THESIS

ANOMALY DETECTION WITH MACHINE LEARNING FOR AUTOMOTIVE CYBER-PHYSICAL SYSTEMS

Submitted by

Sooryaa Vignesh Thiruloga

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Spring 2022

Doctoral Committee:

Advisor: Sudeep Pasricha

Ryan Kim
Indrakshi Ray

ABSTRACT

ANOMALY DETECTION WITH MACHINE LEARNING FOR AUTOMOTIVE CYBER-PHYSICAL SYSTEMS

Today's automotive systems are evolving at a rapid pace and there has been a seismic shift in automotive technology in the past few years. Automakers are racing to redefine the automobile as a fully autonomous and connected system. As a result, new technologies such as advanced driver assistance systems (ADAS), vehicle-to-vehicle (V2V), 5G vehicle to infrastructure (V2I), and vehicle to everything (V2X), etc. have emerged in recent years. These advances have resulted in increased responsibilities for the electronic control units (ECUs) in the vehicles, requiring a more sophisticated in-vehicle network to address the growing communication needs of ECUs with each other and external subsystems. This in turn has transformed modern vehicles into a complex distributed cyber-physical system. The ever-growing connectivity to external systems in such vehicles is introducing new challenges, related to the increasing vulnerability of such vehicles to various cyber-attacks. A malicious actor can use various access points in a vehicle, e.g., Bluetooth and USB ports, telematic systems, and OBD-II ports, to gain unauthorized access to the in-vehicle network. These access points are used to gain access to the network from the vehicle's attack surface. After gaining access to the in-vehicle network through an attack surface, a malicious actor can inject or alter messages on the network to try to take control of the vehicle.

Traditional security mechanisms such as firewalls only detect simple attacks as they do not have the ability to detect more complex attacks. With the increasing complexity of vehicles, the

attack surface increases, paving the way for more complex and novel attacks in the future. Thus, there is a need for an advanced attack detection solution that can actively monitor the in-vehicle network and detect complex cyber-attacks. One of the many approaches to achieve this is by using an intrusion detection system (IDS). Many state-of-the-art IDS employ machine learning algorithms to detect cyber-attacks for its ability to detect both previously observed as well as novel attack patterns. Moreover, the large availability of in-vehicle network data and increasing computational power of the ECUs to handle emerging complex automotive tasks facilitates the use of machine learning models. Therefore, due to its large spectrum of attack coverage and ability to detect complex attack patterns, we adopt and propose two novel machine learning based IDS frameworks (*LATTE* and *TENET*) for in-vehicle network anomaly detection.

Our proposed *LATTE* framework uses sequence models, such as LSTMs, in an unsupervised setting to learn the normal system behavior. *LATTE* leverages the learned information at runtime to detect anomalies by observing for any deviations from the learned normal behavior. Our proposed *LATTE* framework aims to maximize the anomaly detection accuracy, precision, and recall while minimizing the false-positive rate.

The increased complexity of automotive systems has resulted in very long term dependencies between messages which cannot be effectively captured by LSTMs. Hence to overcome this problem, we proposed a novel IDS framework called *TENET*. *TENET* employs a novel convolutional neural attention (TCNA) based architecture to effectively learn very-long term dependencies between messages in an in-vehicle network during the training phase and leverage the learned information in combination with a decision tree classifier to detect anomalous messages. Our work aims to efficiently detect a multitude of attacks in the in-vehicle network with low memory and computational overhead on the ECU.

ACKNOWLEDGEMENTS

I would like to thank all the individuals whose encouragement and support have made the completion of this thesis possible.

First and foremost, I would like to express my sincere gratitude to my advisor, Prof. Sudeep Pasricha, who has guided me through the process of this study with his insightful and timely advice. I can with complete confidence say that it was only with his encouragement and motivation I was able to explore the exciting domain of in-vehicle network cybersecurity. He ensured that I was always on the right track and have all the necessary resources for my research. He nurtured the analytical mindset and gave me sufficient time and opportunities to realize my true potential in accomplishing my M.S. goals. I would like to thank Prof. Sudeep Pasricha for all the help, guidance, inspiration, and encouragement.

I would like to take this opportunity to thank the respected members of my thesis committee,
Prof. Ryan Kim, and Prof. Indrakshi Ray. Their feedback helped me to refine my work from different perspectives.

Furthermore, my special thanks to my research partner Vipin Kumar Kukkala, whose collaboration helped me understand the research challenges in my initial days of maters studies by gaining valuable insights on in-vehicle network cybersecurity and more over for his support and cooperation through intellectually stimulating conversations that boosted my morale while solving complex research problems. This list cannot be complete without mentioning the company and the help from my current and former lab mates in Prof. Pasricha's EPIC lab:

Abhishek Balasubramanium, Joydeep Dey, Saideep Tiku, Ninad Hogade, Shoumik Maiti, Febin Sunny, Asif Anwar Baig Mirza, Kamil Khan, and Liping Wang.

I am blessed to have a wonderful family – my father Thiruloga Chander Prakasam, my mother Revathy Thiruloga Chander, and my sister Mrudha Sri Thiruloga – for their continuous support that empowered me to pursue my master's degree. Their generosity and humility have made me continually strive to be a better person.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF RESEARCH PUBLICATIONS

JOURNAL PUBLICATIONS:

- V. K. Kukkala, S. V. Thiruloga, and S. Pasricha, "Roadmap for Cybersecurity in Autonomous Vehicles", **IEEE Consumer Electronics Magazine (CEM)** (under review)**, 2021**.
- V. K. Kukkala, S. V. Thiruloga, and S. Pasricha, "LATTE: LSTM Self-Attention based Anomaly Detection in Embedded Automotive Platforms", **in ACM Transactions on Embedded Computing Systems (TECS), Volume 20, Issue 5, pp 1–23, 2021**.
- V. K. Kukkala, S. V. Thiruloga, and S. Pasricha, "INDRA: Intrusion Detection using Recurrent Autoencoders in Automotive Embedded Systems", **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, (TCAD), 39(11), Nov 2020**.

CONFERENCE PUBLICATIONS:

- S. V. Thiruloga, V. K. Kukkala, and S. Pasricha, "TENET: Temporal CNN with Attention for Anomaly Detection in Automotive Cyber-Physical Systems", **IEEE/ACM Asia & South Pacific Design Automation Conference (ASPDAC)** *(to appear)***, 2022**.

BOOK CHAPTER:

- V. K. Kukkala, S. V. Thiruloga, and S. Pasricha, "AI for Cybersecurity in Distributed Automotive IoT Systems", **Electronic Design for AI, IoT and Hardware Security** *(to appear)***, Springer, 2022**.

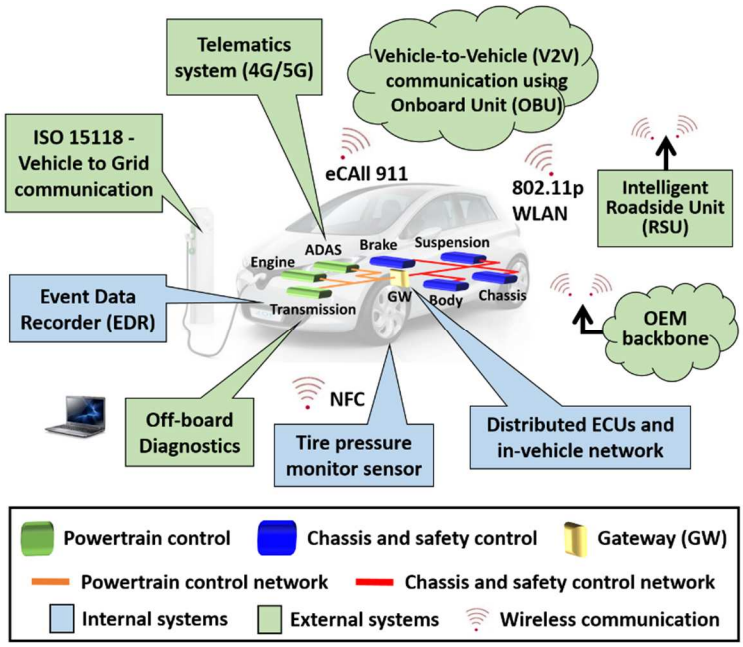
1. INTRODUCTION

This chapter provides an overview of the ongoing transformation of the automotive industry and the necessity for securing future vehicles. Autonomous vehicles are on the horizon and will be transforming transportation safety and comfort. These vehicles will be connected to various external systems and utilize advanced cyber-physical systems (CPS) to perceive their environment and make intelligent decisions. However, this increased connectivity makes these vehicles vulnerable to various cyber-attacks that can have catastrophic effects. Attacks on automotive systems are already on the rise in today's vehicles and are expected to become more commonplace in future autonomous vehicles. Thus, there is a need to strengthen cybersecurity in future autonomous vehicles. This chapter also gives a general overview of the contributions of this thesis.

1.1. OVERVIEW AND MOTIVATION

The aggressive attempts of automakers to make vehicles fully autonomous have resulted in increased software and hardware complexity across automotive subsystems. Many state-of-the-art automotive subsystems for collision avoidance, lane keep assist, pedestrian and traffic sign detection, etc., demand powerful CPS, typically referred to as Electronic Control Units (ECUs), to be integrated into the vehicles. To meet the needs across various subsystems, a diverse set of ECUs consisting of different compute and memory capacities are used in today's vehicles. The ECUs are distributed across the vehicle and communicate using an in-vehicle network. Several in-vehicle network protocols are used in modern vehicles to meet the data rate, timing, and reliability requirements of automotive subsystems. Some of the most commonly used in-vehicle

network protocols include controller area network (CAN), local interconnect network (LIN), FlexRay, and Ethernet. Both ECUs and in-vehicle networks are becoming more complex to



satisfy emerging autonomy needs.

Figure 1 Illustration of an in-vehicle network connecting different ECUs using isolated networks that are connected to a gateway (GW), and external connectivity of modern vehicles with various systems in the environment.

Additionally, a variety of automotive subsystems rely heavily on the data from external systems as shown in Figure 1, which makes modern vehicles highly vulnerable to various security attacks. In the past decade (2010 onwards), nearly 79.6% of all automotive attacks have been remote attacks, which do not require the attacker to be within the vicinity of the vehicle [1]. A variety of attack vectors have been used including wireless fidelity (WiFi), telematics, Bluetooth, keyless entry systems, and mobile applications. We discuss many of these attacks in the next subsection, as well as techniques that have been proposed to protect vehicles from cyber-attacks. However, due to the overall increase of the automotive system complexity

(heterogeneous ECUs, network architectures/protocols, and applications), detecting cyber-attacks is not easy, which poses a major challenge for emerging connected and autonomous vehicles (CAVs). There is a critical need for a monitoring solution that can serve as an intrusion detection system (IDS) to detect cyber-attacks in vehicles. Traditionally, such IDSs in computing systems have relied on using firewalls, or rule-based systems to detect cyber-attacks. These simple systems cannot detect highly complex modern automotive attacks.

Another interesting trend in modern vehicles is the widespread adoption of artificial intelligence (AI) techniques for advanced driver assistance subsystems (ADAS), where environmental perception is required [2]. Such AI techniques can also be deployed in powerful automotive ECUs to monitor and detect cyber-attacks. AI-based solutions are well known to be highly efficient in learning the complex patterns that exist in high dimensional time-series vehicular network data. They can observe anomalous patterns on in-vehicle networks that connect all in-vehicle and external subsystems, to detect cyber-attacks. *With fully autonomous vehicles supporting increased connectivity to external subsystems on the horizon, having an efficient IDS that can detect a variety of cyber-attacks using AI techniques is crucial and an urgent requirement*.

1.2. HISTORY OF AUTOMOTIVE CYBER-ATTACKS

Several automotive attacks have been observed in the past that ranged from targeting a single stationary vehicle to a fleet of vehicles on the road. Here we present a timeline of the major automotive attacks, in Figure 2, and discuss their impacts. The researchers at the University of California at San Diego and the University of Washington demonstrated one of the first vehicle hacks in 2010 [3]. They exploited the onboard telematics system and reverse-

engineered the system and were able to gain full control of the vehicle. The researchers however worked with the manufacturer and did not disclose the full details of the vulnerability. Needless to say, this hack opened up a Pandora's box. Several other works followed this approach and tried to reverse engineer the ECUs in the vehicles in the following years. However, all of these attacks required the attacker to be physically present inside the target vehicle, which resulted in dismissing as an unlikely situation and these hacks not gaining much traction in the media.

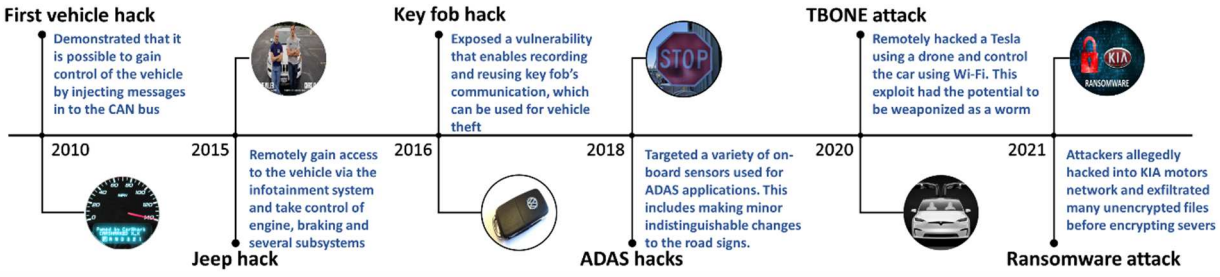


Figure 2 Timeline of major automotive cyber-attacks

This changed in 2015 when the first major remote attack was demonstrated on an unaltered 2014 Jeep Cherokee [4] by two security researchers. The researchers identified a software bug in the vehicle's infotainment system that would allow them to connect to the vehicle remotely over the 4G LTE and send CAN messages to the ECUs in the vehicle. They demonstrated a wide range of attacks ranging from remotely controlling simple functionality such as the vehicle radio, A/C, and windshield wipers to more critical functionality such as controlling brakes, transmission, and even killing the engine while the vehicle was on a freeway. The attackers were able to launch these attacks on a remote vehicle from their homes. This hack created a huge media outburst and the manufacturer had to issue a patch to fix the bug. The same researchers in 2016 exposed another bug that let them remotely control the acceleration, steering wheel, and cruise control systems. Similar attacks came into light in 2016, where an attacker was able to

remotely control a Nissan Leaf in England from Australia. These attacks changed the landscape of how automotive cyber-attacks were carried out and highlighted the urgency to address cybersecurity in vehicles.

Starting around 2016, a new type of attack emerged that focused on hacking the keyless entry system in vehicles. The goal of these attacks was to steal the vehicle rather than remotely control it. The researchers at the University of Birmingham showed how they were able to recover the cryptographic algorithms and keys from the ECUs and clone the Volkswagen group remote control by eavesdropping on a single signal sent by the original remote [5]. Several other attacks have emerged that targeted mobile applications that were used for remote start and immobilizer systems in vehicles. Some of the recent attacks in this class include cloning the Tesla model S key fob in 2018 [6] and the model X key fob in 2019 [7]. Researchers were able to clone key fobs by capturing the Bluetooth communication between the key fob and the body control module (BCM) and were able to use a bootleg BCM to replay it and steal the vehicle in under 90 seconds.

