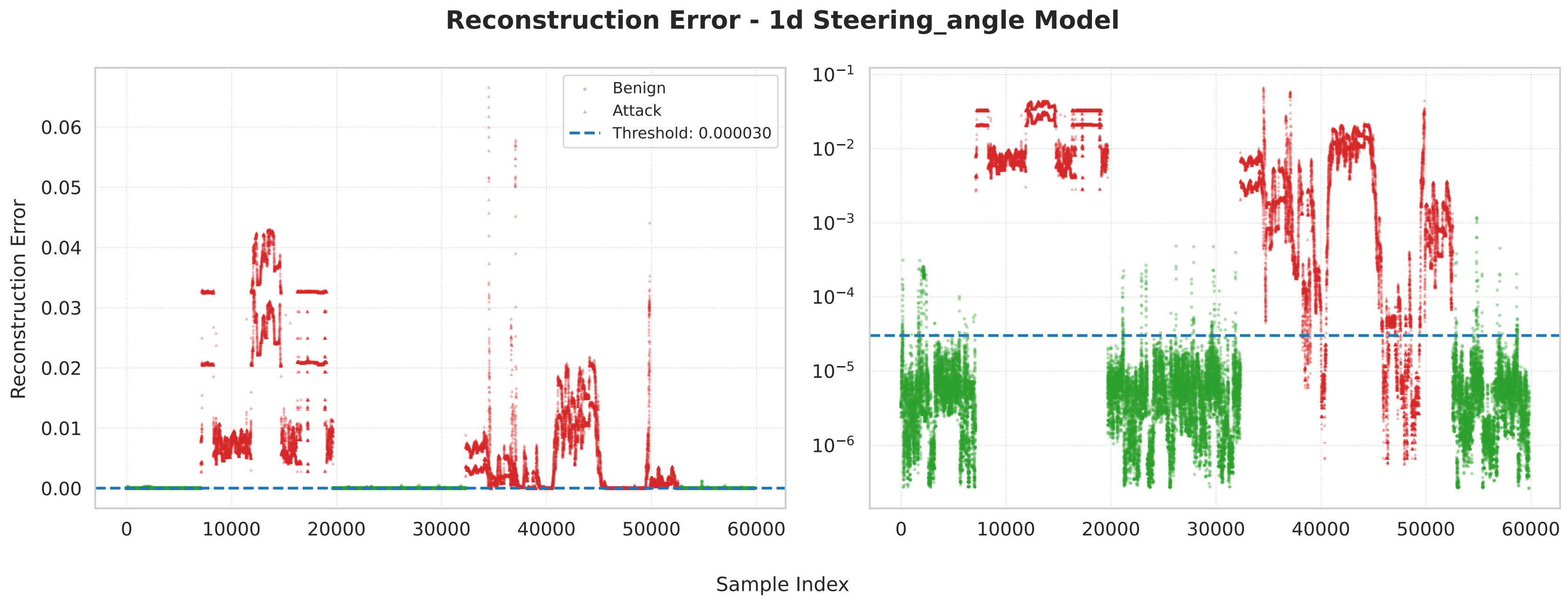


Fig. 5.21 Reconstruction loss of the 1D convolutional autoencoder's predictions on the steering angle test set.

Gear Shifter. Similarly to the fully-connected model, the 1D conv model achieves perfect results on the gear shifter dataset. The plot of reconstruction losses of the test set (see Figure 5.20) shows that the benign samples are reconstructed accurately and as such have a near-zero reconstruction loss. The attack samples are predicted inaccurately with a very high reconstruction loss. The given threshold is set to the lowest attack reconstruction loss due to such a large range of difference between the mean benign reconstruction loss and the lowest attack reconstruction loss.

Steering Angle. With an overall accuracy of 94.202%, the steering angle 1D convolutional model is capable of detecting a majority of attacks in the test set, however, the amount of incorrect predictions further indicate that the task is more difficult. A 1.258% false positive rate may be considered too high, considering the other architectures, such as the fully-connected one, achieve much lower false positive rates. An f1-score of 0.945 is produced from a precision and recall of 0.989 and 0.905, respectively. Ultimately, the performance of this model also follows the same pattern as the fully-connected model, when compared to the model performance on the speedometer data, proving the task to be more complex. Overall, the model can still distinguish between attack and benign samples on a regular basis, but this alone does not make the model suitable for the task given its other faults.

5.8.1.3 2D Convolutional Autoencoders

The performance metrics (see Table 5.10) and the learning curves (see Figure 5.22 of the 2D convolutional autoencoder show that each model is capable of reliably learning how to reconstruct the grid-based representation of the benign CAN windows.

Table 5.10 Converged 2D convolutional autoencoder model performance.

Model	Accuracy (%)	F1-Score	Precision	Sensitivity (Recall)	Specificity	FPR (%)	ROC AUC
Speedometer	98.672	0.989	0.99604	0.981	0.995	0.547	0.999559
Gear Shifter	100.0	1.0	1.0	1.0	1.0	0.0	1.0
Steering Angle	91.540	0.917	0.99016	0.854	0.990	1.029	0.940587

Speedometer The model achieves a false positive rate of 0.547%, which is lower than its 1D and fully-connected counterpart. This is accompanied by an overall accuracy of 98.672%. The precision is similar at 0.996, while the recall is slightly higher at 0.981. Overall, this contributes to a lower f1-score of 0.989, presenting the model to be of a slightly higher quality at the task when compared to the 1D model. The ROC AUC score follows suit at a slightly higher 0.999559.

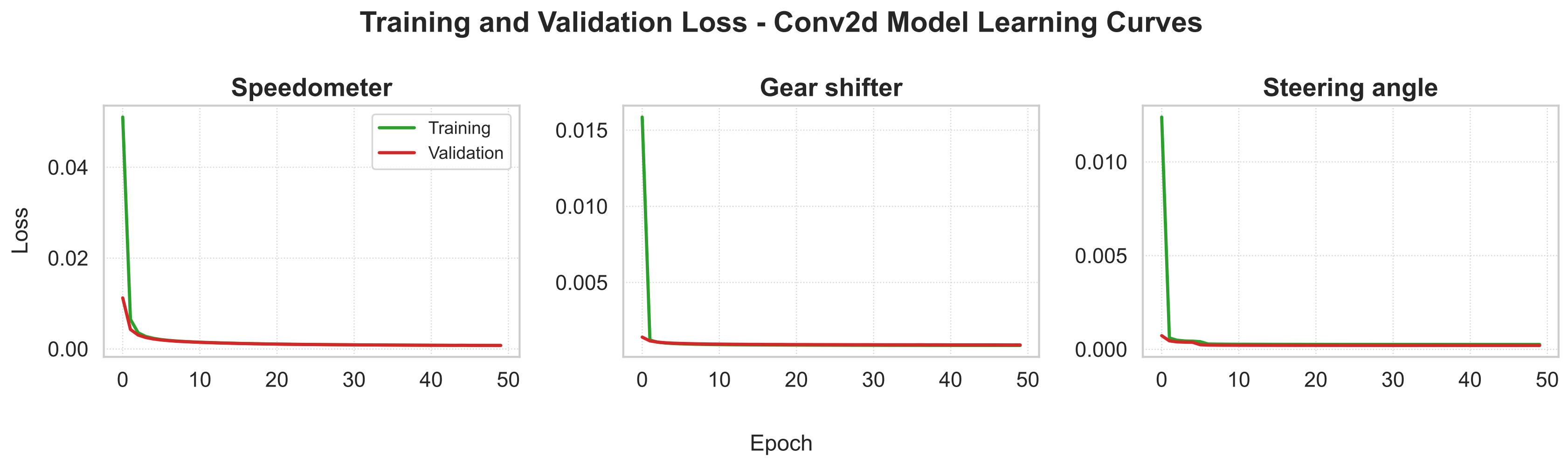


Fig. 5.22 Learning curves of the 2D autoencoder.

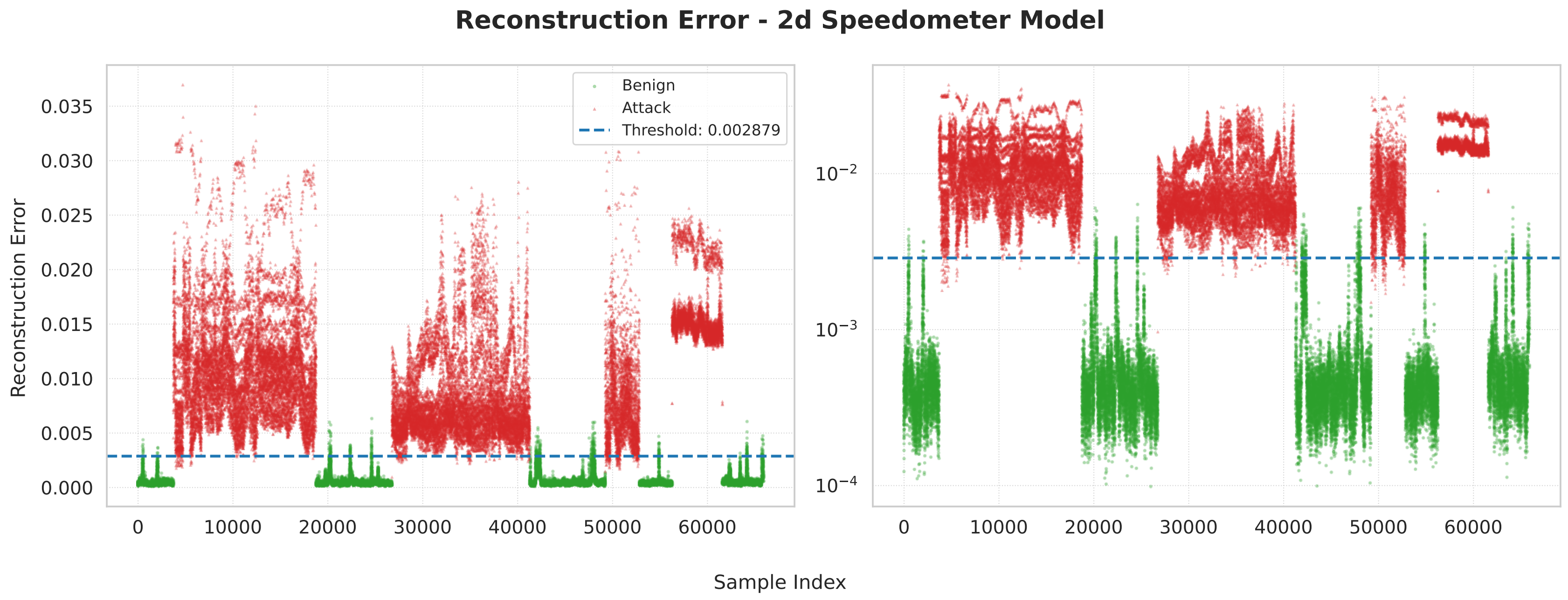


Fig. 5.23 Reconstruction loss of the 2D convolutional autoencoder's predictions on the speedometer test set.