A different class of attacks has gained popularity since 2018, mainly targeting the ADAS systems and the onboard sensors used for perception. In [8], researchers generated various robust visual adversarial perturbations to a stop sign that resulted in it being misidentified as a 45-mph speed limit sign. A few years before this, researchers were able to blind a Mobileye C2-270 camera and demonstrate jamming, spoofing, and relay attacks on an Ibeo LUX3 LiDAR sensor [9]. More recent attacks include tricking lane change system of a Tesla Model S with bright stickers on the road by Tencent Keen security lab in 2019 [10] and object removal attacks on LiDAR sensors in 2021 [11]. Another recent attack that made the headlines was the T-BONE attack [12] where researchers were able to gain remote code execution (RCE) over WiFi on the

infotainment system in a Tesla Model 3 using a drone. They were able to remotely open doors and trunk, change seat positions, steering, and acceleration modes. However, this exploit does not provide driving control of the vehicle. The researchers also highlighted that adding a privilege escalation exploit such as CVE-2021-3347 to T-BONE would weaponize this exploit and turn it into a worm. This would allow them to load new WiFi firmware and exploit other Tesla cars in the victim car's proximity. More recently, at the beginning of 2021, an online hacking group by the name Doppel Paymer claimed to conduct a ransomware attack on KIA motors America and have stolen unencrypted confidential data [13].

1.3. THESIS OVERVIEW

In summary, there is a crucial need for a holistic framework as an IDS that can learn the normal vehicle behavior at design time and monitor the network for anomalies at runtime. Such a framework is not easy to conceptualize because of the increasing complexity of automotive systems. With the increasing complexity, the attack surface is only going to increase, paving the way for more complex and novel attacks in the future. Traditional security mechanisms such as firewalls only detect simple attacks and do not have the ability to detect more complex attacks such as those in [14], [15]. To address the above-mentioned problems, the main contribution of this thesis is the design of two novel IDS frameworks called *LATTE* [16] and *TENET* [17] to actively monitor the in-vehicle network and observe for any deviation from the normal behavior to detect novel and complex attack patterns. The rest of this thesis is organized as follows:

In chapter 2, we start by introducing essential concepts to intrusion detection. We define what a network attack is, what an intrusion detection system is. Different types of IDSs are detailed, with their strengths and weaknesses. We provide an overview of the automotive system

and formally define the communication model considered in this thesis. We then define the different types of complex attacks that can occur in an automotive network. Moreover, we explain the contributions of deep learning in this field and provide a background of state-of-the-art deep learning techniques for intrusion detection. We present existing works related to intrusion detection for automotive systems. Following this, we identify the limitations of existing works and provide ideas on how to improve. The insights gathered in this chapter are used in the design of the IDSs in the following chapters.

In chapter 3, we present a novel anomaly detection framework called *LATTE* to detect cyber-attacks in the Controller Area Network (CAN) based automotive networks [16]. *LATTE* uses a novel stacked LSTM with self-attention architecture that learns the normal system behavior by learning to predict the next message instance under normal operating conditions. We presented a one class support vector (OCSVM) based detector model to detect cyber-attacks by monitoring the message deviations from the normal behavior. We present a detailed analysis by comparing our proposed model with multiple variants of our model and the best-known prior works in this area.

In chapter 4, we present another novel anomaly detection framework called *TENET* [17] for automotive systems based on Temporal Convolutional Neural Attention (*TCNA*) networks to efficiently learn very long-term dependencies between in-vehicle network messages with low memory and computational overhead and accurately detect a multitude of simple, complex, and novel attacks. We also proposed a metric called the divergence score (*DS*), which measures the deviation of the predicted signal value from the actual signal value. A decision tree (*DT*) based classifier was used to learn the model deviations that correspond to the normal vehicle operation at design time. At runtime, the trained *TCNA* and *DT* models were used to observe for deviations

using the deviation score metric to detect cyber-attacks. We compared our framework with the best-known prior works that employ a variety of sequence model architectures for anomaly detection.

Chapter 5 concludes this thesis. We summarize our research in this chapter and also make recommendations for future work.

2. BACKGROUND AND RELATED WORK

This chapter is divided into three major sections. In section 2.1, we provide a background on *(i)* basic concepts of intrusion detection systems, *(ii)* automotive system and communication model considered for our proposed frameworks in chapters 3 and 4, *(iii)* complex attacks considered for evaluating our proposed works, and *(iii)* advanced deep learning based techniques for modeling complex data. In section 2.2, we present multiple existing IDS solutions proposed by researchers to detect attacks on automotive networks. Finally, section 2.3 addresses the limitations of the existing IDS solutions and presents a set of goals that need to be achieved to overcome those limitations.

2.1. BACKGROUND

In this section, we start by introducing the essential concepts of the intrusion detection system in the context of in vehicle networks in subsection 2.1.1. We provide a complete picture of the automotive system (subsection 2.1.2); communication model (subsection 2.1.3) considered for our works and discuss how IDSs can be deployed for anomaly detection in the automotive network. Subsection 2.1.4 defines the different types of complex attacks that can occur in an automotive network. In subsections 2.1.5 and 2.1.6, we explore state-of-the-art deep learning techniques for intrusion detection.

2.1.1 INTRUSION DETECTION SYSTEM (IDS)

A cyber-attack (or an intrusion) is defined as all unauthorized activities that compromise one, two, or all of these three fundamental components of an information system. The three

components are (i) confidentiality, (ii) integrity, and (iii) availability. It is also known as the CIA triad as shown in Figure 4.

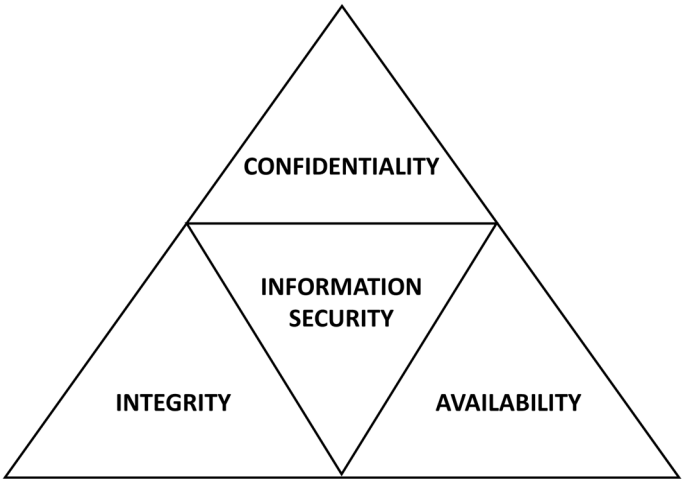


Figure 3 CIA Triad

- **Confidentiality:** it is defined as the "property that information is not made available or disclosed to unauthorized individuals, entities, or processes" [18] by the ISO 27000 standard. This information could be credit card numbers, personal data, or any information considered private. The challenge of confidentiality is to allow legitimate users to access this information while preventing others from doing so. An example of a confidentiality attack would be when a malicious attacker executes a ransomware attack [13] to gain access to confidential data from the vehicle. Failure of confidentiality results in a data breach that cannot be remedied but can be managed in a way that minimizes its impact on users. To ensure the confidentiality property different security mechanisms are implemented such as encryption, two-factor authentication, passwords, etc. Depending on the level of confidentiality of the information, the strength of the security measure is

decided. Confidential information can also be protected by staking multiple layers of protection.

- **Integrity:** It is defined as the "property of accuracy and completeness" [18] by the ISO 27000 standard. The Integrity property ensures that the information is protected from unauthorized parties, and it also ensures that no accidental deletion or modification of information is made by authorized parties. An example of an information integrity attack would be gaining unauthorized access to the in vehicle network. A malicious attacker can gain access to the in-vehicle network through the infotainment systems or wireless interfaces to modify the contents of the network and gain control of the vehicle. Moreover, information can be changed by non-human caused events such as server crash or electromagnetic pulse. Integrity property is commonly ensured using checksums, encryption, and hashing techniques. Redundant systems and backup procedures are important security mechanisms to ensure data integrity.
- **Availability:** It is defined as the "property of being accessible and usable on demand by an authorized entity" [18] by the ISO 27000 standard. The information which is not available at the right time can have serious consequences. Denial of Service (DoS) or Distributed Denial of Service (DDoS) attacks are common attacks against availability. For example, the attackers try to flood the in-vehicle network with malicious messages rendering the ECUs connected in the network useless for legitimate users. This malicious traffic is often detected and blocked by security mechanisms such as firewalls and Intrusion Detection Systems (IDSs).

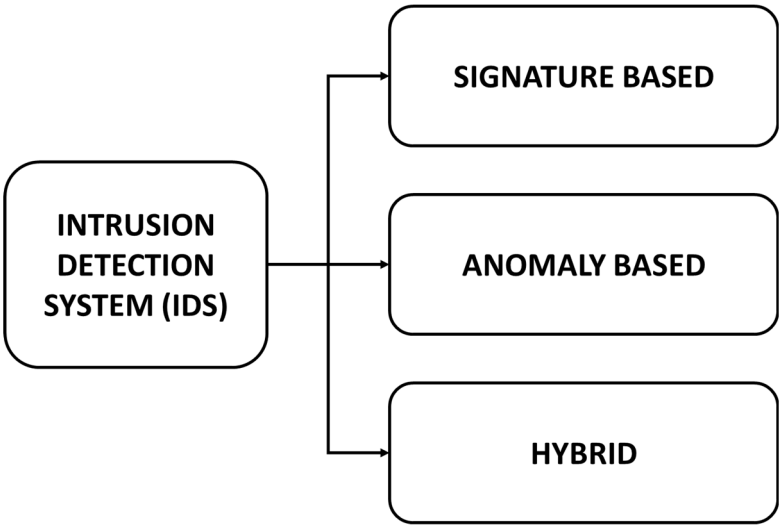


Figure 4 IDS Classification

Monitoring network traffic to detect malicious or unauthorized activities is a process called "intrusion detection" and the system that is used to monitor the network for intrusions is called the intrusion detection system (IDS). An IDS can be a hardware or software-based system that continuously monitors the in-vehicle network to detect attacks without any human supervision. An IDS transforms monitored activities into alerts using its knowledge (database, statistics, artificial intelligence, etc.). IDSs can be classified according to the detection method they use. They fall into three categories: signature-based detection, anomaly-based detection, and hybrid detection as shown in Figure 4.

- **Signature-based detection** (also known as "heuristic based detection") comes with a database of known attack signatures. It compares monitored data with previously observed attack patterns from the signature database. A signature-based IDS checks the input stream for the presence of an attack pattern. To be efficient, the database of this kind of IDSs must be updated regularly. However, even with the latest updates, only known attacks can be detected using this method.

- **Anomaly-based detection** tries to learn a “normal” or “expected” behavior of the system. Any deviation from this behavior is considered a potential attack and will generate an alarm. This method does not require updates or even the presence of a database. It can identify unknown attacks but also creates a lot of false positives that are difficult to process. It is also more difficult to collect information about the attack since it is not clearly identified by a signature.
- **Hybrid detection** combines the two solutions to mitigate weaknesses of each category: anomaly detection then misuse detection, misuse detection then anomaly detection, or both at the same time. The goal is to detect known attacks with their signatures and to use anomaly detection to identify unknown intrusions.

An efficient IDS needs to be reliable, lightweight, robust, and scalable with different system sizes. Moreover, a pragmatic IDS needs to have a large coverage of attacks (able to detect both known and unknown attacks), high confidence in detection, and a low false positive rate as recovery from false positives can be expensive. Since getting the signature of every possible attack is impractical and would limit us to only detecting known attacks and a hybrid detection system suffer slower detection latency with large computational overhead on the ECU, we conjecture that using anomaly-based IDS is a more practical approach to this problem. Additionally, due to the ease of in vehicle network data acquisition (from test driving), there can be a large amount of in-vehicle message data to work with, which facilitates the use of advanced deep learning models for detecting the presence of an attacker in the system. In the next subsections, we discuss the design of modern automotive systems by providing an overview of the system model, communication model, and attack model considered for our proposed IDSs.

2.1.2 SYSTEM OVERVIEW

We consider a generic automotive system consisting of multiple ECUs connected using an in-vehicle network, as shown in Figure 5. Each ECU is responsible for running a set of automotive applications that are hard-real time in nature, meaning they have strict timing and deadline constraints. In addition, we assume that each ECU also executes intrusion detection applications that are responsible for monitoring and detecting intrusions in the in-vehicle network.

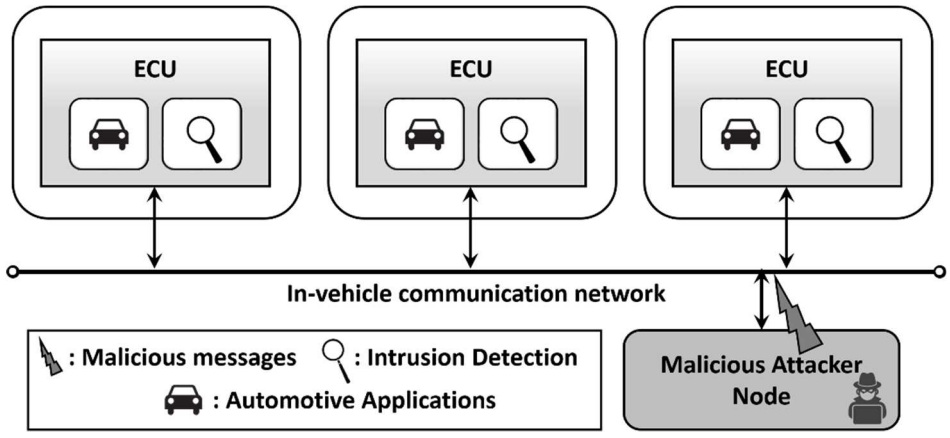


Figure 5 Overview of automotive system

We consider a distributed IDS approach (intrusion applications collocated with automotive applications) as opposed to a centralized IDS approach where one central ECU handles all intrusion detection tasks due to the following reasons:

- A centralized IDS approach is prone to single-point failures, which can completely open up the system to the attacker.

- In extreme scenarios such as during a DDoS attack (explained in subsection 2.1.4), the in-vehicle network can get highly congested, and the centralized system might not be able to communicate with the victim ECUs.
- If an attacker succeeds in fooling the centralized IDS ECU, attacks can go undetected by the other ECUs, resulting in compromising the entire system; whereas with a distributed IDS, fooling multiple ECUs is required which is much harder, and even if an ECU is compromised, this can still be detected by the decentralized intelligence.
- In a distributed IDS, ECUs can stop accepting messages as soon as an intrusion is detected without waiting for a centralized system to notify them, leading to faster response.
- The computation load of IDS is split among the ECUs with a distributed IDS, and the monitoring can be limited to only the required messages. Thus, multiple ECUs can monitor a subset of messages independently, with lower overhead.

Many prior works, e.g., in [19] and [20], consider a distributed IDS approach for these reasons. Moreover, with automotive ECUs becoming increasingly powerful, the collocation of IDS applications with real-time automotive applications in a distributed manner should not be a problem, provided the overhead from the IDS is minimal. The design of an IDS should have low susceptibility to noise, low cost, and a low power/energy footprint.

2.1.3 COMMUNICATION MODEL

In this subsection, we discuss the vehicle communication model that was considered for our proposed IDSs in chapters 3 and 4. We perform experimental analysis related to detecting

intrusions in a CAN bus-based automotive system, although our IDSs are broadly applicable to any vehicle network protocol.

Controller Area Network (CAN) is the defacto industry standard in-vehicle network protocol for automotive systems today. CAN is a lightweight, low-cost, and event-triggered communication protocol that transmits messages in the form of frames. The structure of a standard CAN frame is shown in Figure 6 and the length of each field (in bits) is shown on the top.

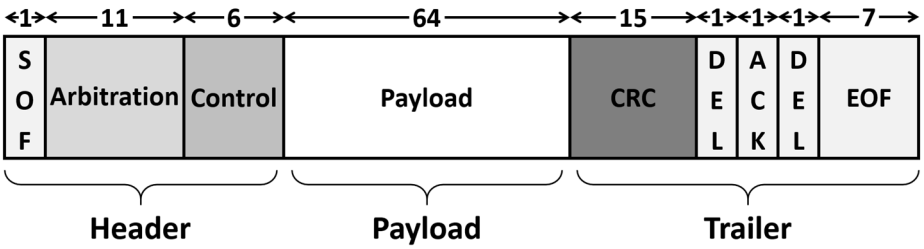


Figure 6 CAN frame format.

The standard CAN frame consists of a header, payload, and trailer segments. The header consists of information such as the message identifier (ID) and the length of the message. The actual data that needs to be transmitted is in the payload segment. The trailer section is mainly used for error checking at the receiver. A variation of the standard CAN, called CAN-extended or CAN 2.0B is also becoming increasingly common in modern vehicles. The major difference is that CAN extended has a 29-bit identifier allowing for more number of message IDs. For the proposed IDSs in chapters 3 and 4, we design with a focus on monitoring the message payload and observing for anomalies to detect intrusions. This is because an attacker needs to modify the message payload to accomplish a malicious activity. While an attacker could target the header or trailer segments, it would result in the message getting rejected at the receiver. The payload

segment consists of multiple data entities called signals. An example real-world CAN message with the signals is shown in Figure 7 [21]. Each signal has a fixed size (in bits), a particular data type, and a start bit that specifies its location in the 64-bit payload segment of the CAN message. In this thesis, all proposed IDSs focus on monitoring individual signals within message payloads to observe for anomalies and detect intrusions. Signal level monitoring would give us the capability to not only detect the presence of an intruder but also help in identifying the signal within the message that is being targeted during an attack. This can be valuable information for understanding the intentions of the attacker, which can be used for developing countermeasures.

Signal Name	Message	Start bit	Length	Byte Order	Value Type
Battery_Current	Status	0	16	Intel	Signed
Battery_Voltage	Status	16	16	Intel	Unsigned
Motor_Current	Status	32	16	Intel	Signed
Motor_Speed	Status	48	8	Intel	Signed
Motor_Direction	Status	56	8	Intel	Unsigned

Figure 7 Real-world CAN message with signal information.

2.1.4 ATTACK MODEL

We assume that attackers can gain access to the in-vehicle network using the most common threat vectors such as connecting to the vehicle OBD-II port, probing into the in-vehicle network, and via advanced threat vectors such as connected V2X ADAS systems, insecure infotainment systems, or by replacing a trusted ECU with a malicious ECU. We also assume that the attacker has access to the in-vehicle network parameters such as flow control, BAUD rate, parity, channel information, etc. that can be obtained by a simple CAN data logger and can help in the transmission of malicious messages. We further assume a pessimistic situation where the attacker can access the in-vehicle network at any instance and try to send malicious messages.

Given the above assumptions, our proposed anomaly detection system tries to protect the in-vehicle network from the multiple types of cyber-attacks listed below. These attacks are modeled based on the most common and hard-to-detect attacks in the automotive domain.

- **Constant attack (Plateau attack):** In this attack, the attacker overwrites the signal value to a constant value for the entire duration of the attack interval. The complexity of detection of this attack depends on the change in magnitude of signal value. Intuitively, a small change in the magnitude of the signal value is harder to detect than larger changes.
- **Continuous attack:** In this attack, the attacker tries to trick the anomaly detection system by continuously overwriting the signal value in small increments until a target value is achieved. The complexity of detecting this attack depends on the rate of change of the signal value. Larger change rates are easier to detect than smaller rates.
- **Replay attack (Playback attack):** In this attack, the attacker plays back a valid message transmission from the past, tricking the anomaly detector into believing it to be a valid message. The complexity of detecting this attack depends mainly on the frequency and sometimes on the duration of the playbacks. High-frequency replays are easier to detect compared to low-frequency replays.
- **Dropping attack (Suppress attack):** In this attack, the attacker disables the transmission of a message or group of messages resulting in missing or dropping of communication frames. The complexity of detecting this attack depends on the duration for which the messages are disabled. Longer durations are easier to detect due to missing message frames for a prolonged time compared to shorter durations.

- **Distributed Denial of Service (DDoS) attack (Flooding attack):** In this attack, the attacker floods the in-vehicle network with an arbitrary or specific message with the goal of increasing the overall bus load and rendering the bus unusable for other ECUs. This is the most common and easy-to-launch attack as it requires no information about the nature of the message. These attacks are fairly simple to detect even using a rule-based approach as the message frequencies are fixed and known at design time for automotive systems. Any deviation in this message rate can be used as an indicator for detecting this attack.

With the availability of large troves of data in vehicles from communication between ECUs and the increased computing capabilities of ECUs, AI-based (deep learning) IDS solutions can be leveraged to parse high dimensional vehicular network data to detect intrusions. In the next subsection, we provide a background on some of the state-of-the-art deep learning techniques employed in this domain.

2.1.5 NEURAL NETWORKS

A neural network is a computing system, inspired by the biological neural networks in the human brain. It consists of a collection of simulated neurons, each of which is the basic unit of a neural network. The neurons are arranged in multiple layers to form the multiple layers of the neural network and can be divided into *input*, *hidden*, *output* layers as shown in Figure 8. A neural network with many hidden layers is known as a deep neural network. A Feed Forward neural network is the simplest form of artificial neural network in which the connections between nodes do not form a cycle as information is only processed in one direction. While the data may pass through multiple hidden nodes, it always moves in one direction and never backward. A

neural network interprets its input data in many ways, three of which are particularly interesting for intrusion detection: *classification*, *regression*, and *reconstruction*.

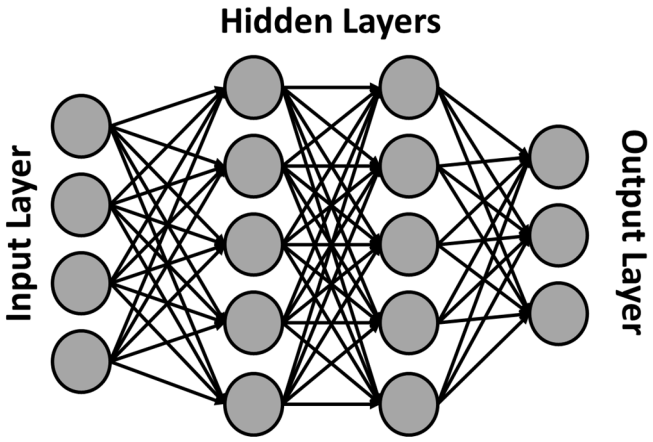


Figure 8 Example of an artificial neural network with 2 hidden layers.

Classification categorizes input samples into several classes, such as "normal" or "anomalous", or even different families of attacks. Regression (also called "prediction") is used to determine continuous values, including a probability that input is an attack. Finally, reconstruction is specific to a certain type of neural network. This task tries to reconstruct the input data by compressing and decompressing them to force the network to learn the features (representation learning). The interpreting of data is known as *learning* and can be classified into two categories: *(i)* supervised learning and *(ii)* unsupervised learning. In a former approach, the neural network receives the input data with its corresponding label (in our case, anomalous or normal). The neural network learns the distribution of the input data during training and tries to predict the target output. Classification and regression are two classic supervised training tasks. In unsupervised learning, the input data does not contain any labels and is classified by understanding interesting patterns within the input data. Reconstruction is an example of an unsupervised task. In a reconstruction-based anomaly detection task, the neural network learns to

replicate its inputs during training by minimizing the replication error. The replication error is calculated in an unsupervised fashion by measuring the deviation between the input and the replicated output. During runtime, the trained model tries to replicate its inputs and the replication error is measured. If the replication error is higher than a threshold value, the respective input sample is flagged as an anomaly.

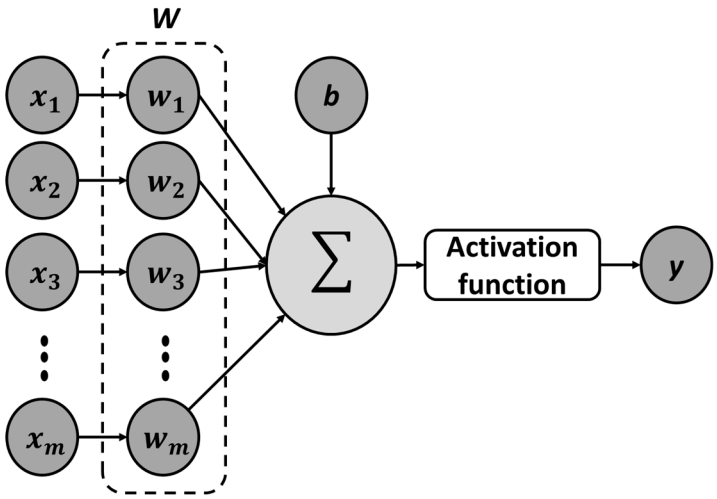


Figure 9 A single artificial neural network neuron model.

Each neuron in a neural network receives a set of inputs $\{x_1, x_2, x_3, \dots, x_m\}$ where m is the total number of inputs to the neuron and computes the predicted y value. Vector x contains the values of the features in one of the examples from the input dataset. Each unit has its set of parameters, normally known as W, column vector of weights $\{w_1, w_2, w_3, \dots, w_m\}$, and b (bias) as shown in Figure 9. These parameters are updated during the learning process. In each iteration, every neuron in a layer calculates a weighted average of the values of the vector x, based on its current weight vector w, and adds bias b. The result of this calculation is passed through an activation function to calculate the output y. In artificial neural networks, the activation function of a node defines the output of that node given an input or set of inputs. Some examples of

activation functions are sigmoid, hyperbolic tangent (tanh), rectified linear unit (ReLU), softmax. These activation functions induce nonlinearity in the network. The loss function is the function that computes the distance between the current output of the algorithm and the expected output. It's a method to evaluate how your algorithm models the data, for example, mean squared error. A neural network learns the input features by minimizing this loss function. Hence, to achieve this goal the gradient of the loss function is calculated, and it is then used to update the weight of each connection and the bias of each hidden neuron. This part of the learning is famously known as the "backpropagation algorithm". During backpropagation, these gradients are used to adjust the *w* and *b* parameters of each neuron. This process of inputs traveling through the network is called forward pass and the process of backpropagating the loss to update the weight parameters is called the backward pass. In an in-vehicle network, the communication between ECUs happens promptly. Hence, there exists a lot of temporal relationships between the messages that can be exploited to detect intrusions. However, this cannot be achieved using the traditional feed-forward neural networks as the output of any input is independent of other inputs. Thus, it makes it harder to capture the temporal dependencies between the messages using feedforward neural networks. This led to the study of sequence models that makes a perfect choice for handling time-series data.

2.1.6 SEQUENCE MODELS

A sequence model can be thought of as a function that ensures that the output is dependent not only on the current input but also on the previous inputs. The first sequence model called the recurrent neural network (RNN) was introduced and implemented in [22].

Recurrent Neural Networks (RNN) is a type of neural network which makes use of sequential information. RNNs have hidden states, which allows the information to persist. These hidden states enable the RNN to connect previous information to their current inputs. Thus, it provides us with a solution for our need for a model that can capture the relation between the inputs of a sequence. The mathematical representation is shown in equation (1). A basic RNN is a neural network with feedback (shown in Figure 10).

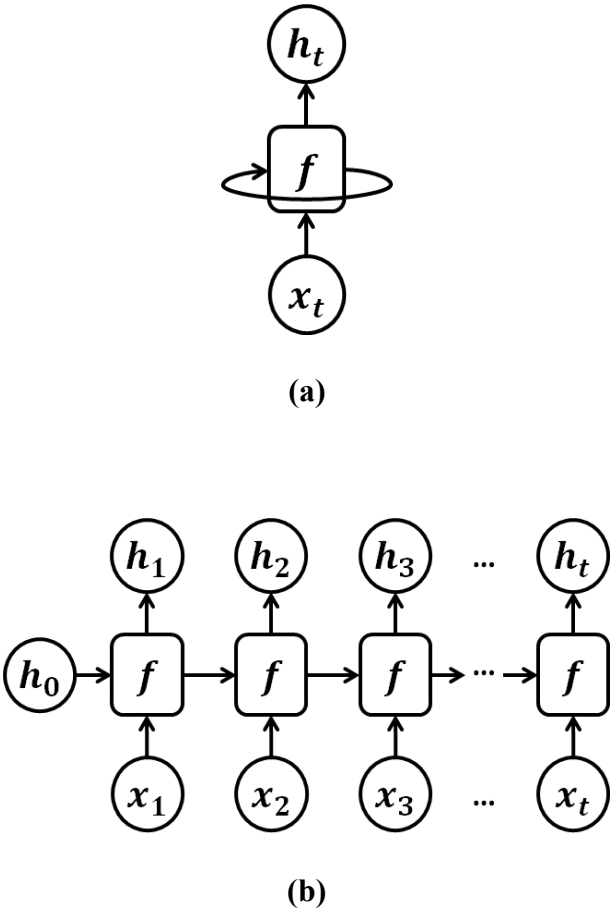


Figure 10 (a) A single RNN cell and (b) unrolled RNN unit; where, *f*-RNN cell, *x*-input, and *h*-hidden state.

The output h_t is a function of both the input x_t and the previous output h_{t-1}:

$$\boldsymbol{h_t = f(Wx_t + Uh_{t-1} + b)} \qquad (1)$$

where *W*, *U* are weight matrices, *b* is a bias term, and *f* is a nonlinear activation function (e.g. the sigmoid or hyperbolic-tangent function). One of the limitations of RNNs is that they are hard to train. Since RNNs and other sequence models deal with sequence or time-series inputs, the back propagation happens through various time samples and is hence known as back propagation through time. During this process, the feedback loop in RNNs causes the errors to shrink or grow rapidly (vanishing or exploding gradients respectively), destroying the information in the backpropagation. This problem of vanishing gradients hampers the RNNs from learning long term dependencies. This was solved in [23], by introducing additional states and gates in the RNN cell to remember long term dependencies, which lead to the birth of Long Short-Term Memory (LSTM) Networks.

To understand LSTMs, we first need to know what a cell state is and how the gates are used to modify them. The cell state can be thought of as a transport highway, that carries relevant information throughout the processing of a sequence to accommodate information from the earlier time steps which can be used in the later time steps. This reduces the effects of short-term memory. The information in the cell state is modified via gates. Hence, the gates in LSTM help the model decide which information has to be retained and which information to forget. An LSTM cell consists of three gates (i) forget gate (ii) input gate (iii) output gate as shown in Figure 11 (a). The first gate is a binary gate; it chooses which information to retain from the previous cell state. The second gate adds relevant information to the cell state.

$$\boldsymbol{i_t = sigmoid(W_i x_t + U_i h_{t-1} + b_i)} \quad (2)$$

$$\boldsymbol{f_t = sigmoid(W_f x_t + U_f h_{t-1} + b_f)} \quad (3)$$

Here W_i, W_f, U_i, U_f is the learned weight matrices and b_i, b_f are bias terms, h_{t-1} is the output of the previous hidden state and x_t is the current state input. The subscript *i, f,* and *o* represent input

forget and output gates respectively. x_t, h_{t-1} is multiplied with weight matrices and added bias in the input and forget gates, shown in equations (2) and (3).

$$\tilde{\boldsymbol{c}} = \boldsymbol{tanh}(\boldsymbol{W_c x_t} + \boldsymbol{U_c h_{t-1}} + \boldsymbol{b_c}) \quad (4)$$

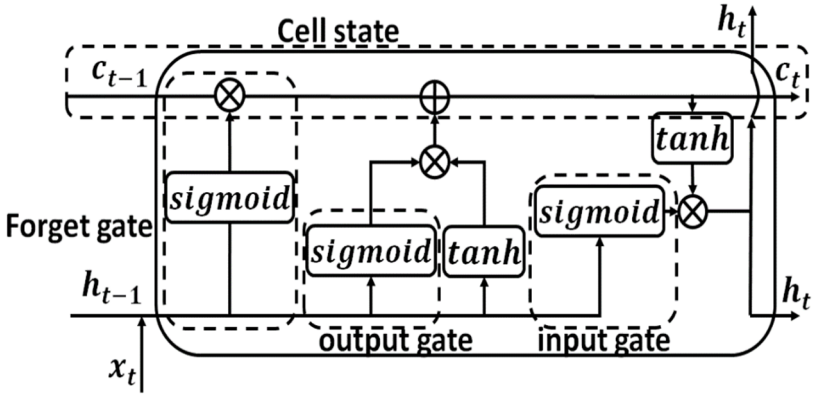
$$\boldsymbol{c_t} = \boldsymbol{f_c c_{t-1}} + \boldsymbol{i} * \tilde{\boldsymbol{c}} \quad (5)$$

i_t and f_t determine the new cell state as a linear combination of the previous internal state c_{t-1} and new candidate internal state $\tilde{c}$, shown in equations (4) and (5).