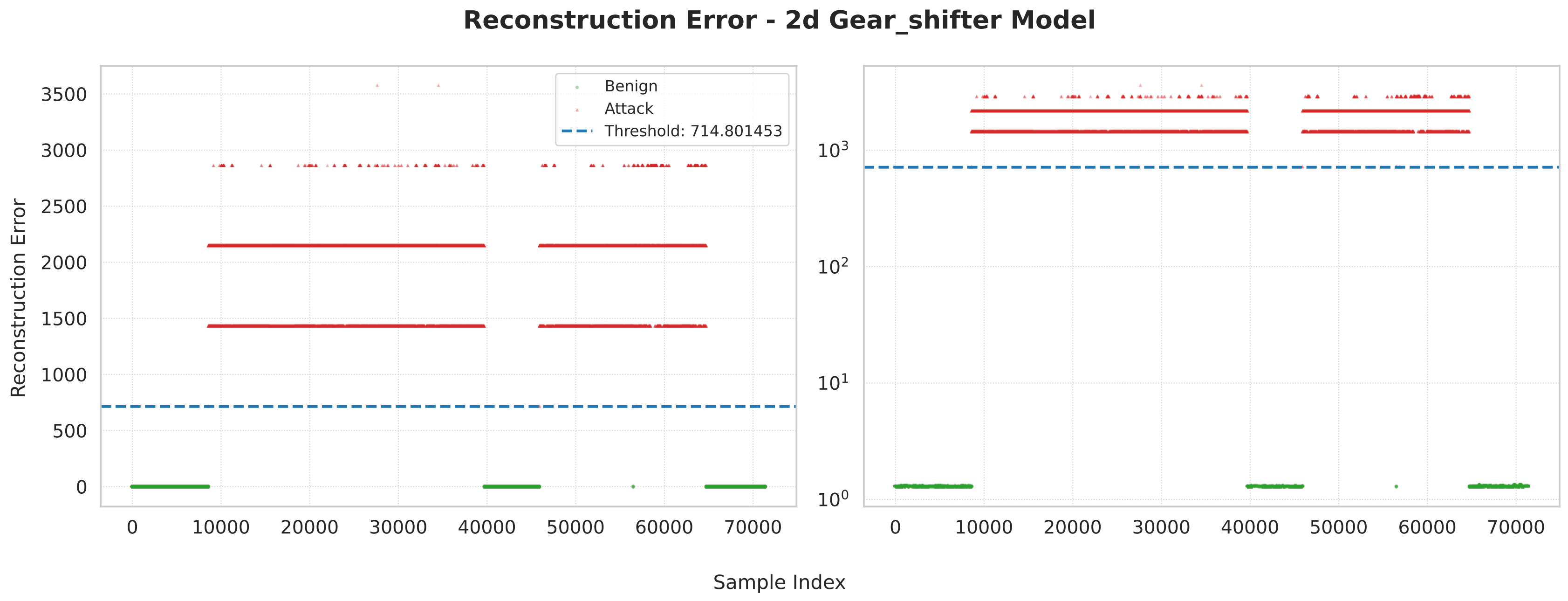


Fig. 5.24 Reconstruction loss of the 2D convolutional autoencoder's predictions on the gear shifter test set.

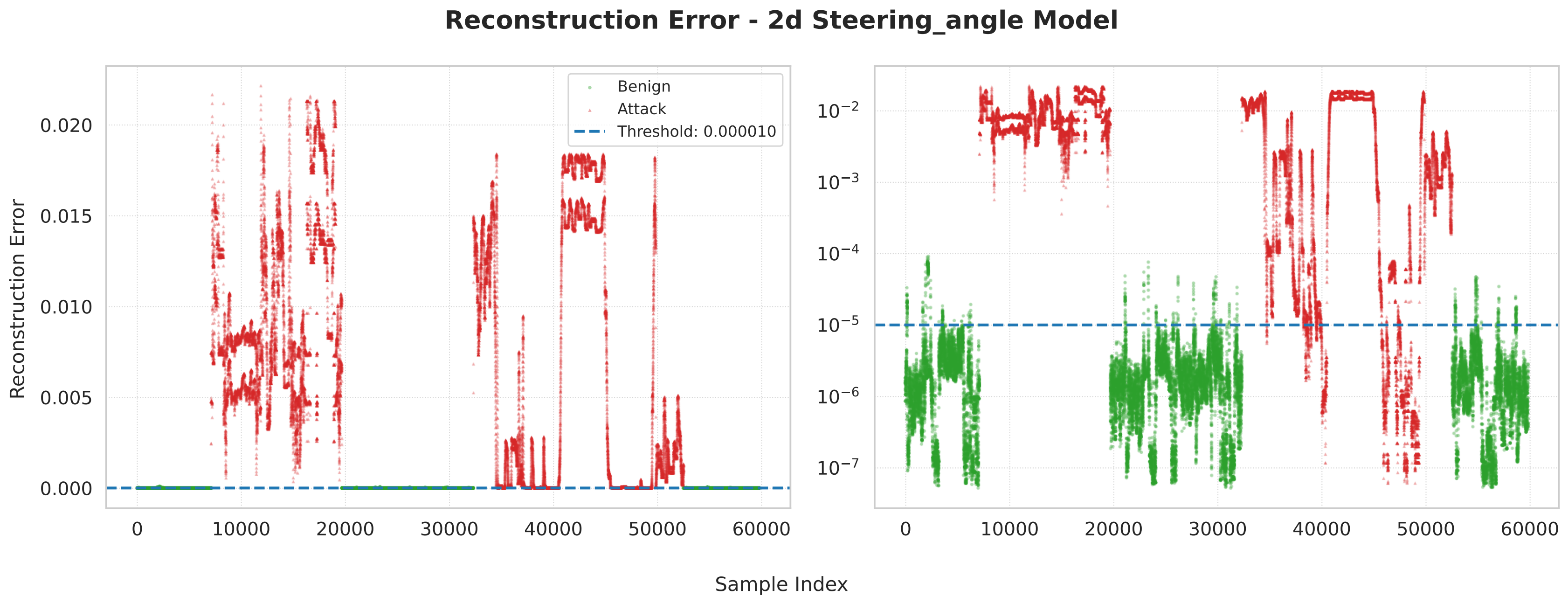


Fig. 5.25 Reconstruction loss of the 2D convolutional autoencoder's predictions on the steering angle test set.

Gear Shifter Matching the other two models, the gear shifter's performance and reconstruction losses on the test set (see Figure 5.24) demonstrates perfect results with an overall accuracy of 100%. As such, the f1-score, precision, and recall are all 1.0 and the false positive rate is at 0%. This is due to the attack representations differing significantly from the benign representations. This creates a large separation in the reconstruction losses as the model cannot accurately reconstruct the attack messages.

Steering Angle The steering angle CAN ID 2D convolutional model has the lowest accuracy of 91.540%. It achieves a similar false positive rate to the 1D convolutional model at 1.029%. A precision of 0.99 means that, when the calculated threshold is applied, almost all attack detections are actually attacks, and there are a very few amount of false detections. With a lower recall score of 0.854, the model misses quite a few detections and predicts them to be benign. The produced F1-score is 0.917. With the lowest score out of all the models, the ROC AUC is 0.94. As stated before, this can still be considered reliable as the model can distinguish between attack and benign samples, however, it may not be for such a time critical and high inference frequency use case.

5.8.1.4 Comparison and Analysis

It can be observed that all three models learn similar representations when provided with windows of length five across the three datasets. This is shown in the similarities in the trend of their reconstruction losses in the reconstruction loss figures.

Speedometer Dataset Models. In the speedometer dataset, the 2D convolutional autoencoder (Figure 5.23) produces reconstruction losses that are lower than its 1D (Figure 5.19) and fully-connected (Figure 5.15) counterparts across both the benign and attack data. Therefore, mean reconstruction losses of the attack samples fall closer to the mean benign reconstruction losses when compared to the other models. As a result, this may cause the slightly lower recall as more attacks fall below the threshold.

In the 1D-convolutional model, there exists better separation between the attack and benign reconstruction losses. However, it can be observed that there are clusters of benign samples which the model struggles to reconstruct, leading to a higher false positive rate, despite the reconstruction losses showing a better model overall than its 2D counterpart.

The dense model has a larger separation between benign and attack reconstruction losses leading to the lowest false positive rate, however, the persistent fpr cluster issue remains. Also the lower false positive rate may be a result of the dense model having a more concentrated set of reconstruction losses for the attack samples. Therefore, the extreme false positive clusters, which also occur in the other models, fall within the range of these more concentrated attack clusters less often. Additionally, the lower false positive rate signifies that the speedometer model has fewer less extreme false positive samples that fall above the threshold.

In Table 5.11, the euclidean distance between the training set and the attack samples of the test set is 0.796. Alongside the combined variance of 0.1004, which does not differ significantly from the individual variances, it can be judged that the samples in both sets are fairly consistent with one another. The attack variance is 0.113 and the benign variance is 0.095.

Gear Shifter Dataset Models. The gear shifter models prove the gear shifter attacks in the dataset to be significantly easy to detect. In all models, the lowest reconstruction losses of the attacks have a large distance to the highest reconstruction losses of the benign samples.

Looking closer at the benign reconstruction losses, the speedometer model is able to reconstruct the benign samples most accurately as it achieves near-zero reconstruction losses. Both convolution models achieve similar minimum and maximum benign sample reconstruction losses with the 1D model producing 1.3019 and 1.4137 while the 2D model produces 1.3003 and 1.4258 respectively. While this can be considered significantly higher than the speedometer model, it shows that the training samples in this dataset have little variation, causing similar reconstructions across an entire stream. This may be due to the nature of a gear shifter, where it only has set modes and therefore few variations. This may also lead to more distinct attacks that are easier to detect, which is observed here.

The statistical measures in Table 5.11 show that attacks can widely differ, given by a variance of 887.353. The euclidean distance of 11.979 between the benign and attack samples and the combined variance of 136.333 suggests that the benign and attack data are widely dispersed from one another.

Steering Angle Dataset Models. The steering angle reconstruction loss figures can be described as the most "chaotic" and harder to detect as a more significant number of attacks are missed and classified as benign across all models when compared to the other datasets. In the figures, the second test set causes the most true negative predictions, while the first test set has a concentrated reconstruction loss cluster which has a larger range between its lowest loss and the benign data's highest loss. The second set of attacks follows similar trends in all three models, with similar samples of the attacks being predicted as benign. This can be observed in the figures, as the same pattern is made, where there is a dip in reconstruction loss.

The 1D-convolutional model finds better separation between the benign and attack samples. While the reconstruction losses of the benign samples are similar between all three models, the attack reconstruction losses are higher in the 1D-convolutional model. In Figure 5.25 and Figure 5.21, there is a relatively larger distance between the losses of the first set of attacks and the benign data. Additionally, unlike the fully-connected model in Figure 5.17, outlier benign samples do not typically achieve reconstruction losses that are consistent with the first attack set.

It can be argued that the fully-connected model's losses are less chaotic and more consistent alongside less false positives overall. However, the dense model's false positives have more extreme reconstruction losses when compared to the other models In this, false positive predictions have a higher spread as well as reaching relatively higher reconstruction losses. Therefore, with improved training methods, improved threshold search, or further online learning, the convolutional models may have a better chance at reducing the false positive rate and possibly improving the detection rate.

This described difficulty of this task is further supported by the statistical measures in Table 5.11, where the Euclidean distance between the set means can be considered marginal at 0.008. This is reflected in the combined variance of 0.03976 which does not significantly differ from the individual variances. The attack variance is 0.03971 while the benign variance is 0.03968.

Less Detections, Lower FPR. Comparing the performance metrics to the reconstruction loss figures show that a model that detects less attacks may be more suitable due to a lower amount of false positives. On the given test set, the clusters of false positives are what the model would detect as false positives in real time. In the provided test set's scenarios where many attacks are injected sequentially, a lower detection performance for a lower false positive rate may be an acceptable trade-off as the model has multiple chances to detect the attack.

Additionally, different model architectures may benefit different types of datasets. For example, many false positives in Figure 5.17 can be considered to be highly overlapping with the attack reconstruction losses. On the other hand, it would be easier to reduce the false positives in Figure 5.25 while maintaining sufficient attack detection as the outliers do not overlap the mean attack reconstruction losses as frequently.

Table 5.11 Statistical measures of all three datasets.

Dataset	Euclidean Distance	Attack Variance	Benign Variance	Combined Variance
Speedometer	0.796	0.113	0.095	0.1004
Gear Shifter	11.979	887.353	0.063	136.333
Steering Angle	0.008	0.03971	0.03968	0.03976

5.8.2 Deployment

The deployment evaluations involve the fully-connected autoencoder's deployment to all three devices and the convolutional model's deployment to the Arduino platform and the Pi 5. These evaluations primarily look at computation metrics to determine the suitability of the model architectures and their deployment methods for the use case of the CAN bus. The computation metrics include mean inference time and mean deployed-OL time for the fully-connected models. Only the former is applicable for the convolutional models.

5.8.2.1 Fully-Connected Autoencoder

Table 5.12 Inference time of the deployed models and training time of the output layer of the deployed dense models.

Device	Inference Time (ms)		TinyOL Time (ms)	
	Mean	SD	Mean	SD
Arduino Stochastic	5.012	0.110	1.932	0.377
Pi 5 Stochastic	0.719	0.331	0.328	0.061
Pi 5 Mini-Batch (512)	1.093	0.739	0.494	0.094
Pi Zero Stochastic	2.754	0.46	1.626	0.306

TFLite. With what can be considered suitable, TensorFlow Lite Micro's setup on the Arduino platform observes the highest average inference time of 5.012ms and average stochastic OL time of 1.932ms. This reflects the Arduino 33 Nano's resource constraints, as it is the "weakest" of the three devices.

Pi Zero. The Pi Zero infers samples in 2.754ms on average with a stochastic OL time of 1.626ms per sample on average. This deployment method with a NumPy implementation may lack dedicated deep learning techniques as NumPy is not a deep learning library. However, it still presents a suitable deployment strategy with appropriate inference and training times.

Pi 5. With 0.719ms in inference time and 0.328ms in training time, the Pi 5 records the fastest stochastic times compared to the other two deployment strategies. The main reason is likely due to the Pi 5 leveraging higher performance components. The inference time is 85.65% faster than the Arduino and 73.89% faster than the Pi Zero. Meanwhile, the training time is 83.02% and 79.83% faster respectively. Alongside this, the Pi 5 setup enables mini-batch-based computing, which can help mitigate any increased latency. For example, it can perform "catch up" can processing multiple samples at once. When utilizing a batch size of 512, the inference time increases by 52.02% to 1.093 and the training time increases by 50.61% to 0.494. These increases are expected, however, they can still be deemed suitable.

5.8.2.2 Convolutional Autoencoders

Table 5.13 Inference time of the deployed models and training time of the output layer of the deployed Conv2d.

Device	Inference Time (ms)	
	Mean	SD
Arduino Stochastic	207.017	0.043
Pi 5 Stochastic	0.522	0.554
Pi 5 Mini-Batch (512)	19.104	5.757

Table 5.14 Inference time of the deployed models and training time of the output layer of the deployed Conv1d model.

Device	Inference Time (ms)	
	Mean	SD
Arduino Stochastic	21.09	0.016
Pi 5 Stochastic	0.503	0.492
Pi 5 Mini-Batch (512)	3.022	0.813

TFLite. In Table 5.13, when compared to other deployment options and the other model architectures, the Arduino platform can be considered not suitable nor optimized for the CAN bus when deploying the full fp32 2D-convolutional model. Its inference time of 207.017ms is significantly higher than all other inference times. When compared to the 1D-convolutional model's time of 21.09, its inference time is 89.81% higher with and increase of 185.93ms. While the 1D-convolutional model's time can still be considered significantly higher than the other models, it may still be considered a suitable solution depending on the frequency of the CAN ID it is tasked at defending.

Pi 5. The Pi 5 presents itself as more suitable deployment options for the convolutional models, obtaining mean inference times of 0.552ms for the 2D model and 0.503 for the 1D model. These are decreases of and 99.75% and 97.61% respectively. When using mini-batches of size 512, the 2D model's inference time increases to 19.104. The 1D model's inference time increases to 3.022. This increase in inference time can still be considered acceptable for the task.

5.8.2.3 Quantized Models

Applying post-training quantization allows for a significant speed-up across all models with minimal performance degradation. As presented in Table 5.15, the 2D-convolutional model's inference time decreases from 207.017ms to 28.494ms, a 86.24% decrease (178.52ms reduction). The 1D-convolutional model follows suit with a decrease from 21.09 to 6.253, which represents a 70.35% decrease (14.84ms reduction). Lastly, the fully-connected model, when completely converted to tflite, achieves a mean inference time of 5.264ms, observes a 98.87% reduction to 0.0594ms (5.20ms reduction). In this fully-connected model, it is considered completely converted as the final layer was not removed prior to conversion.

Table 5.15 Inference times of the deployed fully-connected, Conv1d, and Conv2d models with full-integer post-training quantization applied.

Model	Inference Time (ms)	
	Mean	SD
Fully-Connected	0.0594	0.022
Conv1d	6.253	0.014
Conv2d	28.494	0.037

There is marginal degradation to the performance of these models when compared to their non-quantized counterparts. With little loss of information, quantization can be posed as a viable approach for improving inference time in CAN IDS systems. Additionally, the memory footprint of the models decrease as uint8 values require less storage when compared to higher precision data types, such as the full floating point 32 model. Given the fully-connected model's parameter count of 2,792, a full model of fp32, where each parameter requires 4 bytes (32 bits), would require 11,168 bytes (11.168 KB). When quantized to 8-bit integer, where each parameter requires 1 byte (4 bits), the parameters require 2792 bytes (2.792 KB). This is a 75% decrease in size. The 1D- and 2D-convolutional models, exhibiting the same relative reduction, originally have a parameter memory footprint of 4,457 bytes (17.828KB) and 13,600 (13.6KB) respectively. The former decreases to 4,457 bytes (4.457KB) while the latter decreases to 3,400 bytes (3.400KB).

In the case of the 2D-convolutional model, the significant speed-up in inference time improves the model's suitability for the CAN bus and and its low-latency and high-message frequency environment. For example, the original Arduino inference time may not be compatible with the message frequency of some CAN IDs, forcing the implementation of alternative architectures that could be less suitable. With very similar performances in the evaluated CAN IDs compared to the other two architectures, the quantized fully-connected model presents an optimal solution achieving a highly efficient inference time on a low-power and low-cost microcontroller. Even with the mean inference time reductions from quantization, the convolutional models may benefit from more specialized hardware, such as FPGAs, for improved efficiency through acceleration.

5.8.3 Demonstrating Effectiveness of Output-Only Online Learning

The OL evaluation tracks the performance metrics (see subsection 2.5.1) calculated at each recalibration step (described in Table 5.5 and Table 5.6). The metrics at each data point presents the learning trend over time. In this work, this represents the short-term affect and learning capacity of the technique with the provided data stream (e.g. stochastic). Alongside the standard performance metrics on a test set, the validation loss at each recalibration point is included.

The results of each experiment are presented in Figure 5.26, Figure 5.27, Figure 5.29, and Figure 5.28. In the early-stage model's initial training session, a validation set data point represents a full epoch, which contains multiple mini-batch updates. On the other hand, each data point resembles either one (mini-batch) or a few (stochastic) update iterations in the OL. This results in more minute changes on the validation loss when compared to the initial training, however, each data point cannot be considered equal. Additionally, the trends may have greater fluctuations due to increased sensitivity from more local updates. Overall, during the OL, the validation loss gradually declines, indicating that the model progressively improves at reconstructing benign data based.

The initial early-stage model for the OL begins with an accuracy of 90.02%. This is alongside an F1-score of 0.9175, a precision of 0.89, and a recall of 0.947. The false positive rate (fpr) of 16.64% can be considered extremely significant for this real-time use case with high message frequency. It suggests that during typical operation of a vehicle, the IDS will make regular incorrect alerts.

5.8.3.1 Mini-Batch Online Learning

The mini-batch workflow (see Figure 5.26) presents more stable changes over time due to updates that are less representative of single samples. This can be observed by less relative noise in the performance trends with less deviation around the line of best fit. However, OL does not follow traditional training where a model learns repeatedly off the same samples at each epoch, the incoming samples are unpredictable. Therefore, instability is still a possibility.