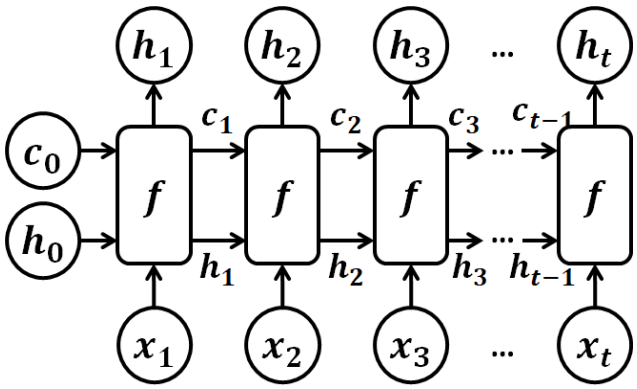
$$\boldsymbol{o_t} = \boldsymbol{sigmoid}(\boldsymbol{W_o x_t} + \boldsymbol{U_o h_{t-1}} + \boldsymbol{b_o}) \quad (6)$$

$$\boldsymbol{h_t} = \boldsymbol{o_t}\, \boldsymbol{tanh}(\boldsymbol{c_t}) \quad (7)$$

Here W_c, W_o, U_c, U_o is the learned weight matrices and b_c, b_o are bias terms. The output layer is also controlled by a gate function and uses all the information from the last two layers to produce an output. x_t, h_{t-1} is multiplied with weight matrices and added bias in the output gate, shown in equation (6). The current hidden layer h_t is the result of the multiplication of o_t and tanh applied current cell state c_t as shown in equation (7).



(a)



(b)

Figure 11 (a) A single LSTM cell with different gates and (b) unrolled LSTM unit; where, *f*-LSTM cell, *x*-input, *c*-cell state, and *h*-hidden state.

In conclusion, LSTMs can learn long term dependencies of a sequence. However, they are not computationally efficient as the sequence path has become more complicated due to the addition of gates. It also takes a lot of resources to train these networks faster and has a memory component associated with it. In [24] the authors presented a simpler recurrent neural network than LSTMs, which trains faster but also remembers long sequences of data with low memory overhead and can solve the vanishing gradient problem and is called a Gated Recurrent Unit (GRU).

A GRU cell uses an alternate route for gating information when compared to the LSTM networks. It combines the Input and forget gate layers of the LSTM network into a solitary update layer and furthermore combines hidden and cell state layers. Figure 12 shows a traditional GRU cell has two gate layers (i) reset gate (ii) update gate. The reset gate combines new input with past memory and the update layer chooses the amount of pertinent data that should be held.

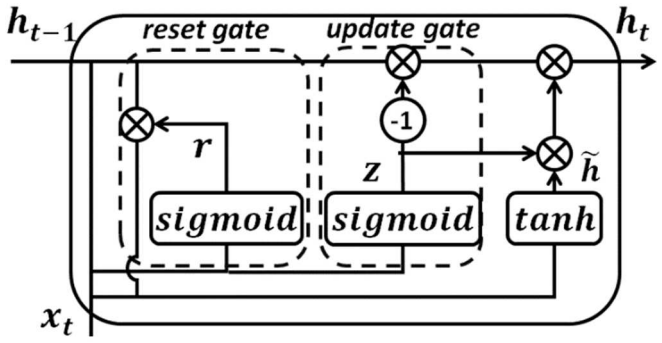
$$\boldsymbol{r = sigmoid(x_t W^r + h_{t-1} U^r)} \quad (8)$$

$$\boldsymbol{z = sigmoid(x_t W^z + h_{t-1} U^z)} \quad (9)$$

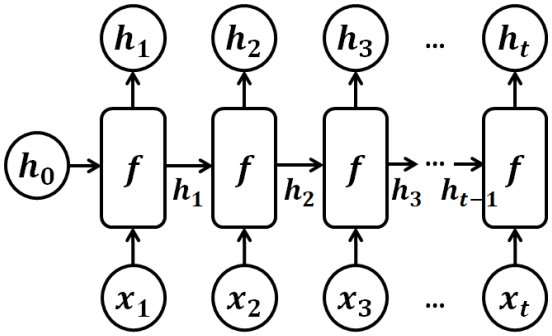
$$\tilde{\boldsymbol{h}} = \boldsymbol{tanh}(\boldsymbol{x_t W^h} + (\boldsymbol{h_{t-1}} * \boldsymbol{r})\boldsymbol{U^h}) \quad (10)$$

$$\boldsymbol{h_t} = (\boldsymbol{1} - \boldsymbol{z}) * \tilde{\boldsymbol{h}} + \boldsymbol{z} * \boldsymbol{h_{t-1}} \quad (11)$$

where, *W*, *U* are the learned weight matrices and *r*, *z* is the reset and update layers of the GRU cell. h_{t-1} is the output of the previous hidden state and x_t is the current state input. In the update gate, *z* inputs are multiplied with weight matrix *W,* and the hidden state is also multiplied with weight matrix *U*. A sigmoid function is applied to the sum of this multiplication. A similar process happens in the rest gate with different weight matrices as shown in equations (8) and (9). The current memory content $\tilde{h}$, uses the reset gate to store the relevant information from the past. This is performed by doing element-wise multiplication of h_{t-1} and *r* and then the results are summed with the input x_t. Tanh activation is applied to the summed result to obtain $\tilde{h}$ as shown in equation (10). In the last step, to calculate the current hidden state h_t, the update gate is used to collect important information from h_{t-1}, and $\tilde{h}$. This is done by applying element wise multiplication between the update gates and h_{t-1}, $\tilde{h}$, shown in equation (11). These results are summed to obtain the h_t vector, which holds information for the current unit and passes to the network. Thus, a GRU cell can control the data stream like an LSTM by uncovering its hidden layer contents. Moreover, GRUs achieve this using fewer gates and states, which makes them computationally more efficient with low memory overhead. As real-time automotive ECUs are highly resource-constrained systems with tight energy and power budgets, it is critical to use low overhead models for inferencing tasks.



(a)



(b)

Figure 12 (a) A single GRU cell with different gates and (b) unrolled GRU unit; where, *f*-GRU cell, *x*-input, and *h*-hidden state.

Due to the increase in the number of sensors and ECUs in modern vehicles, the complexity of the in-vehicle network has increased tremendously. This increased complexity results in very long dependencies between the messages which cannot be effectively captured using LSTMs and GRUs. This is mainly because the current time step output of both LSTMs and GRUs is heavily influenced by the recent time steps compared to past time steps, which makes it hard to capture very long term dependencies, e.g., for sequence lengths of 50 or more. Processing very long sequences also exacerbate the memory overhead of LSTMs and GRUs. These shortcomings have led to the exploration of novel sequence models using convolutional neural networks (CNN) in recent years.

CNNs are widely used in the areas of computer vision and other image-based learning applications. CNNs consist of convolutional layers and a set of *filters* for each convolution layer that is used to identify various features in the input. A filter can be thought of as a tensor that is multiplied with a subset of the input data to compute the convolution output. The filter is moved over the inputs to compute many such convolution outputs, known as *feature maps*. The dimension of the filter is known as *kernel size* and the number of steps moved by the filter due to its sliding operation is called the *stride*. In the case where the inputs are an image, the subsequent feature map values are calculated according to equation (12) , where the input image is denoted by *f* and our kernel by *h*. The indexes of rows and columns of the result matrix are marked with *m* and *n* respectively.

$$\boldsymbol{G[m,n] = (f * h)[m,n] = \sum_j \sum_k h[j,k]\, f[m-j, n-k]} \qquad (12)$$

The dimensions of the received tensor meet equation (13), in which *n* is the input size, *f* is the filter size, n_c is the number of channels of the input, *p* is the padding value used, *s* is the stride value, n_f is the number of filters.

$$\boldsymbol{[n,n,n_c] * [f,f,n_c] = \left[\left[\frac{n+2p-f}{s} + 1\right], \left[\frac{n+2p-f}{s} + 1\right], n_f\right]} \qquad (13)$$

An early adaptation of CNNs for sequence modeling tasks was presented in [25], where a convolution based time-delay neural network (TDNN) was proposed for phoneme recognition and compared against hidden markov models (HMM). To capture very-long term dependencies, traditional CNNs need to employ a very deep network of CNN layers with large filters. Consequently, this increases the number of convolutional operations incurring a high computational overhead. Thus, adapting CNNs directly to sequence modeling tasks in resource constrained automotive systems is not a feasible solution. However, recent promising advances

in this area have enabled the use of CNNs to capture very long term dependencies in time series data.

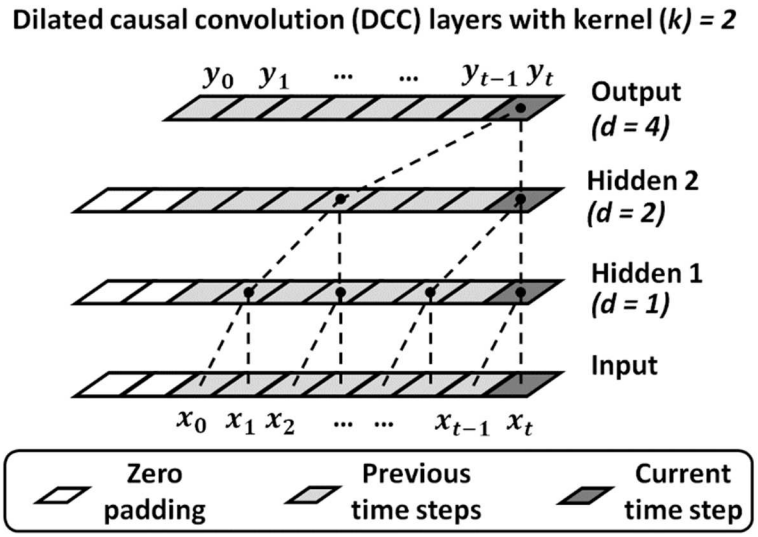


Figure 13 Example CNN based sequence network using dilated causal convolutional layers with dilation factors (d) of 1, 2, and 4.

The authors in [26] proposed a temporal convolutional network (TCN) architecture that uses dilated causal convolution (DCC) layers for sequence modeling tasks. The operation of the DCC layers in a simple network with two hidden layers is illustrated in Figure 13. The terms $\{x_0, x_1, x_2, \ldots, x_t\}$ represents the input time series data, and $\{y_0, y_1, y_2, \ldots, y_t\}$ represents the output sequence at the output layer. The subscript for each term in the input and output represents the associated time step. Each DCC layer has a kernel size (k) of 2 indicated by the dotted lines, and the dilation factor (d) of 1, 2, 4 for the hidden layers 1, 2, and the output DCC layer, respectively. The dilation factor dictates the number of input samples to be skipped by each DCC filter. The total number of samples influencing the output at a particular time step is called the receptive

field. In the network shown in Figure 13, the output at y_t is influenced by 8 input samples, and hence the receptive field is 8.

The TCN architecture has two interesting features. Firstly, the TCN uses a one-dimensional (1-D) fully convolutional layer, where each hidden layer is the same length as the input layer. Additionally, zero paddings are added to keep the subsequent layers the same length as the previous layer. Thus the architecture produces an output that is of the same length as the input which is required for sequence modeling tasks. Secondly, the output at time *t* is obtained by performing the convolution operations on inputs at time t and outputs from earlier time steps. This causal nature of the architecture ensures no information leakage from the future into the past, during training. Formally, for a 1-D sequence of inputs, $X = \{x \in \mathbb{R}\}$, and a filter $f: \{0, \ldots, k-1\}$, the dilated convolution operation F on element s of the sequence is defined as in equation (14):

$$\boldsymbol{F(s) = \sum_{i=0}^{k-1} f(i).\, x_{s-(d*i)}} \qquad (14)$$

where d is the dilation factor, k is the filter size and the subscript $s - (d * i)$ represents the number of elements in the sequence considered for computing the output. Moreover, the dilated convolution behaves like a normal convolution operation when $d=1$. Using a larger dilation value enables an output at the top level to represent a wider range of inputs, which helps the TCN in effectively learning very-long term dependencies. Moreover, the receptive field of the TCN can be increased either by changing the filter size k and dilation factor d or by increasing the depth (number of DCC layers) of the network (l). Unlike recurrent architectures, CNNs do not have to wait for the previous time step output to process the input at the current time step. Thus, TCNs can process the input sequences in parallel as compared to sequential processing in

recurrent architectures. This makes TCN more computationally efficient while training and testing when compared to recurrent networks.

2.2. RELATED WORK

Several techniques have been proposed to design IDS for time-critical automotive systems. The goal of these works is to detect various types of attacks in automotive systems by monitoring the in-vehicle network traffic. These works can be mainly classified into two groups (based on the previously mentioned IDS categories): *(i)* signature-based IDS and *(ii)* anomaly-based IDS.

Signature-based IDS relies on detecting known and pre-modeled attack signatures. The messages in the system are compared to the known attack signatures and observed for any matching patterns to detect intrusions. The authors in [19], [27], used a language theory-based model to derive attack signatures. However, this technique fails to detect intrusions when it misses the packets transmitted during the early stages of an attack. In [28], the authors used transition matrices that have the paired sequences of CAN IDs as the elements, to check for mismatches between the incoming CAN packet and its sequence to detect intrusions. Despite achieving a low false-positive rate for trivial attacks, this technique failed to detect more realistic replay attacks. The authors in [29], [30] identify notable attack patterns such as an increase in message frequency and missing messages to detect intrusions. The authors in [31] measured the time interval between the request and response messages in the CAN bus to detect attacks. However, the model is not scalable as different in-vehicle network protocols have different time intervals. A specification-based methodology to detect intrusions is proposed in [32], where the authors analyze the behavior of the system and compare it with the predefined attack patterns to

detect intrusions. Nonetheless, their system fails to detect unknown attacks. The authors in [33] proposed an IDS technique using Myers algorithm [34] under the map-reduce framework to speed up the pattern matching algorithm. The authors in [35] proposed a framework that utilizes the frequency of CAN messages to perform a time-frequency analysis to detect multiple intrusions. A message frequency-based in-vehicle network monitoring approach was proposed in [20]. They define a rule-based regular operating mode region by analyzing the messages during design time and observes for deviations to detect anomalies. In [36] the authors measured the Hamming distance between messages to detect attacks. However, this approach incurs a high computational overhead on the ECUs. In [37], ECUs were fingerprinted using their voltage measurements during message transmission and reception. However, as this method is only applied at the physical layer, it cannot detect attacks at the application layer. In [38], the authors propose a fingerprinting based approach to model the relationship between the sender ECU's clock-skew and the messages and detect intrusions by observing for variations in the clock-skew during runtime. In [39] the authors presented a formal analysis for clock-skew based IDS and evaluated their approach on a real vehicle. In [40] a memory heat map based is used to characterize the memory behavior of the operating system to detect intrusions. The authors in [41] propose an entropy-based IDS, that observes for change in system entropy to detect intrusions. However, the technique fails to detect small scale attacks where the entropy change is minimal. The authors in [42] proposed a sliding window approach based on information entropy to detect attacks. However, their approach fails to detect complex attacks that modify the contents of a CAN message. In a nutshell, signature-based techniques offer a solution to the intrusion detection problem with low false positive rates but cannot detect more complex and novel attacks. Moreover, modeling signatures of every possible attack is impractical.