In our evaluation, the OL instability causes minute drops in performance, as overall, the mini-batch method does not alter model performance significantly in its configuration. For example, at around recalibration step 225, the recall falls relatively drastically. However, this fall from 0.948 to below 0.945 is a reduction of 0.003. This is a stark contrast when compared to the stochastic flow where moments of instability see reductions of over 0.05 in precision and over 0.02 in recall at given recalculation points. The latter demonstrates the possibility of reductions that can severely degrade model performance.

As stated, the improved stability comes at the cost of the rate of improvement in performance. Less sensitive updates produce more gradual changes when compared to the stochastic OL. For example, the overall accuracy of the model increases from just above 90.0% to 90.03%. The false positive rate sees roughly a 1% decrease, representing an increased separation between reconstruction losses of benign and attack samples as a threshold is able to separate the test set with fewer benign samples falling alongside the attack data above the threshold. The f1-score increases from roughly just below 0.918 to climbing above 0.919. The recall has a slight decrease based on the fitted line,

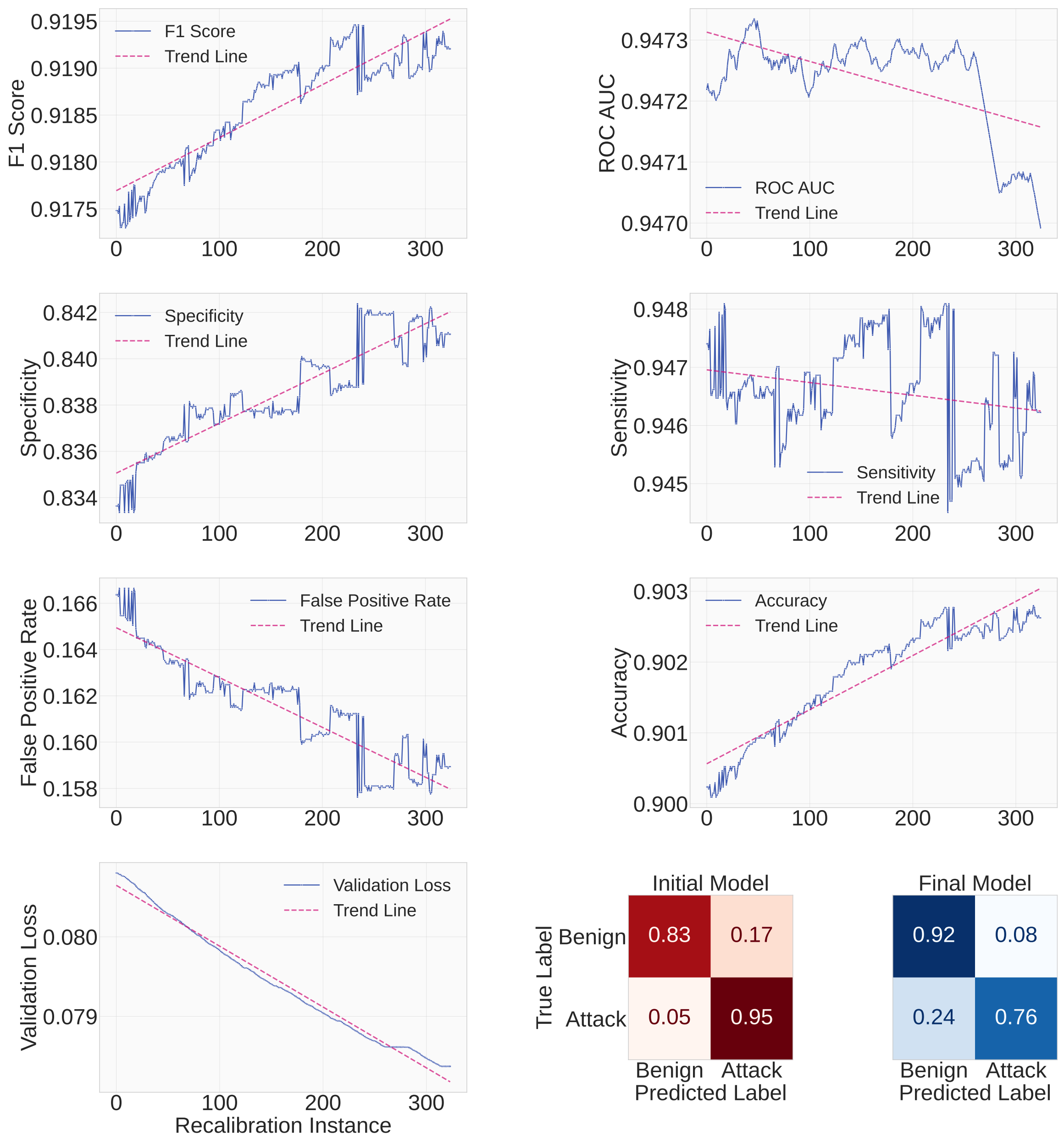


Fig. 5.26 The learning trends of the speedometer OL in the benign mini-batch workflow.

with what can be considered relatively more chaotic fluctuations around this line as its overall trend is not as consistent as the other metrics. However, as stated previously, this instability is minute and not as noticeable when compared to stochastic OL. As indicated by a very slightly decreased final validation loss, the model continues to learn to reconstruct benign representations. The ROC AUC score remains stable, while the trend line points slightly downwards, the changes are, again, very small overall.

The minute relative instability at various update points, that progress into a positive trend, can be considered insignificant as the change in performance would likely not impact the quality and suitability of the model of the use case. However, further analysis would be required on how it improves on a more granular basis, for example, if it specifically improves on incorrect predictions.

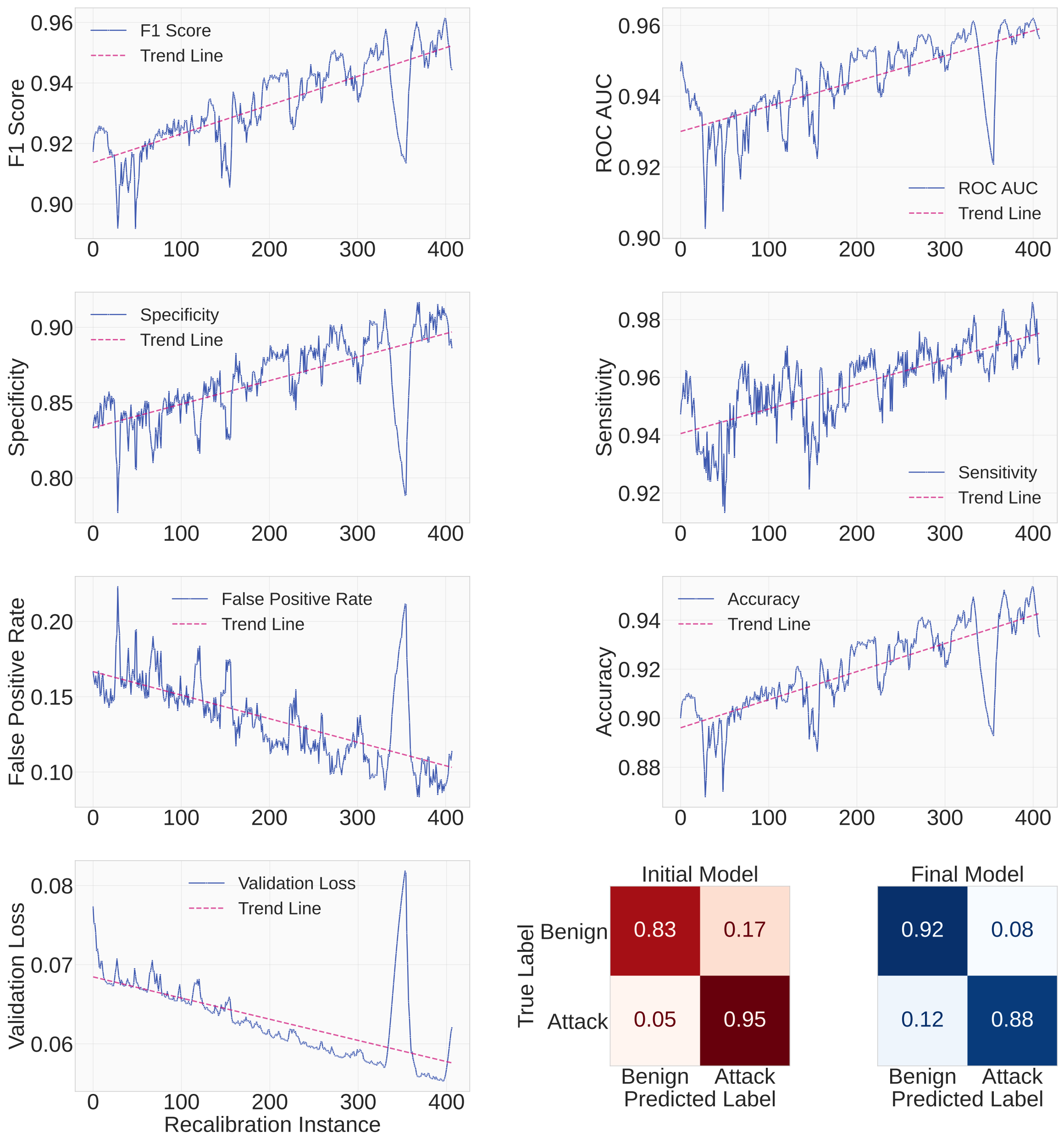


Fig. 5.27 The learning trends of the speedometer OL in the benign stochastic workflow.

5.8.3.2 Stochastic Online Learning

While the traditional strengths and weaknesses of stochastic-based updates are shown, the model sees considerable improvements in the stochastic OL (see Figure 5.27) . This workflow exhibits what can be considered relatively higher variability in its trends as there is a higher sensitivity to individual samples. However, the improvements in the performance metrics are larger, when compared to the mini-batch updates. For example, by the end of the OL, the accuracy surpasses 95%, roughly a 5 percentage point increase from the starting point and higher than when using mini-batches.

Following the trend of the accuracy, the F1-score climbs towards 0.96, roughly 0.04 score points above the workflow 1 F1-score peak. The precision and recall support this with positive trends, as the former climbs towards 0.98, and the latter climbs above 0.925. This again demonstrates significant improvements over a shorter time span when compared using mini-batches. The false positive rate falls by over 5 percentage points to below 10%, reflecting that the model likely captures outlier behavior in benign data better through more sensitive updates. The validation loss over time confirms the model's growing ability to reconstruct benign samples, with an even greater decrease, where it falls below 0.06, when compared to the mini-batch's sub-0.079.

However, between 300 and 400 recalibration steps, the performance metrics observe a large negative spike, which is reflected in the validation loss. For example, the false positive rate, after a steady decrease towards 10%, spikes up to above 20%. The f1-score, after a steady increase towards 0.96, falls below 0.92. The validation loss sees a spike that rises further than 0.08, which is above its original starting point.

Another example of more minute instability can be observed in other parts of the OL. Between 0 and 100 recalibration steps, the trend is positive on all metrics, however, a small increase in validation loss is seen around and before the 50 recalibration step mark. Here, the accuracy falls below 87.5%, which can be considered a significant drop. The f1-score, recall, precision, and false positive rate experience the same negative point in performance. The false positive rate increases, from roughly 15%, to above 0.20, an all time high fpr in the OL session. While this model may already exhibit many false flags in real-time, such an increase at any point (e.g. if the model had a much lower fpr) may render the model unusable. However, the increase in validation loss can be considered small, when compared to previously mentioned instability, demonstrating the limitation of utilizing validation loss for model quality in this use case. The performance metrics see what can be considered significant instability, while the validation loss, which judges the model's ability on only benign samples, does not reflect similar instability.

The possible instability, as demonstrated, may make the model unsuitable for real-time use at some points of OL. Additionally, any significant increase or minute change in metrics, such as recall or false positive rate, may be detrimental depending on which samples are no longer correctly predicted. In real-world use, future instability is not predictable, as we cannot know the next frames in a real-time stream. However, with the appropriate management of said instability, the capacity of adaptation and improved performance will derive a more secure system at a faster rate with smaller batch sizes.

Backing up the other metrics, the ROC AUC finds that the model is slightly better at distinguishing between benign and attack samples over time. This improvement is not as significant as the other metrics, proving that further refinement is needed over a longer time span in this early-stage model scenario. However, it also demonstrates that despite a large degradation of ROC AUC at the beginning, the model is still capable of large improvements. Again, this demonstrates the "unknown" factor, where the quality of the OL can be extremely dependent on unknown future frames and may require additional methods to minimize any risk. Lastly, the confusion matrix presents the improvement at a more granular level. While the amount of attacks detected reduce, the improvement follows what can be considered a more ideal trade-off where a threshold search (with the same weightings as the initial model) produces a model better at detecting benign samples. This is paired with less benign samples predicted as attack. With the windowed samples, this trade-off can be considered more ideal as a model has multiple chances to detect an attack frame. At the same time, attacks are rare when considering the amount of CAN frames that are transmitted, therefore, a lower false positive can be considered key in such a trade-off.

5.8.3.3 Attack-Contaminated Online Learning

The attack variations of the workflows (see Figure 5.29 for mini-batch and Figure 5.28 for stochastic) highlights the limitations of OL under adversarial conditions.

While the attack-contaminated stream length is a lot shorter than the benign stream length, we can still observe that there is no overall improvement in performance on the test set both stochastic and mini-batch by the end of the online learning. This may be due to the shorter time span, however, it can be judged that the contamination has halted or limited the model's learning capacity for the specific task.

The Stochastic OL workflow (Figure 5.28) shows performance that does not match the validation loss. The validation loss increases, showing the model is worse at reconstructing benign samples. However, the other metrics demonstrate improved performance at the same time. This can be due to the validation loss only representing the model's performance at reconstructing benign samples, and it performs worse on this metric as it is beginning to be optimized for attack samples. This should mean that the other metrics also worsen at the increase of validation loss, however, this is not consistently the case as the performance can be considered stagnated instead. Additionally, some attack samples may have similar characteristics to the benign data, as the attack samples may have some benign frames within them. Therefore, it can be argued that these attack samples may not completely degrade performance. Ultimately, the performance worsens as the validation loss begins to decrease again, which may demonstrate the effects of the poisoned model. This decrease in performance may not be reflected in the validation loss because the validation set only contains benign samples, therefore, as the attacks stop, the model is optimized for only benign samples again. The model's performance on attack samples is unknown in the validation set. Based on the trade-off conditions previously mentioned, the confusion matrix suggests that the model has progressed into an improved direction with a higher amount of benign samples predicted correctly.

As expected, mini-batch OL workflow (Figure 5.29) is more stable, as the performance sees some improvement by the end of the OL. However, the overall trends as described for the stochastic-contaminated workflow is roughly the same. As what may be due to the less local updates, while there are some moments of instability, the model manages to maintain performance more consistently, demonstrating greater resilience from averaged updates. While the stochastic improvements are greater and quicker, the model ends up in a worser state by the end. Just like in the benign only streams, the mini-batch-attack workflow does not reach the same level of improvement during the stream, but its robustness is demonstrated.

Lastly, the model can now be considered poisoned, and even with similar performance trends when compared to its starting point. As a result, it may be a worse model overall. The model may still retain residual poisoning effects and may require human or more controlled intervention. Due to the frequency of attacks this may not be a problem, however, a deliberate poisoning attack by an attacker may attempt to reduce the reconstruction loss of some attack samples by introducing outlier benign samples to the training set. While this may also be considered rare and an edge case, it may require further research into mitigation techniques for the IDS as it introduces a possible vulnerability. Again, the confusion matrix finds that the model progressed to the more acceptable trade-off despite smaller improvements in the other metrics. However, the amount of correctly predicted attack samples can be considered significantly less.

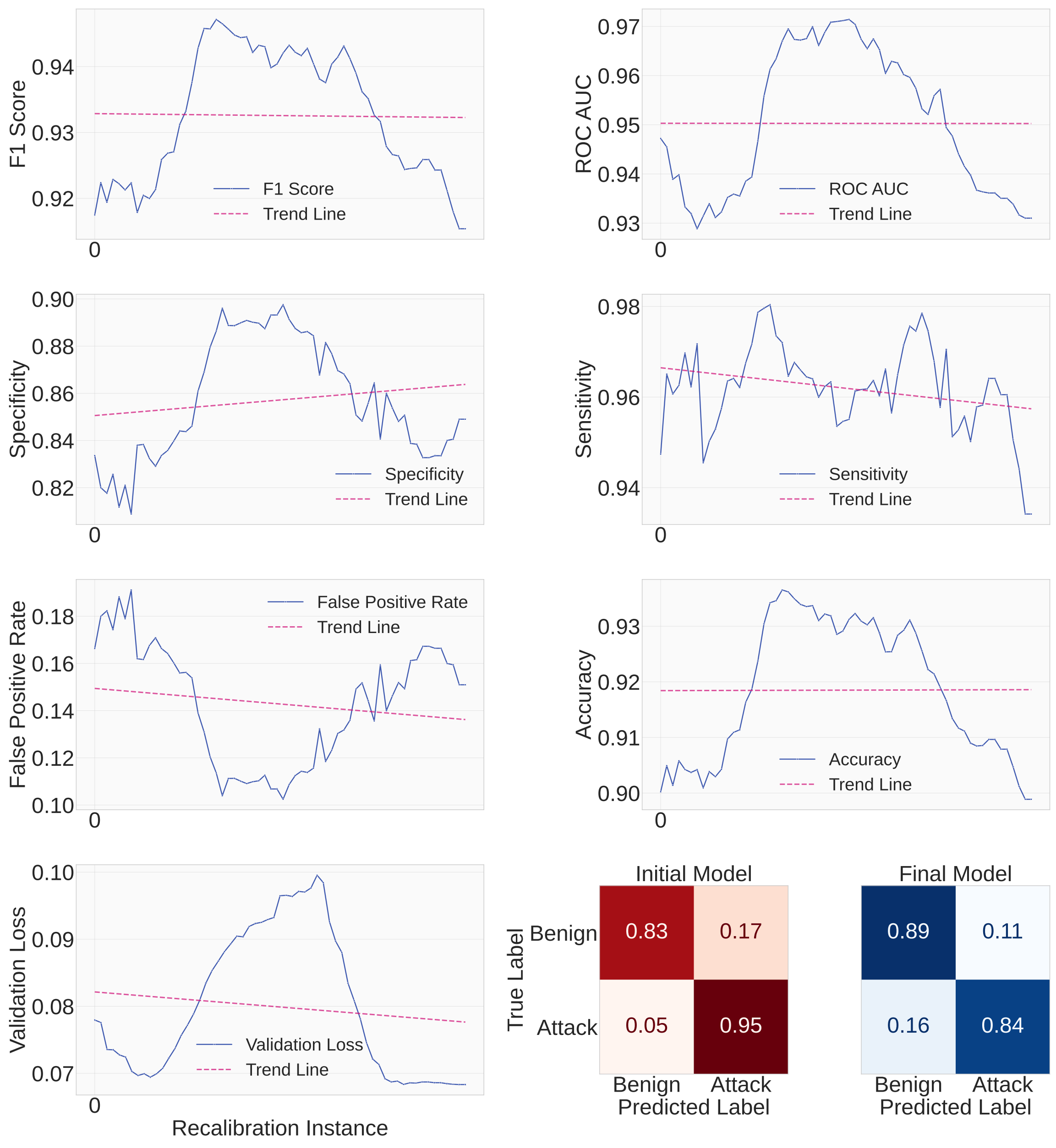


Fig. 5.28 The learning trends of the speedometer OL in the attack stochastic workflow.