Anomaly-based IDS aims to learn the normal system behavior offline and observes for any deviation from the learned normal system behavior to detect intrusions. This approach has been widely adopted in many domains as it can detect both known and unknown attacks. However, in recent years this is being increasingly explored in the automotive domain to detect the presence of attackers in the in-vehicle networks. In [43], the authors propose a sensor-based IDS approach that utilizes attack detection sensors to monitor various events in the system to observe deviations from normal behavior. However, this approach is not only expensive but also suffers from a poor detection ratio. A one-class support vector Machine (OCSVM) based IDS, was proposed in [44] and was able to perfectly detect attacks that lasted more than 0.5s. However, this approach suffers from poor detection latency. In [45], the authors used four different nearest neighbor classifiers to distinguish between normal CAN payload and attack induced CAN payload. This technique was evaluated on low priority messages and showed poor performance for denial of service (DoS) and fuzzy attacks. In [46] the authors proposed a decision-tree based detection model to monitor the physical features of the vehicle to detect intrusions. However, this model is not realistic and suffers from high detection latencies. The authors in [47] proposed a hidden markov model (HMM) based technique that monitors the temporal relationships between messages to detect intrusions. They estimated log-likelihood based on the predictions of a regression model that was built on those temporal features. However, the detection latency of their model was not presented. A deep neural network (DNN) based approach was proposed to examine the messages in the in-vehicle networks in [48]. The authors mainly evaluated their approach on a low priority tire pressure monitoring system (TPMS), which makes it hard to adapt to high priority safety-critical powertrain applications. A deep autoencoder approach was proposed in [49] to detect attacks on the in-vehicle network. A LSTM based multi-message ID

detection model was proposed in [50]. However, the model architecture is highly complex, which incurs high overhead on the ECUs. In [51], the authors used LSTM based IDS to detect insertion and dropping attacks. The authors in [52] propose an LSTM based predictor model, that predicts the next time step message value at a bit level and observes for large variations in loss to detect intrusions. The authors in [53] proposed another LSTM based network to classify attacks in the in-vehicle network. They used a supervised learning method to train their LSTM classifier. However, their proposed model was not tested on complex attacks such as replay attacks. A GRU based autoencoder architecture was proposed to learn the normal system behavior in [54]. However, they used a static threshold to classify the messages which may not be able to capture non-linear behaviors. The authors in [55] proposed an LSTM based encoder-decoder architecture with a hierarchical attention mechanism to reconstruct input messages. They used a kernel density estimator (KDE) and k-nearest neighbors (KNN) to detect anomalies. However, the model does not target CAN networks and incurs a high memory overhead on the ECU. In [56], a replicator neural network based IDS was proposed to learn the normal patterns in CAN messages in the in-vehicle networks. But their model doesn't consider the correlation between message ids. A hybrid IDS was proposed in [57], which utilizes a specification-based system in the first stage and an RNN based model in the second stage to detect anomalies in time-series data. An LSTM in combination with a convolutional neural network (CNN) based approach was proposed in [58] to learn the dependencies between messages in a CAN network. However, the model was trained on a labeled dataset in a supervised fashion; due to the large volume of in-vehicle CAN message data, labeling the data is impractical.

2.3. LIMITATIONS OF EXISTING APPROACHES

Signature-based anomaly detection systems provide low-cost and high-speed detection techniques but fail to detect complex and new attacks. Additionally, modeling every possible attack signature is practically impossible, and hence these anomaly detection approaches have a limited scope. On the other hand, most of the existing machine learning based works attempt to increase the detection accuracy and attack coverage but, none of them offers a holistic system-level solution that is lightweight, fast, scalable, and reliable to detect multiple types of attacks for in-vehicle networks.

The following are some of the goals that we considered for our IDS:

- **Lightweight:** Intrusion detection tasks can incur overhead on the ECUs that could result in poor application performance or missed deadlines for real-time applications. This can be catastrophic in some cases. Hence, we aim to have a lightweight IDS that incurs low overhead on the system.
- **Few false positives:** This is a highly desired quality in any kind of IDS (even outside of the automotive domain), as handling false positives can become expensive very quickly. A good IDS needs to have a few false positives or false alarms.
- **Coverage:** This is the range of attacks an IDS can detect. A good IDS needs to be able to detect more than one type of attack. High coverage for IDS will make the system resilient to multiple attack surfaces.
- **Scalability:** This is an important requirement as emerging vehicles have increasing numbers of ECUs, and high software and network complexity. A good IDS should be highly scalable and be able to support multiple system sizes.

- **Learn very long term dependencies:** Due to the increase in the number of sensors and ECUs in modern vehicles, the complexity of the in-vehicle network has increased tremendously. This increased complexity has resulted in very long dependencies between the in-vehicle messages. Therefore, a good IDS must have the ability to process very long sequences with relatively low memory and runtime overhead and still achieve reasonably high performance.

3. LATTE: LSTM SELF-ATTENTION BASED ANOMALY DETECTION IN AUTOMOTIVE CYBER-PHYSICAL SYSTEMS

In this chapter, we present a novel anomaly detection framework called *LATTE* [16] to detect attacks in CAN based automotive networks. Our proposed *LATTE* [16] framework uses sequence models in deep learning in an unsupervised setting to learn the normal system behavior. *LATTE* [16] leverages that information at runtime to detect anomalies by observing for any deviations from the learned normal behavior. This is illustrated in Figure 14.

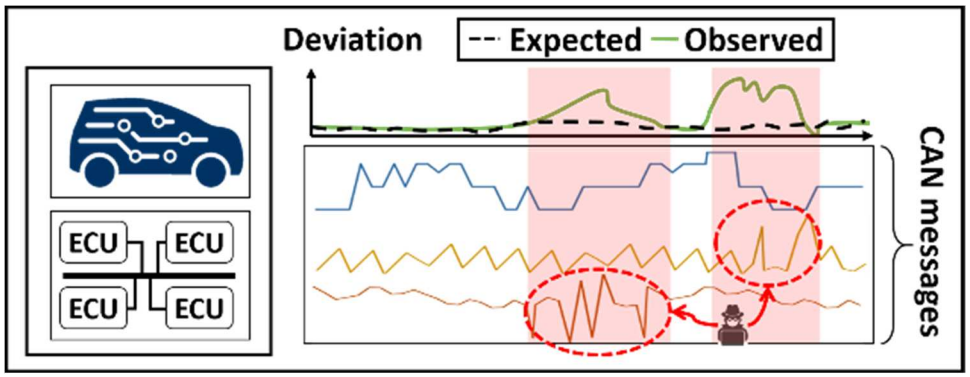


Figure 14 An example of an anomaly detection framework that monitors the network traffic and detects deviations from expected normal behavior during the attack intervals shown in red.

The plot on the top right shows the expected deviation (computed using the model that was trained at design time) vs the observed deviation. The divergence in signal values during the attack intervals (shown in the red area) can be used as a metric to detect cyber-attacks as anomalies. Our proposed *LATTE* [16] framework aims to maximize anomaly detection accuracy, precision, and recall while minimizing the false-positive rate. Our novel contributions in this work can be summarized as follows:

- We propose a stacked Long-Short Term Memory (LSTM) based predictor model that integrates a novel self-attention mechanism to learn the normal automotive system behavior at design time;
- We design a one class support vector machine (OCSVM) based detector that works with the LSTM self-attention predictor model to detect different cyber-attacks at runtime;
- We present modifications to existing vehicle communication controllers that can help in realizing the proposed anomaly detection system on a real-world ECU;
- We perform a comprehensive analysis on the selection of deviation measures that quantify the deviation from the normal system behavior;
- We explore several variants of our proposed *LATTE* framework and selected the best performing one, which is then compared with the best-known prior works in the area, to show *LATTE's* effectiveness.

3.1 LATTE FRAMEWORK: OVERVIEW

An overview of our proposed *LATTE* framework is shown in Figure 15. Our framework consists of a novel self-attention based LSTM deep learning model that is trained with data obtained from a data acquisition step. The data acquisition step collects trusted in-vehicle network data under a controlled environment. We then post-process and use this data to train the stacked LSTM self-attention predictor model in an unsupervised setting to learn the normal operating behavior of the system. We also developed a one class support vector machine (OCSVM) based detector model that utilizes the predictions from the LSTM predictor to detect

cyber-attacks as anomalies at run-time. After training, the framework is tested by being subjected to various attacks. The details of this framework are presented in the subsequent subsections.

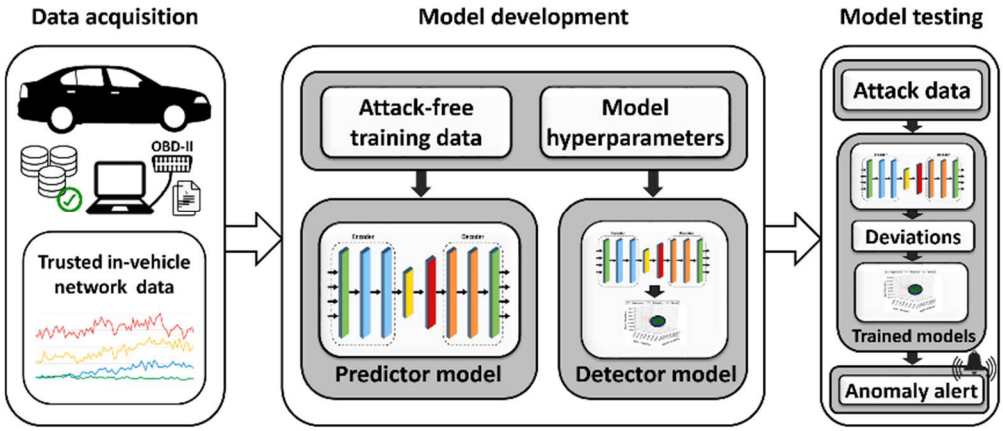


Figure 15 Overview of proposed *LATTE* framework.

3.1.1 DATA COLLECTION

This is the first step of the *LATTE* framework and involves collecting the in-vehicle network data from a trusted vehicle. It is important to ensure that the in-vehicle network and the ECUs in the vehicle are free from attackers. This is because the presence of an attacker can result in logging corrupt in-vehicle network data that falsely represents the normal operating conditions, leading to learning an inaccurate representation of the normal system behavior with our proposed models. Moreover, it is also crucial to cover a wide range of normal operating conditions and have the data collected over multiple intervals, to ensure high confidence in the collected data. The performance of the anomaly detection system is highly dependent on the quality of the collected data, and thus this is a crucial step. Additionally, the type of data collected depends on the functionalities or ECUs that are subjected to monitoring by the anomaly detection system. The most common access point to collect the in-vehicle network data is the OBD-II port, which gives access to the diagnostic and most commonly used messages. However, probing into the

CAN network and logging the messages is preferred, as it gives unrestricted access to the in-vehicle network, unlike the OBD-II port.

After collecting the message data from the in-vehicle network, the data is prepared for pre-processing to make it easier for the training models to learn the temporal relationships between messages. The full dataset is split into groups based on the unique CAN message identifier and each group is processed independently. The data entries in the dataset are arranged as rows and columns with each row representing a single data sample corresponding to a particular timestamp and each column representing a unique feature of the message. The columns consist of the following features: *(i)* timestamp at which the message was logged, *(ii)* message identifier, *(iii)* number of signals in the message, *(iv)* individual signal values (one per column), and *(v)* a single bit representing the label of the message. The label column is 0 for non-anomalous samples and 1 for anomalous samples. The label column is set to 0 for all samples in the training and validation dataset as all the data samples are non-anomalous and collected in a trusted environment. The label column will have a value of 1 for the samples in the test dataset during the attack interval and 0 for the other cases. Moreover, for each signal type, the signal values are scaled between 0 to 1 as there can be high variance in the signal magnitudes. Such high variance in the input data can result in very slow or unstable training. Details related to the models and the training procedure are discussed in the next subsections, while the dataset is discussed in subsection 3.2.1 of this chapter.

3.1.2 PREDICTOR MODEL

We designed predictor and detector models that work in tandem to detect cyber-attacks as anomalies in the in-vehicle network. The predictor model attempts to learn the normal system

behavior via an unsupervised learning approach to predict the next message instance with high accuracy at design time using the normal (non-anomalous) data. During this process, the predictor model learns the underlying distribution of the normal data and relates it to the normal system behavior. This knowledge of the learned distribution is used to make accurate predictions of the next message instances at runtime for normal messages. In the event of a cyber-attack, the message values no longer represent the learned distribution or maintain the same temporal relationships between messages, leading to large deviations between the predictions and the true (observed) messages. These deviation patterns are then learned by the detector model, as discussed in the next subsection, using a non-linear classifier to detect attacks.

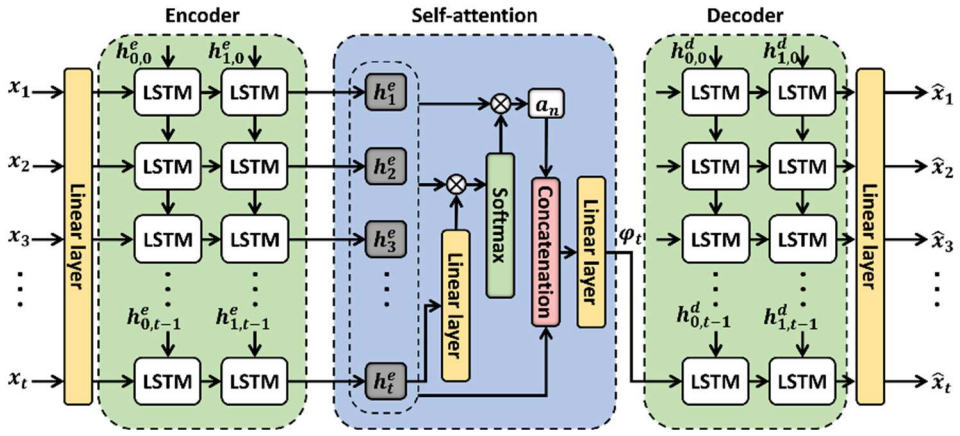


Figure 16 Our proposed predictor model for the *LATTE* anomaly detection framework showing the stacked LSTM encoder – decoder rolled out in time for *t* time steps along with the self-attention mechanism generating context vector for time step *t*. The output at time step *t* ($\hat{x}_t$) is the prediction of the input at time step *t+1* (x_{t+1}).

Our proposed predictor model consists of a stacked LSTM based encoder-decoder architecture with the self-attention mechanism. This is illustrated in Figure 16, The first linear layer in the predictor model takes the time series CAN message data as the input and generates a 128-dimensional embedding for each input. Each input sample consists of *k* features where each

feature represents a particular signal value within that message. The output embedding from the linear layer is passed to the stacked two-layer LSTM encoder to produce a 64-dimension encoder output $\{h_1^e, h_2^e, \ldots, h_t^e\}$. The encoder output is the latent representation of the input time-series signal values that encompass the temporal relationships between messages. The self-attention block generates the context vector (φ_t) by applying the self-attention mechanism to the encoder outputs. The self-attention mechanism begins by applying a linear transformation to the encoder's current hidden state (h_1^e) and multiplies the result with the encoder output. The output from the multiplication is passed through a softmax activation to compute the attention weights. The attention weights represent the importance of each hidden state information from the earlier time steps, at the current time step. The attention weights are scalars multiplied with the encoder outputs to compute the attention applied vector (a_n) which is then combined with the encoder output to compute the input to the decoder (context vector (φ_t)). The context vector along with the previous decoder's hidden state (h_{t-1}^d) is given as input to the stacked two-layer decoder, which produces a 64-dimension output that is passed to the last linear layer to obtain a k dimensional output. This k dimension output represents the signal values of the next message instance.