5.8.4 Test Rig

Message Statistics. Out of a total of 24639 captured messages of the CAN ID 0x1A0, we observe a mean interval time of 14.221ms. The recording session had a duration of 350.393 seconds (roughly just under 6 minutes), where the 24639 total speedometer messages result in a frequency of 70.318 messages per second. If we compare the mean interval time to the inference and OL times of the fully-connected

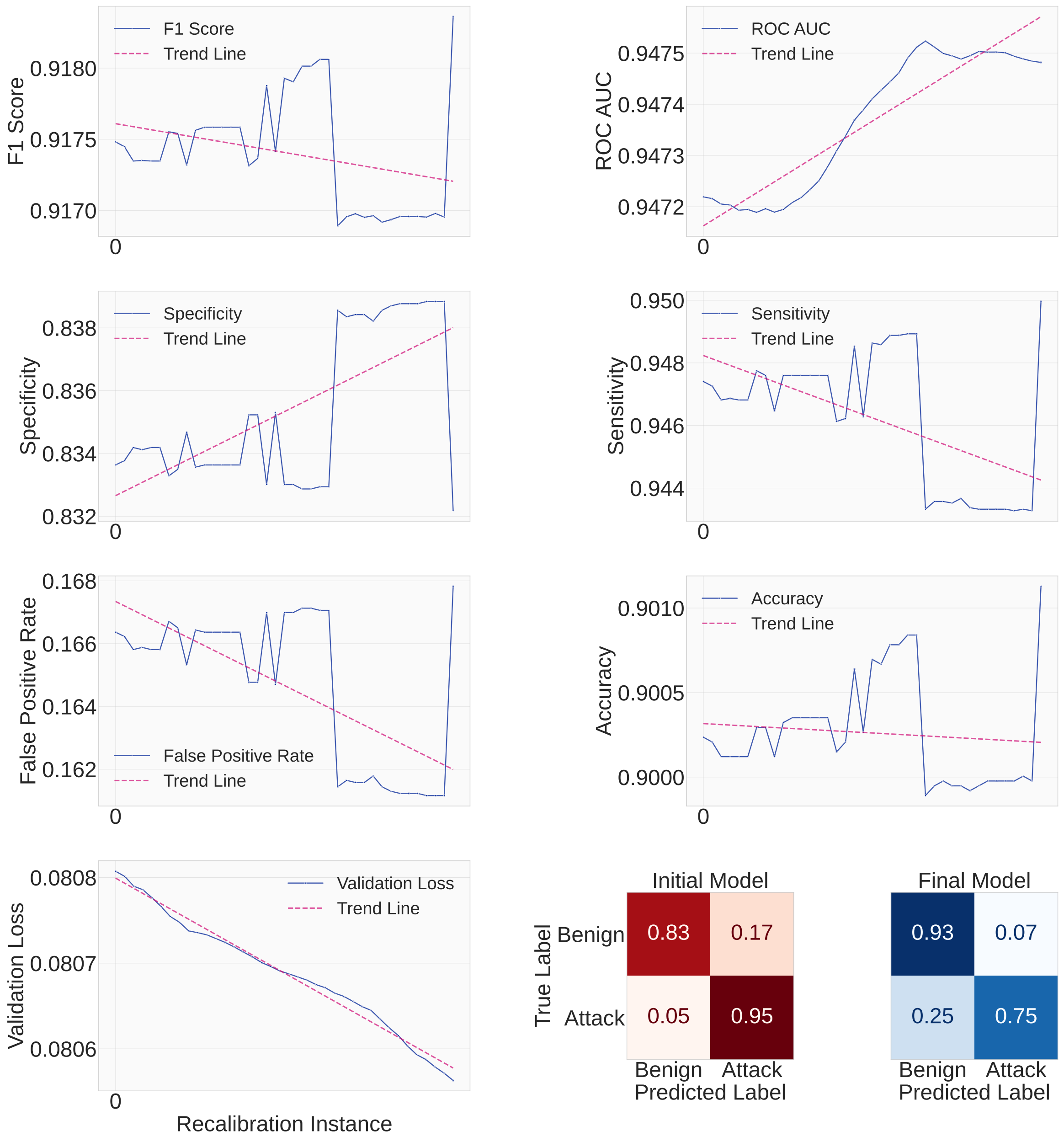


Fig. 5.29 The learning trends of the speedometer OL in the attack mini-batch workflow.

speedometer model, each device is capable of sufficiently meet the requirements of the test rig's can message. For example, with a summed mean inference and stochastic OL time of 1.047ms, the Raspberry Pi 5 can conduct both operations before the next frame requires processing. This is also the case for Arduino deployment of this architecture, where the sum of inference times is 6.944ms, which while longer is still sufficient.

When considering only inference times in the 1D- and 2D-convolutional models, the Arduino deployment may not be appropriate as the mean inference times of 21.09ms and 207.017ms may not keep up with the frequency of incoming messages. The quicker 1D-convolutional model may be squeezed into some CAN IDs, however, it could cause issues with latency. When quantized, the 1D-convolutional model could sufficiently fit in the time constraints, while even with drastic improvements, the 2D-convolutional model may introduce latency or not sufficiently keep up with the message stream. The Pi 5 and Pi Zero deployment variants of the 1D- and 2D-convolutional models in stochastic flows would typically be capable of inferring samples at a sufficient frequency.

These comparisons demonstrate guarantees that the model architectures operate sufficiently, when deployed appropriately to the correct hardware. For example, the convolutional models may require more specialized deployment options such as with edge devices that accelerate their operations. Sufficient operation requires the model to be likely to process the next window sample without delay once the system has completed operations on the previous sample.

Model Performances. The test rig models (Table 5.16), trained with data of a different CAN system, provide additional demonstration of performance capability on CAN data with temporal patterns. The dense model achieves an F1-score of 0.999 with an accuracy of 99.93% and a false positive rate of 0.079%.

Table 5.16 Test-rig data performances.

	Accuracy	F1-Score	Precision	Recall	FPR
Dense	99.93%	0.999	0.998	0.9995	0.079%
Conv1d	100%	1.0	1.0	1.0	0%
Conv2d	100%	1.0	1.0	1.0	0%

While the test set involves a set of simple CAN attacks, the inclusion of additional attack types alongside a larger test set with more attack data would provide a more comprehensive and accurate evaluation. Additionally, it demonstrates how attacks can vary in difficulty, as the created attacks here can be considered too easy and less stealthy as their deviate a lot more from the typicall representation of benign samples.

5.8.5 Discussion and Observations

This section provides a discussion and presents observations based on the methodologies and the evaluations.

5.8.5.1 Data and Model Performances

As observed by the model's performances and reconstruction loss trends, different CAN IDs may benefit from different types of models. For example, the gear shifter model may benefit from a simpler model with less parameters, while a more complex CAN ID, such as steering angle, may benefit from a more complex or specialized model. A system of specialized models per use case may also enable an improved system as more time-critical CAN IDs may require models with less parameters for lower latency, however, this requires further effort on the trade-off of efficiency and detection performance.

Additionally, the differences in test sets demonstrates how the type of attack has unpredictable results and there may be attacks that make it through, even if the model achieves extremely reliable performance results. For example, the gear shifter achieves perfect results on the test set, however, a calculated and planned replay attack may not be detected as the model if it happens to create a window that appears as benign. The same can be true for the speedometer and steering angle models.

The hyperparameters in the linear threshold search can also heavily impact the quality of the model. With smaller steps in the search, a threshold closer to the most optimal threshold is less likely to be found. For example, when the search is conducted in steps of 0.00001, the 2D-convolutional autoencoder model achieves a false positive rate of 0.026% and an f1-score of 0.879. Alternatively, with more precise steps of 0.0000001, the model achieves a false positive rate of 1.306% and an f1-score of 0.907. The former model may be considered safer as it will raise significantly less false flags, especially considering the rarity of attacks. Additionally, prioritizing specific metrics may derive a model that is capable at detecting all attacks, however, may also flag too many benign samples. On the other hand, setting higher weighting for the false positive rate may result in a model that misses out on stealthy attacks with representations that are more consistent with benign samples. In this work, the importance of the false positive rate was highlighted. Therefore, when evaluating the converged full models, a higher weighting was applied to the false positive rate in the threshold search. When using equal weightings, the increase in false positive rate can be considered unsuitable for the task. Overall, it can be observed that the performance of the model can vary depending on the linear threshold search, and as such, it should be carefully adjusted with consideration of the trade-offs.

5.8.5.2 Online Learning Strategy

With the TinyOL method of training only the last layer to enable training on low-resource devices, we find that the model is capable of adapting and learning benign representations from an unseen benign stream as the model is seen improving in the short-term. Alongside this, given message frequency statistics and deployed-model computation metrics, we can determine that the system can conduct both inference and OL while causing little to no increased latency.

To improve the short-term and long-term performance of the OL, we discuss how the system can be modified.

Dynamic Hyperparameters. While in the evaluations, OL with static conditions in hyperparameters is applied, a more dynamic system may provide a best-of-both-worlds solution. The mini-batch approach provides more stable updates where the model is more likely to remain suitable for the task during its OL lifetime. On the other hand, the stochastic approach provides quicker adaptation and improvements given its more sensitive nature, however, the model is more likely to see stark degradation in model quality. Adjusting the mini-batch size dynamically would allow for moments of quicker improvement alongside moments of increased stability. This mini-batch size may depend on the type of benign samples within the batch. For example, more sensitive updates on a type of benign sample may lead to overfitting or other performance issues at a model's given state. If the learning rate is too high, this can also be the case. A low learning rate may not allow for meaningful updates. Therefore, learning rate can also be adjusted dynamically.

Sample Selection Strategy. Alternative sample selection strategies to better select benign samples with appropriate techniques, like dynamic hyperparameters, to avoid overfitting or other issues, may result in a more stable and suitable model in the long-term. For example, to improve the model's reconstruction of outlier benign samples, such as those that are reconstructed with the same loss of attacks, it may be beneficial to specifically focus on these samples in OL. However, as the statistical measure of these samples when compared to the mean samples, such as variance, may differ more significantly, it can lead to a level of catastrophic forgetting. Therefore, the model may also benefit from the replay technique often found in methods that are susceptible to catastrophic forgetting, such as continual learning.

However, bias in the training samples may become an issue when specifically selecting training samples.

Adaptation After Convergence. The evaluation scenario used in the OL contains a model that still has room to grow. This is presented by an early-stage model. In real world use, it is possible the deployed model may already have converged on the initial training set. Therefore, its adaptation in the OL would be focused on refinement and handling any drift in the data. This may not be reflected in a set test set, but may rather require a "per-sample" basis of model evaluation. For example, model performance on a test set may remain consistent, however, it may have higher confidence in reconstructing some benign samples. This may not be reflected in performance metrics due to the frequency of such samples in the test set. For example, if it is an unseen sample, in both classification due to an unseen attack or autoencoder-based anomaly detection due to data drift, the sample may not be considered appropriately, if at all, in a given test set. Therefore, real-time monitoring and evaluations may derive more insightful and beneficial judgments on model quality over time. Additionally, occurrences of OL may be selected and not continuous, as a converged model may not significantly benefit from further training until any performance beigns to degrade. Alternatively, as previously described, a converged model may still benefit from instances of OL with selective samples in order to better reconstruct outliers.

Use Cases. While the suitability and implementation must be evaluated in further use cases, the system provides the groundwork for a locally adaptive IDS to allow for personalization and combat drift in data. Driver identification often does not use the raw untranslated CAN frames as seen in this study. Once translated, CAN frames may provide key information such as x, y and z. These features can be more precise, specific, and informative on a vehicles state at given moments. A drivers data is typically not known beforehand and must be adapted to online. Additionally, road types and other factors, such as time of day, can affect driver behavior, impacting their data. Therefore, one set of initial training may not sufficient, and continuous adaption may be required. In this use case, data privacy is an issue, as the driving data directly corresponds and is being mapped to a given person. As the driving data is also unique to this person, federated learning leveraging data and training from other vehicles would not be suitable. Therefore, local-only edge-based training would allow for such data privacy. Even as hardware can be a limitation, the proposed system presents a solution.