This predictor model is trained using non-anomalous (normal) data without any labels in an unsupervised manner. To train the model with sequences, we employ a rolling window approach. We consider a window of fixed size length (known as subsequence length) consisting of signal values over time. The window with signal values is called a subsequence and has a subsequence length number of samples of signal values. Our predictor model learns the temporal dependencies that exist between the signal values within the subsequence and uses them to predict the signal values in the next subsequence (i.e., window shifted to the right by one-time

step). The signal values corresponding to the last time step in the output subsequence represent the final prediction, as the model consumes the entire input subsequence to generate them. We compare this last time step in the output subsequence with the actual signal values and compute the prediction error using the mean square error (MSE) loss function. This process is repeated until the end of the training dataset. The predictor model is trained by splitting the dataset into training (80%) and validation (20%) data without shuffling, as shuffling would destroy the existing temporal relationships between messages. During the training process, the model tries to minimize the prediction error in each iteration (a forward and backward pass) by adjusting the weights of the neurons in each layer using backpropagation through time. At the end of each training epoch, the model is validated (forward pass only) using the validation dataset to evaluate the model performance. We employ mini-batches to speed up the training process and use an early stopping mechanism to avoid overfitting. The details related to the non-anomalous dataset and the hyperparameters selected for the model are presented in subsection 3.2.1 of this chapter.

3.1.3 DETECTOR MODEL

After training the predictor model, we train a separate classifier (detector model) that utilizes the information from the predictor to detect attacks. The anomaly detection problem can be treated as a binary classification problem as we are mainly interested in distinguishing between normal and anomalous messages. In general, as the in-vehicle network data recordings can grow in size very rapidly, labeling this data can get very expensive. Additionally, due to the nature of the frequency of attack scenarios, the number of attack samples would be quite small compared to normal samples even when the dataset is labeled. This results in having a highly imbalanced dataset that would result in poor performance when trained with a traditional binary classifier in

a supervised learning setting. However, a popular non-linear classifier known as a support vector machine (SVM) can be altered to make it work with unbalanced datasets where there is only one class. Hence, in this work, we use a one class support vector machine (OCSVM) to classify the messages as anomalous or normal. The OCSVM learns the distribution of the training dataset by constructing the smallest hypersphere that contains the training data at design time and identifies any sample outside the hypersphere as an anomaly at runtime.

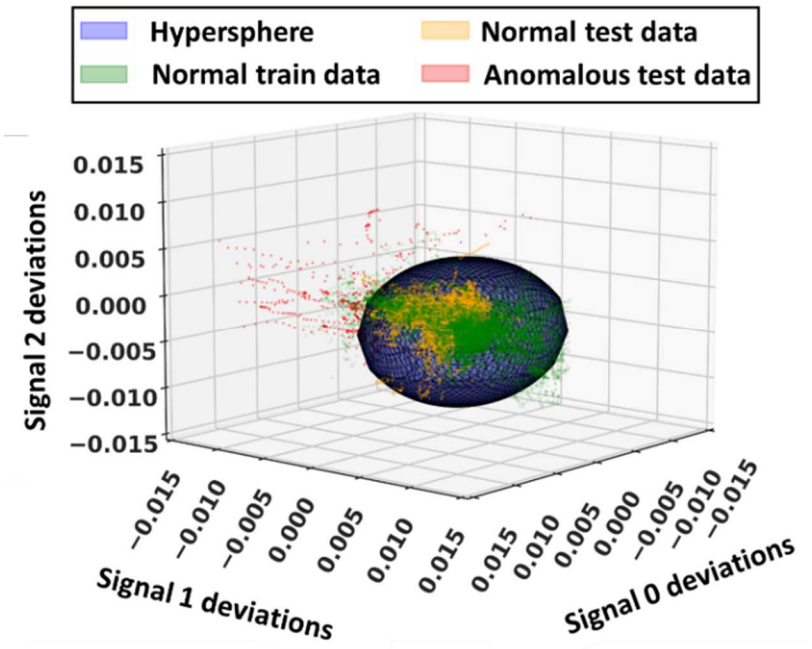


Figure 17 OCSVM decision boundary shown in the blue sphere with the green dots showing the normal samples from training data, and yellow and red dots showing the normal and anomalous samples respectively from test data.

We train an OCSVM by using the output from the previously trained predictor model. We begin by giving the previously used normal training dataset as the input to the predictor to generate the predictions. We then compute the deviations (prediction errors) for all the training data and pass it as input to the OCSVM. The OCSVM tries to generate the smallest hypersphere

that can fit most of the deviation points and uses it at runtime to detect anomalies. Figure 17 shows an example of a hypersphere generated by training an OCSVM for a message with three signals. Each axis in the figure represents the relevant signal deviation and the dark blue sphere represents the decision boundary. It can be observed that almost the entirety of training data (shown via green dots) is confined to within the blue sphere.

In our work, the deviation of a message is represented as a vector where each element of the vector corresponds to the difference between the true and predicted signal value. Therefore, for a message m with k_m number of signals, the deviation vector ($\Delta_{m,t}$) computed at time step t is given by equation (15).

$$\Delta_{\boldsymbol{m,t}} = (\widehat{\boldsymbol{S}}_{\boldsymbol{i,t}} - \boldsymbol{S}_{\boldsymbol{i,t}}) \in \mathbb{R}^{\mathbf{2}}, \ \forall \boldsymbol{i} \in [\mathbf{1}, \boldsymbol{k_m}] \qquad (15)$$

where $\hat{S}_{i,t}$ represent the prediction of the next true i^{th} signal value ($S_{i,t+1}$) made at time step t. We also experimented with other deviation measures that are given by equations (16), (17), and (18).

$$\Delta_{\boldsymbol{m,t}}^{\boldsymbol{sum}} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{k_m}} |\Delta_{\boldsymbol{m,t}}| \ , \forall \boldsymbol{i} \in [\mathbf{1}, \boldsymbol{k_m}] \qquad (16)$$

$$\Delta_{\boldsymbol{m,t}}^{\boldsymbol{avg}} = \frac{\mathbf{1}}{\boldsymbol{k_m}} \sum_{\boldsymbol{i=1}}^{\boldsymbol{k_m}} |\Delta_{\boldsymbol{m,t}}| \ , \forall \boldsymbol{i} \in [\mathbf{1}, \boldsymbol{k_m}] \qquad (17)$$

$$\Delta_{\boldsymbol{m,t}}^{\boldsymbol{max}} = \boldsymbol{max}(|\Delta_{\boldsymbol{m,t}}|, \forall \boldsymbol{i} \in [\mathbf{1}, \boldsymbol{k_m}] \qquad (18)$$

Moreover, there can be situations where some of the signal deviations in a message can be positive while others are negative. This could potentially result in making the sum or mean of signal deviations zero or near zero, falsely representing no deviation or very small deviation. To avoid these situations, we use the absolute signal deviations to compute the deviations for the variants. Note: Unlike equation (15) that uses a vector of k dimensions to represent the message deviation, equations (16), (17), and (18) reduce the vector to a single value using different reduction operations. We explored these deviation measures to determine the best one, as discussed in subsection 3.2.2 in this chapter.

In summary, our predictor model predicts the normal samples with very small deviations and anomalous samples with high deviations. The OCSVM takes this predictor property into account when constructing the hypersphere. In Figure 17, the yellow dots and red dots represent the normal and anomalous samples respectively in the test dataset. It can be observed that when the test data with anomalies is given as input to the OCSVM, it generally correctly classifies the yellow samples within the hypersphere and red samples outside the hypersphere. Thus, both predictor and detector models work collectively to detect attacks as anomalies. The details related to the testing process are described in the next subsection.

3.1.4 MODEL TESTING

In the deployment/testing step, we present a test dataset consisting of anomalous samples representing multiple attacks (outlined in chapter 2 subsection 2.1.4) along with the normal samples to the *LATTE* framework. The normal messages have a label value of 0 and the attack messages have a label value of 1. During this step, each sample (signal values in a message) is first sent to the predictor model to predict the signal values of the next message instance, and the deviation is computed based on the true message data. This deviation vector is passed to the OCSVM detector model, to compute the position of the deviation vector in the k-dimensional space, where *k* represents the number of signals in the message. The message is marked as non-anomalous when the point corresponding to the deviation vector falls completely inside the learned hypersphere. Otherwise, the message is marked as anomalous and an anomaly alert is raised. This can be used to invoke an appropriate remedial action to suppress further actions from the attacker. However, the design of remedial actions and response mechanisms falls outside the

scope of our work. The performance evaluation of our proposed *LATTE* framework under various attack scenarios is presented in detail in subsections 3.2.2 and 3.2.3 of this chapter.

3.2 EXPERIMENTAL STUDIES

3.2.1 EXPERIMENTAL SETUP

To evaluate the effectiveness of our proposed *LATTE* framework, we first explored five variants of the same framework with different deviation criteria: *LATTE-ST*, *LATTE-Diff*, *LATTE-Sum*, *LATTE-Avg*, and *LATTE-Max. LATTE-ST* uses our proposed predictor model with a static threshold (ST) value to determine whether a given message is anomalous or normal based on the deviation. The other four variants use the same predictor model but different detection criteria for computing the deviations for OCSVM. *LATTE-Diff* uses the difference in signal values (equation (15)); *LATTE-Sum* and *LATTE-Avg* use a sum and mean of absolute signal deviations respectively (equations (16), and (17)); and *LATTE-Max* uses the maximum absolute signal deviation (equation (18)), as the input to the detector model.

Subsequently, we compare the best variant of our framework with three prior works: Bitwise Message Predictor (BWMP [52]), Hierarchical Attention-based Anomaly Detection (HAbAD [55]), and a variant of [55] called Stacked HAbAD (S-HAbAD [55]). BWMP [52] trains an LSTM based neural network that aims to predict the next 64 bits of a CAN message by minimizing the bitwise prediction error using a binary cross-entropy loss function. At runtime, BWMP [52] uses the prediction loss as a measure to detect anomalies. HAbAD [55] uses an LSTM based autoencoder model with hierarchical attention. The HAbAD model attempts to recreate the input message sequences at the output and aims to minimize reconstruction loss.

Additionally, HAbAD uses supervised learning in the second step to model a detector using the combination of a non-parametric kernel density estimator (KDE) and k-nearest neighbors (KNN) algorithm to detect cyber-attacks at runtime. Lastly, S-HAbAD is a variant of HAbAD that uses stacked LSTMs as autoencoders and uses the same detection logic used by the HAbAD. The S-HAbAD variant is compared to show the effectiveness of using stacked LSTM layers. The results of all experiments are discussed in detail in the next subsections 3.2.2 and 3.2.3.

We conducted all experiments using an open-source CAN message dataset developed by ETAS and Robert Bosch GmbH [50]. The dataset consists of CAN message data for different message IDs consisting of various fields such as timestamps, message ID, and individual signal values. Additionally, the dataset consists of a training dataset with only normal data and a labeled test dataset with multiple attacks (as discussed in chapter 2 subsection 2.1.4). It is important to note that we do not use any labeled data during the training or validation of our models and learn the normal system behavior in an unsupervised manner. The labeled data is given to the models only during the testing phase and used to compute performance metrics.

We used PyTorch 1.5 to implement all of the machine learning models including *LATTE* and its variants, and the models from the comparison works. Our proposed predictor model is trained with 80% of the available normal data and the remaining 20% is used for validation. The training phase is repeated for 500 epochs with an early stopping mechanism that monitors the validation loss after the end of each epoch and stops if there is no improvement after 10 (patience) epochs. We used the ADAM optimizer with mean squared error (MSE) as the loss function. Additionally, we employed a rolling window approach (discussed in subsection 3.1.2) with a subsequence length of 32 time steps, a batch size of 256, and a starting learning rate of 0.0001. We used the scikit-learn package to implement the OCSVM in the detector model. We

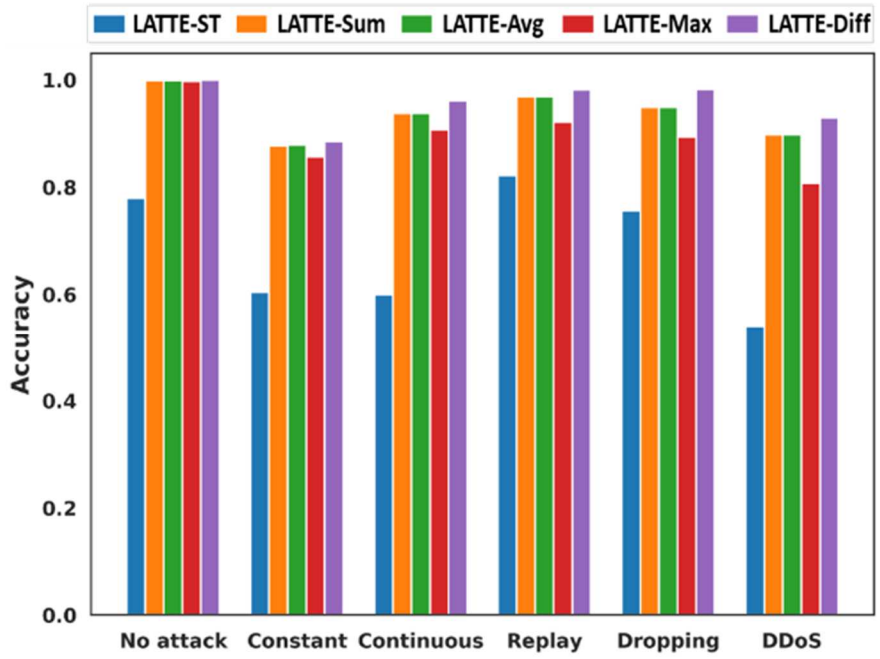
used a radial basis function (RBF) kernel with a kernel coefficient (gamma) equal to the reciprocal of the number of features (i.e., number of signals in the message). Moreover, to speed up OCSVM training, we set the kernel cache size to 400 MB and enabled the shrinking technique to avoid solving redundant optimizations. All the simulations are run on an AMD Ryzen 9 3900X server with an Nvidia GeForce RTX 2080Ti GPU.

Before looking at the experimental results for various performance metrics, it is important to understand some key definitions in the context of anomaly detection. We define a true positive as the scenario when an actual attack is detected as an anomaly by the anomaly detection system and a true negative as the situation where an actual normal message is detected as normal. Additionally, a false positive would be a false alarm where a normal message is incorrectly classified as an anomaly and a false negative would occur when an anomalous message is incorrectly classified as normal. Using the above definitions, we evaluate our proposed framework using four different metrics: *(i) Detection accuracy*: a measure of the anomaly detection system's ability to detect anomalies correctly, *(ii) False positive rate*: i.e., false alarm rate, *(iii) F1 score*: a harmonic mean of precision and recall; we use the F1-score instead of individual precision and recall values as it captures the combined effect of both precision and recall metrics, and *(iv) receiver operating characteristic (ROC) curve with area under the curve (AUC)*: a popular measure of classifier performance. A highly efficient anomaly detection system has high detection accuracy, F1 score, and ROC-AUC while having a very low false-positive rate.

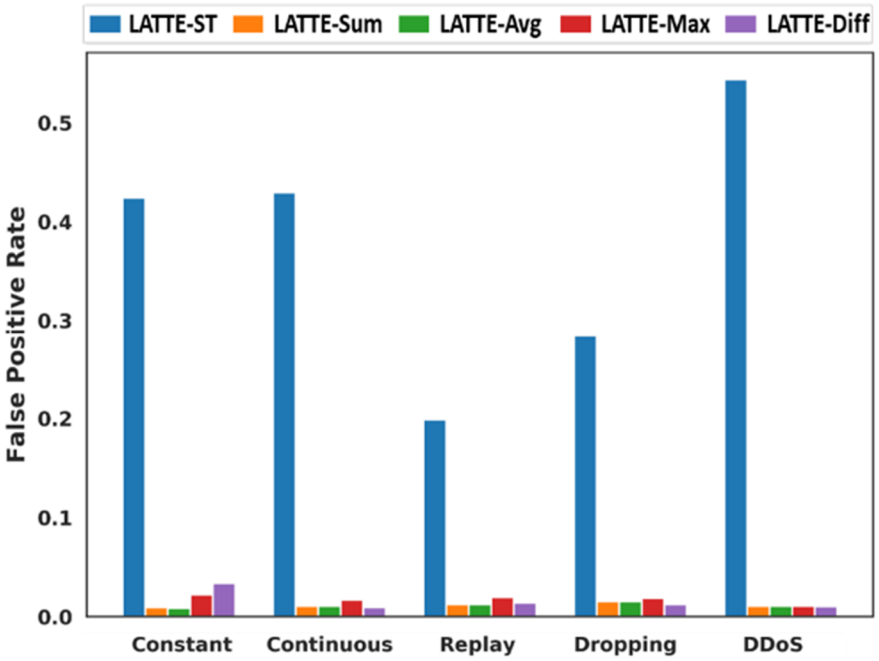
3.2.2 COMPARISON OF LATTE VARIANTS

In this subsection, we present the comparison results of the five variants *LATTE-ST*, *LATTE-*

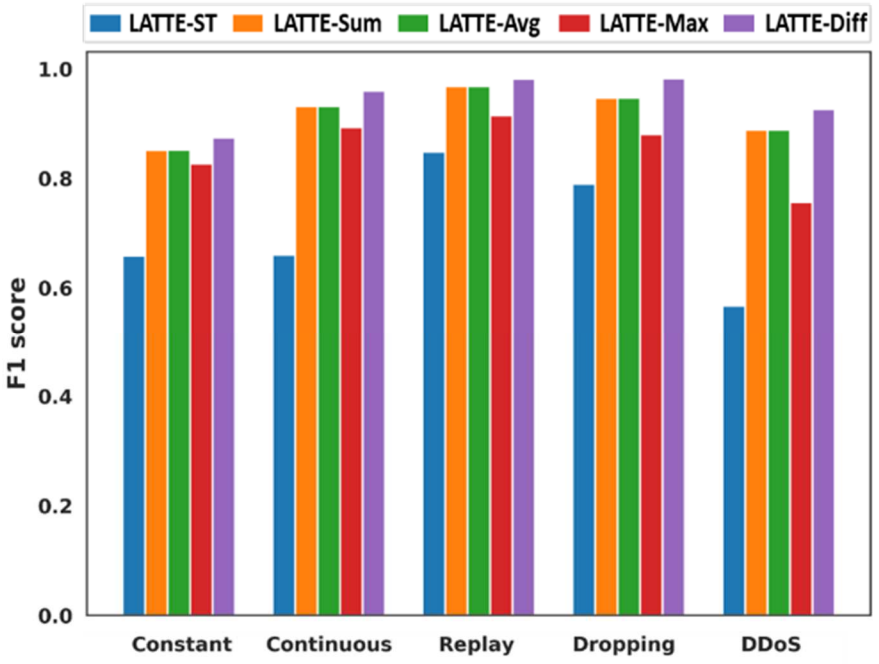
Sum, *LATTE-Avg*, *LATTE-Max,* and *LATTE-Diff*. All the variants of *LATTE* use the trained predictor model (discussed in subsection 3.1.3) to make the predictions and use OCSVM as a detector except in the case of *LATTE-ST*, which uses a fixed threshold scheme introduced in [19] to predict the given message as normal or anomalous. The main purpose of this experiment is to analyze the impact of using a non-linear classifier such as OCSVM on the model performance instead of a simple static threshold scheme (*LATTE-ST*). Additionally, with the last four variants, we aim to study the effect of different deviation criteria on the OCSVM detection performance. The deviations for any given message in *LATTE-Diff* ($\Delta_{m,t}$), *LATTE-Sum* ($\Delta_{m,t}^{sum}$), *LATTE-Avg* ($\Delta_{m,t}^{avg}$) and *LATTE-Max* ($\Delta_{m,t}^{max}$) are computed using the equations (15), (16), (17), and (18) respectively.



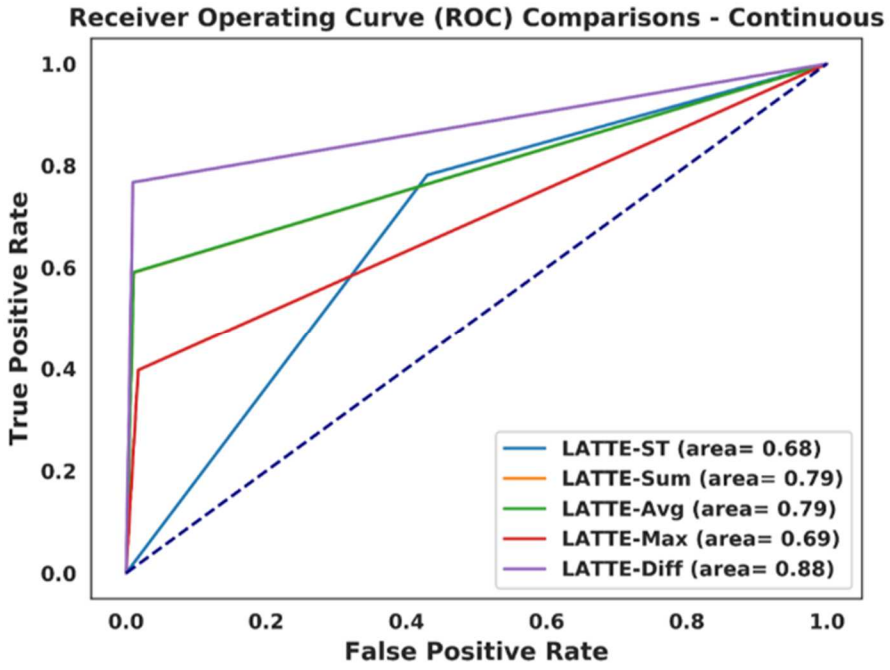
(a)


LATTE-ST
LATTE-Sum
LATTE-Avg
LATTE-Max
LATTE-Diff
False Positive Rate
0.0
0.1
0.2
0.3
0.4
0.5
Constant
Continuous
Replay
Dropping
DDoS

(b)


LATTE-ST
LATTE-Sum
LATTE-Avg
LATTE-Max
LATTE-Diff
F1 score
0.0
0.2
0.4
0.6
0.8
1.0
Constant
Continuous
Replay
Dropping
DDoS

(c)

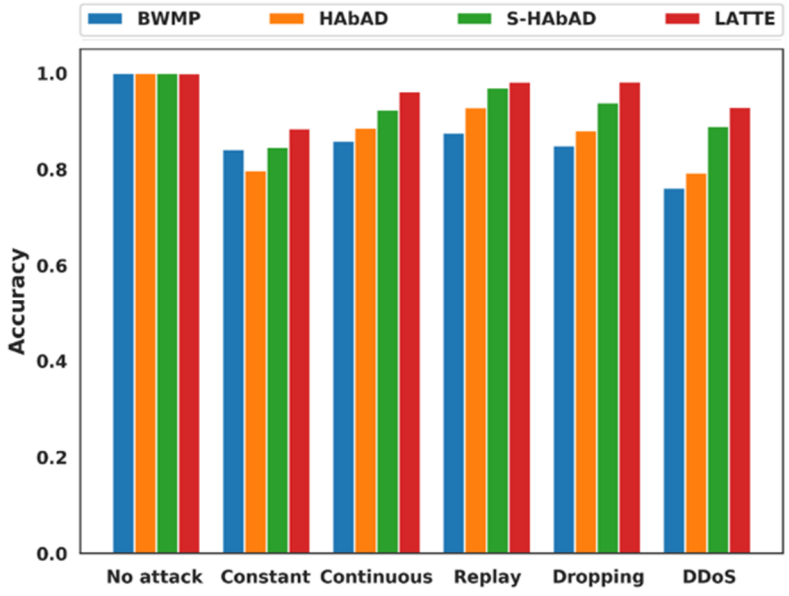


(d)

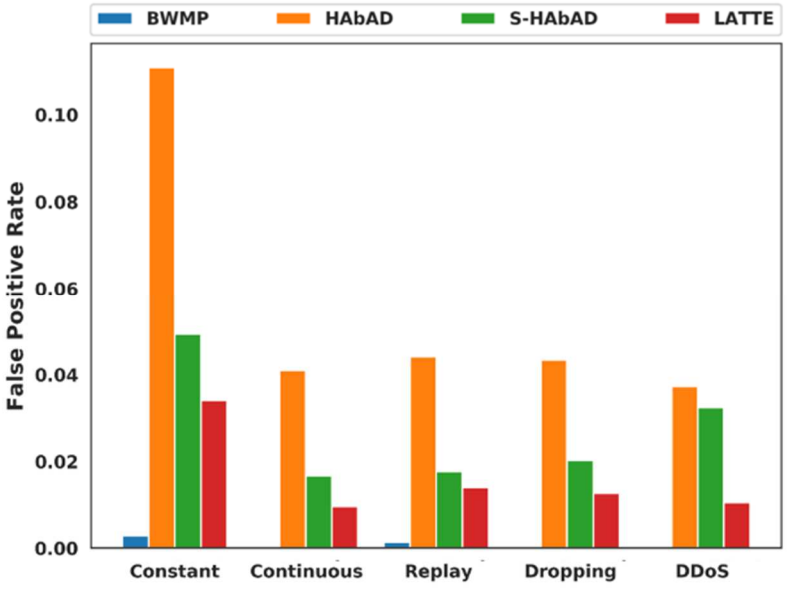
Figure 18 Comparison of (a) detection accuracy, (b) false-positive rates, (c) F1 score of *LATTE* variants under different attack scenarios, and (d) ROC curve with AUC for continuous attack.

3.2.3 COMPARISON OF LATTE WITH PRIOR WORKS

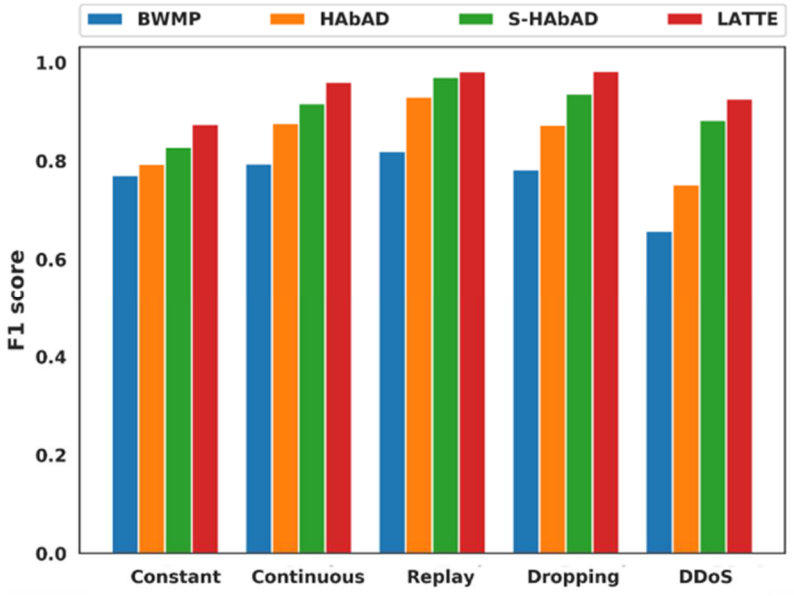
We compared our *LATTE* framework with BWMP [52], HAbAD [55], and a variant of HAbAD called S-HAbAD [55]. Figure 19 (a)-(c) show the detection accuracy, false-positive rate, and F1 score respectively for these frameworks under different attack scenarios. It can be observed that *LATTE* outperforms all the prior works in terms of detection accuracy, false-positive rate, and F1 score. This is due to three factors. Firstly, the stacked LSTM encoder-decoder structure provides adequate depth to the model to learn complex time-series patterns. This can be seen when comparing HAbAD with S-HAbAD, as the latter differs only in terms of stacked LSTM layers in comparison to the former.


BWMP
HAbAD
S-HAbAD
LATTE
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
No attack
Constant
Continuous
Replay
Dropping
DDoS

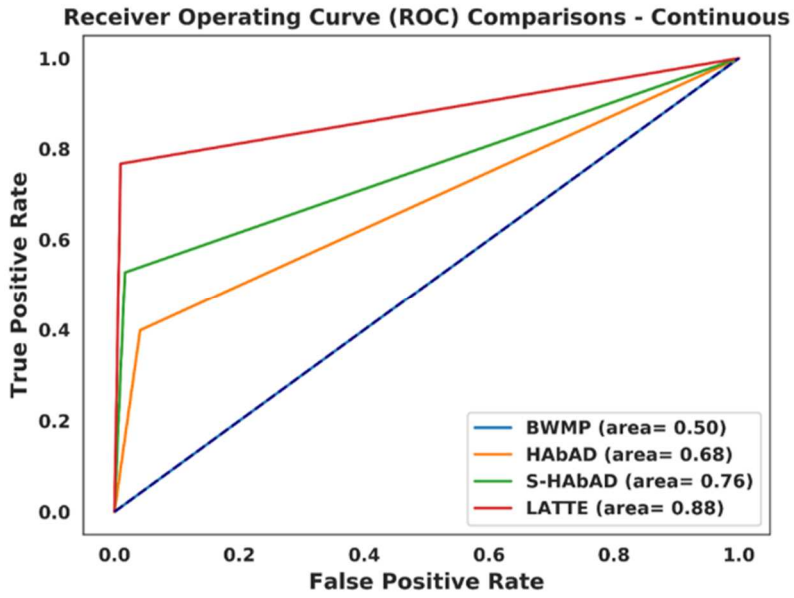
(a)


BWMP
HAbAD
S-HAbAD
LATTE
False Positive Rate
0.10
0.08
0.06
0.04
0.02
0.00
Constant
Continuous
Replay
Dropping
DDoS

(b)



(c)



(d)

Figure 19 Comparison of (a) accuracy, (b) false-positive rates, (c) F1 score of *LATTE* and the comparison works under different attack scenarios, and (c) ROC curve with AUC for continuous attack.

Second, the self-attention mechanism helps *LATTE* in learning message sequences that have very long-term dependencies. Lastly, the use of powerful OCSVMs as non-linear classifiers helps in constructing a highly efficient classifier. These factors together resulted in the superior performance of *LATTE* compared to all the comparison works. On average across all attacks, *LATTE* was able to achieve up to 66.7% improvement in accuracy, 120.2% improvement in F1 score, 76% improvement in AUC.

3.2.4 LATTE OVERHEAD ANALYSIS

In this subsection, we present an overhead analysis of our *LATTE* framework. We quantify the overhead of our *LATTE* framework and the comparison works using memory footprint, the number of model parameters, and the inference time metrics. We profiled each framework on a dual core ARM Cortex-A57 CPU on an Nvidia Jetson Tx2 board, which has similar specifications to that of a real-world ECU. We repeated the inference time experiment 10 times and computed the average inference time.

Table 1 Memory, model and runtime overhead of *LATTE* in comparison with BWMP **[52]**, HAbAD **[55]**, and S-HAbAD **[55]**

Framework	**Memory footprint (KB)**	**Number of Model parameters ($x10^3$)**	**Average inference time (µs)**
BWMP [52]	13,147	3435	644.76
HAbAD [55]	4558	64	685.05
S-HAbAD [55]	5600	325	976.65
LATTE	1439	331	193.90

From Table 1, we can observe that our *LATTE* framework has minimal overhead compared to both attention-based prior works (HAbAD and S-HAbAD) and the non-attention based work (BWMP). The high runtime and memory overhead in HAbAD and S-HAbAD is associated with

the use of KNNs. KNN does not generalize the data in advance, but rather scans through each training data sample to make a prediction. This makes it very slow and consumes high memory overhead (due to the requirement of having training data available at runtime).