5.8.5.3 Real-Time Stream

False Positive Rate. Based on the test-rig real-time inference, false positives can become a significant problem if not appropriately addressed. With a model trained on a higher batch size, where the updates are less likely to represent outlier benign samples, a model with a higher false positive rate is produced. For example, increasing the batch size when training the fully-connected speedometer model on the test rig data to 1024, the false positive rate increases to 2%. While this may not be significant at a glance, there is a high frequency of CAN messages. Therefore, in a typical driving stream of the length of the recorded test set, roughly 2% of messages may create false flags if the same type of benign messages appear. When looking instead at the reconstruction losses per sample, in both the main models and the higher batch size example, the unpredictability of the CAN stream can determine that if a specific benign sample occurs, a false flag is raised. Based on this observation, any false positive rate can lead to frequent false flags depending on the driving scenario. This is demonstrated with the test rig during real time inference, where the higher false positive rate causes frequent false positives during benign driving conditions to a degree that can render the model unsuitable. When individual samples are better represented with a lower batch size, the more optimized model is observed to what can be described as "rare" false positive occurrences. For example, on a single real-time inference simulation, one small cluster of false positives is typically observed. It can be argued that even a small increase in the false positive rate can be detrimental to the system due to how many CAN frames must be inferred in a period of time.

On-device OL can address this problem in future work by identifying better sample selection strategies that, when handled appropriately, can assist the model with benign samples that the model lacks confidence in. This may be paired with per-sample-adaptive hyperparameters. Larger test sets that cover various driving scenarios and styles would also provide better insights into which benign samples the model struggles to reconstruct as well as more reliable performance metrics. Additionally, false positives must be investigated past the surface level metric, as specific types of benign samples may not be represented enough during training. Therefore, while a model can appear to be learning well based on learning curves, the learning curves themselves are typically a reflection of quality of the model on the training and validation set as a whole.

Real Time Attack Behavior. To provide context on the chosen CAN ID, this paragraph highlights analysis on behavior of the system after real-time attacks. Replay attacks on both the specified CAN ID and the network as a whole are chosen. For this, the network is initially sniffed and packets are recorded. As the selected CAN ID, 0x1A0, changes with the acceleration and deceleration of the vehicle, the replay attack involves replaying a message at an earlier point in time while the speed is zero.

For the general network attack, network activity is recorded for a duration of a few seconds. This recording captures all CAN traffic.

The results varied depending on whether the simulation was controlling the dashboard, as this would change the messages that were being sent into the network. The effects of the attacks typically involved suppressing warning signals on the instrument as well as instances where the speedometer values changed. If this attack was conducted in a real vehicle, various malfunctions could occur as transmissions from other ECUs are suppressed while the malicious transmitted messages takeover the vehicle. Ultimately, based on these observations, while an attack may be understood, its effects may still be unpredictable.

5.9 Summary

This section provides a brief summary of this chapter, rounding off this main work.

To summarize, as demonstrated by previous work and expanded upon in this study, the practice of TinyML is suitable for the application of vehicular intrusion detection systems as they successfully address the challenges laid out in section 1.1. Specifically, small deployable models, and those with applied compression techniques, can achieve the time restraints of the CAN bus as well as meet the hardware requirements of microcontrollers. This is observed in the computational metrics, where the fully-connect model, and quantized convolutional models, can be deemed suitable based on their inference times.

Additionally, this is further met when combined with TinyOL, which enables on-device adaptation. In this, the additional fully-connected model training time remains appropriate across devices, where the IDS latency remains suitable for the task. Despite less learning power as updates are restricted to the output layer, the autoencoder can successfully improve at reconstructing benign samples over time based on an incoming CAN stream in both stochastic and mini batch workflows. This is primarily observed in the stochastic workflow, where significant improvements are made to the model.

Lastly, the importance of false positives is highlighted. It is observed that even a reliable model of high quality, based on a high accuracy and f1-score, can be considered unsuitable depending on the false positive rate. Additionally, the importance of more granular evaluations, focusing on a per-sample basis, is emphasized. Especially when considering false positives, incoming samples are not predictable and incorrect predictions can be costly given the limited time constraints.

Chapter 6

Conclusions

This chapter concludes the thesis as a whole. Future work is presented followed by the concluding remarks.

6.1 Future Work

The systems demonstrated and the ideas derived in this work propose edge-based adaptive systems for vehicular networks. Primarily, the main contribution of this work lays groundwork to an adaptive IDS for the CAN bus by combining TinyOL [16] with TinyML. Based on what has been considered out of scope, or observations from the contributions, various future works has been identified.

6.1.1 Continual Learning in EV Infrastructure

Future research in the continual learning for EV infrastructure may beign with alternative learning objectives. For example, classification methods would enable diagnostic and analysis applications, allowing for interpretation of the network activity. This may involve incremental continual learning, where the architecture is altered per new task, with more precise labeling. Naturally, this would require methods to identify what label a new task sample would require. For example, this may be conducted with clustering or other statistical methods to determine labels prior to training. New task training sessions may also involve multiple new tasks or new labels. The continual learning use case may also benefit to expanding towards a degree of federated learning, enabling the option of updates from other sources.

Additionally, a model and its complexity may only handle a certain number of tasks until its performance begins to deteriorate across all tasks. Future work may involve a "limit test" to determine how many new tasks a model can handle in both the anomaly detection autoencoder and classification objectives. This would also introduce further work into the trade-offs between relevant variables, such as model complexity, computational efficiency, and long-term performance. This may lead to other issues, as full continual learning may not be supported on more resource constrained hardware, especially for models that have a higher level of complexity. Additionally, the hardware choice would determine batch size limitations which would influence the continual learning updates and ability to utilize a replay set.

Lastly, expanding the system evaluations to other EV infrastructure data or other vehicular networks alongside deployment to real EV infrastructure hardware would ensure a more complete system is explored. This would aid in determining its usability for real-world use. For example, evaluating a diagnostic-focused continual learning use case on real EV infrastructure hardware to detect and analyze faults and non-benign behavior.

6.1.2 Efficient TinyML IDS

To develop more efficient deep learning models for intrusion detection systems, future work may include a comprehensive investigation of compression techniques alongside acceleration techniques in the use case of intrusion detection systems for vehicular networks. For example, implementing an IDS which features convolutional layers alongside hardware and software options that enable the acceleration of convolutional computation. Pruning and alternative quantization techniques with compatible systems presents directions to further reduce computation times and enable more constraint deployment options. For example, an IDS that takes advantage of the sparsity of pruning. Alternatively, Neural Architecture Search (NAS) [47] techniques can automatically and computationally derive more optimal architecture configurations, ensuring tiny IDS models are as efficient as possible while still holding reliable performance.

A system of small specialized models, such as those designed for specific CAN IDs, may derive better results in both performance and computation when compared to a one-size-fits-all solution. Therefore, exploring this system further for the CAN bus may include investigating how a model's architecture and preprocessing method may be specialized. Preprocessing may include window size, where a window size of 5 may derive worse results than a larger window size. Meanwhile, a larger window size may require a larger model with more specialized architecture, such as a 1D-CNN to handle the increased temporal complexity.

Similarly to what is conducted in Andreica et al. [42], deployment directly onto existing vehicle, or similar, hardware allows for evaluations on realistic hardware conditions. Future work should explore model deployment and evaluations directly on CAN infrastructure, which may include utilizing a real functioning CAN bus.

6.1.3 Adaptive TinyOL IDS

Future work in exploring OL and TinyOL for the CAN bus intrusion detection systems should explore its application in alternative learning objectives as well as investigate long-term effects, stability management methods, and on-device evaluations.

Alternative learning objectives may include driver identification for vehicle theft and classification focused applications for more precise detections or diagnostics. In the former, continuous adaptation enables the system to better understand a driver's behavior or adapt to multiple driver. In the latter, online learning may be conducted through incremental/continual learning to handle new data, such as unseen attacks.

The long-term affects of the OL are unpredictable, as it is dependent of the incoming data stream. Therefore, it would be insightful to investigate long-term effects where the OL has a negative affect on the model overall. Methods to manage stability and improve short-term and long-term training performance may include improved sample selection strategy. For example, training with samples that a model requires better confidence at reconstructing, and/or introducing the replay of key benign data. With this, the model may improve at what can be considered "outlier" benign samples while maintaining strong performance with traditional samples by mitigating any catastrophic forgetting. Additionally, to determine how precisely the OL will affect the model in the learning objectives in the real world, long-term evaluations should be conducted in a CAN system that has drifting data.

Lastly, conducting the OL experiments directly on the edge devices and/or within real CAN systems will further the evaluations and provide improved insight. This may also involve additional methods, such as varying training times with buffered samples, to perform the OL more effectively given the constraints and context of the vehicular use-case.

6.1.4 Closing the Gap

Real-time evaluations in both lab-based test rigs and real vehicles can bring adaptable intrusion detection systems closer to real-world-ready-use. Future work should deploy the systems directly onto the CAN bus with either real ECUs or off-the-shelf edge devices. For example, a detection system could be implemented within the microcontroller of the CAN ID's ECU, or be its own ECU on the network. This would allow for more purpose-driven design.

Furthermore, the next step after detection is response. It is just as important for an IDS to detect an attack than it is for it to handle the attack. Straying away from the side of detection, the implementation of efficient response techniques should be studied, as well as considering how responses may psychologically affect the driver.

6.2 Concluding Remarks

The significant threat posed by vehicular network attacks, particularly those targeting the CAN bus, has been made clear throughout the thesis. With the provided case studies (2.1.5) and literature (3, it is evident that the problem requires continuous investigation before the threat becomes a regular reality. In this thesis, we explore the challenges of adaptive and edge-based intrusion detection systems for vehicular networks in order to defend against malicious threats. In particular, we present a system that utilizes continual learning to reconstruct new attacks in electric vehicle infrastructure power consumption with a focus on justifying edge-only computing. This leads into the main contribution, where tiny machine learning and tiny online learning for CAN bus intrusion detection systems is explored. This IDS can facilitate edge-based computing through various deployment techniques and quantization. The demonstration of the efficient and edge-focused online learning strategy is aimed at enabling continuous adaptation for the IDS models in both drifting data or personalization tasks. In the first contribution, catastrophic forgetting was effectively mitigated through the replay of old samples. Additionally, an edge-only workflow presents itself as suitable for the use case. In the main contribution, three efficient model architectures are trained on three CAN IDs, enabling detection of CAN attacks on edge devices. The subsequent deployment suggests that the fully-connected model respects the latency of the CAN bus with suitable inference times on the Arduino platform, with quantization enabling this on the 1D-convolutional model. An early-stage model is capable of successfully refining the model with stream-based online learning, demonstrated by improving model metrics over time, such as an increased F1-score and reduced false positive rate. Lastly, the importance of the trade off between false positive rate and anomaly detection is highlighted, as well as the necessity to consider per-sample evaluations when determining model quality for a CAN bus IDS. Future work primarily involves refining the system and bridging the gap to real-world use. While the contributions are applied to their sole use cases, they are transferable and can be applied to various other learning objectives vehicular systems and edge-based applications.

References

[1] Transport for London, "Technical Note 12: How many cars are there in London?," rtf supporting document, Transport for London, 2023. Accessed: 2025-10-17.

[2] Department for Transport, "Nts 2023: Car availability and trends in car trips." https://www.gov.uk/government/statistics/national-travel-survey-2023/nts-2023-car-availability-and-trends-in-car-trips, 2025. Accessed: 2025-10-17.

[3] Department for Transport, "Vehicle licensing statistics: April to june 2024." https://www.gov.uk/government/statistics/vehicle-licensing-statistics-april-to-june-2024, 2024. Accessed: 2025-10-17.

[4] Autocar, "Average weight of new cars rises by nearly 400kg in seven years." https://www.autocar.co.uk/car-news/new-cars/average-weight-new-cars-rises-nearly-400kg-seven-years, 2024. Accessed 2025-09-02.

[5] World Health Organization, "Road traffic injuries." https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries, 2023. Accessed 2025-09-22.

[6] Department for Transport, "Reported road casualties, great britain: Annual report 2023," accredited official statistics, UK Government, Department for Transport, 2024. Accessed 2025-09-22.

[7] Moneybarn, "Carflation 2022." https://www.moneybarn.com/carflation/, 2022. Study revealing car prices are rising 1.94 times quicker than the average UK salary, with cars costing almost 39% more on average than in 2012.

[8] Market.us, "Smart car market size, share | cagr of 17.2 %." Market.us, 2024. Accessed: 2025-10-17.

[9] K. Tindell, "Can injection: keyless car theft," 2023. Accessed: 2025-08-18.

[10] A. Greenberg, "Hackers remotely kill a jeep on the highway—with me in it," *Wired*, vol. 7, no. 2, pp. 21–22, 2015.

[11] D. Schneider, "Jeep hacking 101," *IEEE Spectrum*, August 2015.

[12] H. Zhang, K. Zeng, and S. Lin, "Federated graph neural network for fast anomaly detection in controller area networks," *IEEE Trans. Inf. Forensics Secur.*, vol. 18, pp. 1566–1579, 2023.

[13] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-efficient learning of deep networks from decentralized data," 2016.

[14] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," 2018.

[15] M. Althunayyan, A. Javed, and O. Rana, "A robust multi-stage intrusion detection system for in-vehicle network security using hierarchical federated learning," *Vehicular Communications*, vol. 49, p. 100837, Oct. 2024.

[16] H. Ren, D. Anicic, and T. A. Runkler, "Tinyol: Tinyml with online-learning on microcontrollers," in *2021 International Joint Conference on Neural Networks (IJCNN)*, IEEE, July 2021.

[17] M. Falch, "Can bus explained – a simple intro." https://www.csselectronics.com/pages/can-bus-simple-intro-tutorial, 2025. Accessed: 2025-09-02.

[18] OpenGarages (Craig Smith), "Instrument Cluster Simulator (ICSim)." https://github.com/zombieCraig/ICSim, 2014. Accessed: 2025-09-02.

[19] S. Rajapaksha, H. Kalutarage, G. Madzudzo, A. Petrovski, and M. Al-Kadri, "Can-mirgu: A comprehensive can bus attack dataset from moving vehicles for intrusion detection system evaluation," in *Proceedings Symposium on Vehicle Security amp; Privacy*, VehicleSec 2024, Internet Society, 2024.

[20] J. Antoun, M. E. Kabir, B. Moussa, R. Atallah, and C. Assi, "A detailed security assessment of the ev charging ecosystem," *IEEE Network*, vol. 34, p. 200–207, May 2020.

[21] F. Makhmudov, D. Kilichev, U. Giyosov, and F. Akhmedov, "Online machine learning for intrusion detection in electric vehicle charging systems," *Mathematics*, vol. 13, p. 712, Feb. 2025.

[22] E. D. Buedi, A. A. Ghorbani, S. Dadkhah, and R. L. Ferreira, "Enhancing ev charging station security using a multi-dimensional dataset: Cicevse2024," Mar. 2024.

[23] J. E. Rubio, C. Alcaraz, and J. Lopez, "Addressing security in ocpp: Protection against man-in-the-middle attacks," in *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, p. 1–5, IEEE, Feb. 2018.

[24] M. Multin, "Iso 15118 as the enabler of vehicle-to-grid applications," in *2018 International Conference of Electrical and Electronic Technologies for Automotive*, IEEE, July 2018.

[25] R. Porter, M. Biglari-Abhari, B. Tan, and D. Thrimawithana, "Enhancing security in the iso 15118-20 ev charging system," *Green Energy and Intelligent Transportation*, p. 100262, Jan. 2025.

[26] M. Almehdhar, A. Albaseer, M. A. Khan, M. Abdallah, H. Menouar, S. Al-Kuwari, and A. Al-Fuqaha, "Deep learning in the fast lane: A survey on advanced intrusion detection systems for intelligent vehicle networks," *IEEE Open Journal of Vehicular Technology*, vol. 5, p. 869–906, 2024.

[27] E. C. P. Neto, H. Taslimasa, S. Dadkhah, S. Iqbal, P. Xiong, T. Rahman, and A. A. Ghorbani, "Ciciov2024: Advancing realistic ids approaches against dos and spoofing attack in iov can bus," *Internet of Things*, vol. 26, p. 101209, July 2024.

[28] A. Alfardus and D. B. Rawat, "Intrusion detection system for can bus in-vehicle network based on machine learning algorithms," in *2021 IEEE 12th Annual Ubiquitous Computing, Electronics amp; Mobile Communication Conference (UEMCON)*, p. 0944–0949, IEEE, Dec. 2021.

[29] G. Dangwal, M. Wazid, S. Nizam, V. Chamola, and A. K. Das, "Automotive cybersecurity scheme for intrusion detection in CAN-driven artificial intelligence of things," *Secur. Priv.*, vol. 8, Jan. 2025.

[30] R. Gundu and M. Maleki, "Securing can bus in connected and autonomous vehicles using supervised machine learning approaches," in *2022 IEEE International Conference on Electro Information Technology (eIT)*, p. 042–046, IEEE, May 2022.

[31] R. Rai, J. Grover, P. Sharma, and A. Pareek, "Securing the CAN bus using deep learning for intrusion detection in vehicles," *Scientific Reports*, vol. 15, p. 13820, Apr. 2025.

[32] H. M. Song, J. Woo, and H. K. Kim, "In-vehicle network intrusion detection using deep convolutional neural network," *Vehicular Communications*, vol. 21, p. 100198, Jan. 2020.

[33] J. Shi, Z. Xie, L. Dong, X. Jiang, and X. Jin, "Ids-dec: A novel intrusion detection for can bus traffic based on deep embedded clustering," *Vehicular Communications*, vol. 49, p. 100830, Oct. 2024.

[34] H. Xu, X. Shi, H. Liu, Y. Wang, J. Lu, H. Zeng, R. Li, and D. Wu, "Mulsam: Multidimensional attention with hardware acceleration for efficient intrusion detection on vehicular can bus," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, p. 1–1, 2025.

[35] J. Ashraf, A. D. Bakhshi, N. Moustafa, H. Khurshid, A. Javed, and A. Beheshti, "Novel deep learning-enabled lstm autoencoder architecture for discovering anomalous events from intelligent transportation systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, p. 4507–4518, July 2021.

[36] W. Aljabri, M. A. Hamid, and R. Mosli, "Lightweight and adaptive data-driven intrusion detection system for autonomous vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 26, p. 2282–2292, Feb. 2025.

[37] J. A. Khan, D.-W. Lim, and Y.-S. Kim, "A deep learning-based ids for automotive theft detection for in-vehicle can bus," *IEEE Access*, vol. 11, p. 112814–112829, 2023.

[38] P.-Y. Tseng, P.-C. Lin, and E. Kristianto, "Vehicle theft detection by generative adversarial networks on driving behavior," *Engineering Applications of Artificial Intelligence*, vol. 117, p. 105571, Jan. 2023.

[39] K. H. Park and H. K. Kim, “This car is mine!: Automobile theft countermeasure leveraging driver identification with generative adversarial networks,” 2019.

[40] S. Ahmed, O. Esbel, M. Mühlhäuser, and A. S. Guinea, “Towards continual knowledge learning of vehicle can-data,” in *2023 IEEE Intelligent Vehicles Symposium (IV)*, p. 1–6, IEEE, June 2023.

[41] H. Im and S. Lee, “Tinyml-based intrusion detection system for in-vehicle network using convolutional neural network on embedded devices,” *IEEE Embedded Systems Letters*, vol. 17, p. 67–70, Apr. 2025.

[42] T. Andreica, C.-D. Curiac, C. Jichici, and B. Groza, “Android head units vs. in-vehicle ecus: Performance assessment for deploying in-vehicle intrusion detection systems for the can bus,” *IEEE Access*, vol. 10, p. 95161–95178, 2022.

[43] H. Xu, D. Wu, Y. Lu, J. Lu, and H. Zeng, “Models on the move: Towards feasible embedded AI for intrusion detection on vehicular CAN bus,” in *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, (Santa Clara, CA), pp. 1049–1063, USENIX Association, July 2024.

[44] S. Khandelwal and S. Shreejith, “Seccan: An extended can controller with embedded intrusion detection,” *IEEE Embedded Systems Letters*, p. 1–1, 2025.

[45] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, “Array programming with NumPy,” *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[46] B. Thorne and contributors, “python-can: Controller area network (can) interface library for python.” https://python-can.readthedocs.io/, 2024. Version 4.3.0.

[47] T. Elsken, J. H. Metzen, and F. Hutter, “Neural architecture search: A survey,” 2018.