3.3 CONCLUSION

In this chapter, we proposed a novel stacked LSTM with a self-attention framework called *LATTE* that learns the normal system behavior by learning to predict the next message instance under normal operating conditions. We presented a one class support vector (OCSVM) based detector model to detect cyber-attacks by monitoring the message deviations from the normal behavior. We presented a detailed analysis by comparing our proposed model with multiple variants of our model and the best-known prior works in this area. Our *LATTE* framework surpasses all the variants and the best-known prior works under different attack scenarios while having a relatively low memory and runtime overhead.

4. TENET: TEMPORAL CNN WITH ATTENTION FOR ANOMALY DETECTION IN AUTOMOTIVE CYBER-PHYSICAL SYSTEMS

In this chapter, we propose an anomaly detection framework called *TENET* that uses a novel TCNA (temporal CNN with attention) model to monitor the in-vehicle network and detect various real-world attacks. During learning, our proposed TCNA network is able to efficiently capture very long-term dependencies between in-vehicle network messages in an unsupervised manner. In the evaluation phase, we employ our trained TCNA network to reliably detect multiple real-world attack scenarios with a low memory overhead. *TENET* attempts to increase the detection accuracy, receiver operating characteristic curve with area under the curve (ROC-AUC), false positive rate (FPR), false negative rate (FNR) and mathews correlation coefficient (MCC) metrics (defined in subsection 4.2.1) with minimal overhead. Our novel contributions in this work can be summarized as follows:

- We present a novel temporal convolutional neural attention (TCNA) network architecture to learn very-long term temporal dependencies between messages in the in-vehicle network;
- We introduce a metric called divergence score to quantify the deviation from expected behavior;
- We present a decision tree based classifier to detect various attacks at runtime using the proposed divergence score metric;
- We present a comprehensive analysis on the selection of key parameters to effectively capture very-long term dependencies and analyzed different classifiers to detect anomalies;

- We compare our *TENET* framework with various state-of-the-art IDS frameworks to demonstrate its effectiveness.

4.1 TENET FRAMEWORK: OVERVIEW

Our proposed *TENET* framework consists of three phases: *(i)* data collection and preprocessing, *(ii)* learning, and *(iii)* evaluation. The first phase involves collecting in-vehicle network data from a trusted vehicle and preprocessing the collected data. In the learning phase (*offline*), the preprocessed data is used to train a novel Temporal Convolutional Neural Attention (TCNA) network in an unsupervised manner to learn the normal behavior of the system. Lastly, in the evaluation phase (*online*), the trained TCNA network is used to calculate a divergence score (DS), which is then used by a decision tree-based classifier to detect attacks. The overview of our proposed *TENET* framework is shown in Figure 20. The following subsections describe the three phases of the framework.

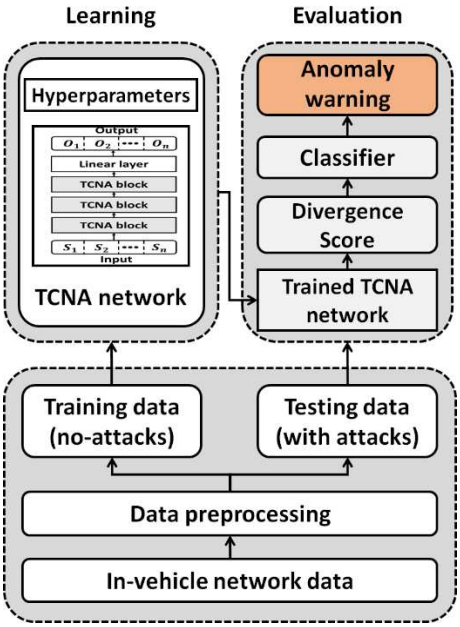


Figure 20 Different phases of our proposed *TENET* framework

4.1.1 DATA COLLECTION

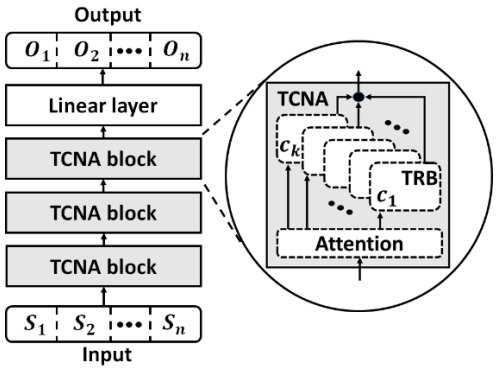
This first phase of the *TENET* framework involves collecting in-vehicle network data from a trusted vehicle. During the data collection step, it is crucial to ensure that the in-vehicle network and the associated ECUs are trusted and free from any malware. Otherwise, the model learns the incorrect representation of the normal operation of the in-vehicle network. Moreover, it is important to collect data from a variety of normal operating conditions, and for different durations. This provides a rich collection of the in-vehicle network data that captures various normal operating conditions of the vehicle, which can then be used to train better models. One of the ways to collect the in-vehicle network data is by using the OBD-II port which gives access to most of the commonly used messages in addition to the unified diagnostic service (UDS) messages. However, we recommended splicing into the CAN network and directly logging the messages using a standard logger such as Vector GL 1000 [59]. This provides full access to the CAN network and allows us to record any CAN message traversing the network.

After the data collection, the data is prepared for pre-processing to facilitate easy and efficient training of the models. Every CAN message has a unique identifier and each message in the dataset is grouped by the unique identifier and processed independently. The processed entries are arranged in a tabular format where each row represents a single data sample and each column represents various unique attributes of the message. Each message has the following attributes (columns): *(i)* unique timestamp corresponding to the log time of the message, *(ii)* message identifier, *(iii)* number of signals in the message, *(iv)* individual signal values in the message (which together constitute the CAN message payload), and *(v)* label of the message (‘0’ for no-attack and ‘1’ for attack). The learning phase and evaluation phases of our *TENET*

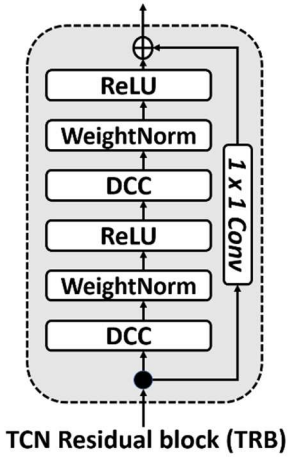
framework involve different datasets. Therefore, we split the collected data into training and testing data. Labels of all collected samples in the training dataset are set to 0 to represent no-attack data. The test data will have a label value of 1 for attack samples and a label value of 0 for no-attack samples. Furthermore, the original training data is split into training (85%) and validation (15%) data for training and evaluating the trained model respectively. Due to the possibility of high variance in CAN signal values, all signal values of each signal type are scaled between 0 and 1. In this work, we primarily focus on monitoring the payload of the CAN message i.e., signal values within each message. This is because an attacker needs to modify the bits in the payload to launch most attacks. As our proposed *TENET* framework relies primarily on monitoring the message payload it can also be adapted to other in-vehicle network protocols such as FlexRay, LIN, CAN-FD, etc. with minimal changes. More details about the training procedure and the model architecture are discussed next.

4.1.2 TENET LEARNING PHASE

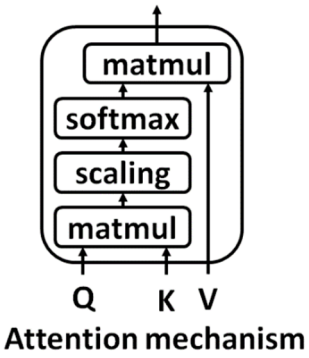
In this subsection, we present our proposed novel TCNA network architecture and the training procedure we employed to train the TCNA network. *TENET* uses the TCNA network to learn the normal system behavior of the in-vehicle network in an unsupervised manner. The proposed TCNA model takes the series of signal values in a message as the input and tries to predict the signal values of the next message instance. The TCNA network achieves this by trying to learn the underlying probability distribution of the normal data while attempting to minimize the prediction error during training.



(a)



(b)



(c)

Figure 21 (a) TCNA network architecture with the internal structure of the TCNA block, (b) TCN residual block showing the various layers of transformation and, (c) the attention mechanism.

The proposed TCNA network consists of three stacked TCNA blocks and a final linear layer before the output as shown in Figure 21 (a). A TCNA block consists of an attention block and a TCN residual block (TRB). The TRB is adopted from [26] and employs a residual architecture containing two DCC layers, two weight normalization layers, and two ReLU nonlinearity layers stacked together as shown in Figure 21 (b). This residual architecture helps to efficiently backpropagate the gradients and encourages the reuse of learned features. The input to the first TCNA block is a time series of CAN message data with *n* signal values as features. This partial sequence from the complete time-series dataset, which is given as the input to the model every time, is called a subsequence. Since padding zeros to the input subsequence will distort the sequential nature of time series data, we avoid zero-padding to the input time-series by computing the length of the subsequence using (2).

$$\boldsymbol{R = (k - 1) * 2^{l}} \quad (19)$$

where *R* is the subsequence length, *k* is the kernel size, and *l* is the number of DCC layers in the networks. The first TCNA block does not contain an attention block, and the inputs are directly fed to the TRB as shown in Figure 21 (a), where $\{c_1, c_2, \dots, c_k\}$ represents multiple channels of the TRB and *k* is the number of channels of TRB inputs. The first DCC layer inside the TRB processes each feature of the input sequence as separate channels. A 1-D dilated causal convolution operation is performed using a kernel of size two and the number of filters is three times the input features (*n*) in each DCC layer. The input and output dimensions are the same except for the first TRB. The output from the DCC layer is weight normalized for faster convergence and avoids explosion of weight values. The weight normalized outputs are passed through a non-linear ReLU activation function. This process is repeated one more time inside the

TRB. A convolution layer with a filter size of 1x1 is added to make the dimensions of the outputs from the last ReLU activation and the input of the TRB consistent with each other. Each DCC layer in the TRB tries to learn the temporal relationship between messages by applying various filters to its inputs and updating the filter weight values accordingly.

The output feature maps of the TRB are given as the input to the attention block, shown in Figure 21 (c). The attention block repeats its inputs to obtain the query *(Q)*, key *(K),* and value *(V)* vectors. A scalar-dot product is performed between Q and transpose of key (K^T) to calculate the similarities between each Q and K vectors. The resultant dot product is scaled by a factor of $1/\sqrt{d_k}$ and passed through a softmax layer to calculate the attention weights, as shown in equation (20).

$$\boldsymbol{Attention(Q,K,V) = softmax(}\boldsymbol{QK^T}/\sqrt{\boldsymbol{d_k}}\boldsymbol{)} \qquad (20)$$

where the term d_k represents the dimension of the K vector. The attention weights represent the importance of each feature map of the previous DCC layer. The attention weights are then scalar multiplied with V to produce the output of the attention block. Thus, the attention block uses a self-attention mechanism to improve the quality of feature maps that will be received by the subsequent TRBs. Similarly, the input sequence flows through the TCNA network and is fed to the final linear layer which produces an output of n dimensions. The n-dimensional output represents the predicted signal values of each dimension.

The TCNA network model is trained in an unsupervised manner using the training data that has no-attack samples and label information. We employ a rolling window approach to train the TCNA network. Each window consists of signal values corresponding to the current subsequence. Our TCNA network learns the temporal dependencies between messages inside a

subsequence and tries to predict the signal values of the subsequence that is shifted by one time step to the right. We employ a mean squared error (MSE) loss function to compute the prediction error by comparing the signal values of the last time step in the predicted subsequence with the last time step of the input subsequence. The error is propagated backward to update the weights for the filters. This process is repeated for each subsequence until the end of the training data, which constitutes one epoch. We train the model for multiple epochs and employ a mini-batch training approach to speed up the training. At the end of each epoch, the model is evaluated using the unseen validation data. Additionally, we employed an early stopping mechanism to prevent the model from overfitting. The details related to the model hyperparameters are discussed later in subsection 4.2.1.

4.1.3 TENET EVALUATION PHASE

We use the trained TCNA network in conjunction with a detection classifier to efficiently detect in-vehicle network attacks at runtime. As we are mainly interested in distinguishing between attack and no-attack samples, we transform the detection problem into a binary classification problem. The high frequency of messages in the in-vehicle network requires a detection classifier that is lightweight and can classify messages quickly with high detection accuracy and minimal overhead. Hence, we use a categorical variable decision tree based classifier to detect between normal and attack samples due to their simpler nature, speed, and precise classification capabilities.

A decision tree starts with a single node (root node), which branches into possible outcomes. Each of those outcomes leads to additional nodes called branch nodes. Each branch node branches off into other possibilities and ends in a leaf node giving it a treelike structure.

During training, the decision creates the tree structure by determining the set of rules in each branch node based on its input. During testing, the decision tree takes the input and traverses the tree structure until it reaches a leaf node, which classifies the input as either a no-attack or attack sample.

The evaluation phase begins by splitting the test data with attacks into two parts: *(i)* calibration data, and *(ii)* evaluation data. In the first part, only the calibration data is fed to the trained TCNA network to generate the predicted sequences. We then compute the divergence score (DS) for each signal in every message using equation (21).

$$\boldsymbol{DS_i^m(t) = \left(\widehat{S}_i^m(t) - S_i^m(t+1)\right) \forall\, i \in [1, N_m], m \in [1, M]} \quad (21)$$

where m represents the m^{th} message sample and M represents the total number of message samples, i represents the i^{th} signal of the m^{th} message sample and N_m represents the total number of signals in the m^{th} message, t represents the current time step, $S_i^m(t)$ represents the i^{th} predicted signal value of the m^{th} message at time step t, and $S_i^m(t+1)$ represents the true i^{th} signal value of the m^{th} message sample at time step $t + 1$.

The divergence score is higher during an attack as the TCNA model is trained on the no-attack data and fails to predict the signal values correctly in the event of an attack. This sensitive nature of the DS to attacks makes it a good candidate for the input to our detection classifier. Moreover, the group of signal level DS for each message sample is stacked together to obtain the DS vector. Thus, we train the decision tree classifier using the DS vector as input to learn the distribution of both no-attack samples and attack samples. We use the unseen evaluation data (that has both attack and no-attack samples) to evaluate the performance of *TENET*.

4.2 EXPERIMENTS

4.2.1 EXPERIMENTAL SETUP

To evaluate the effectiveness of the *TENET* framework, we conducted various experiments. We compare *TENET* against three state-of-the-art prior works on automotive IDS: RN [56], INDRA [54], HABAD [55], and LATTE [16]. Together, these approaches reflect a wide range of sequence modeling architectures. RN [56] uses vanilla RNNs to increase the dimensionality of input signal values and reconstruct the input signal at the output by minimizing MSE. The trained RN model scans continuously for large reconstruction errors at runtime to detect anomalies over in-vehicle networks. HABAD [55] uses an LSTM based autoencoder model with attention to detect anomalies in real-time CPS. This model reduces the MSE reconstruction loss by trying to replicate the input message signal at the output. HABAD uses a supervised learning detector that combines a kernel density estimator (KDE) and k-nearest neighbors (KNN) algorithm to detect anomalies. LATTE [16] uses a LSTM based predictor model with a novel self-attention mechanism to predict the next time step message signal value in an unsupervised fashion. A one class support vector machine (OCSVM) based detector works together with the LSTM self-attention predictor model to detect different cyber-attacks at runtime. Lastly, INDRA [54] uses a GRU-based autoencoder that reconstructs input sequences at the output by reducing the MSE reconstruction loss. At runtime, INDRA utilizes a pre-computed static threshold to flag

anomalous messages. The comparisons of TENET with the above-mentioned IDS are presented in subsections 4.2.2 and 4.2.3.

We adopted an open-source CAN message dataset developed by ETAS and Robert Bosch GmbH [50] to train our model, and the comparison works. The dataset consists of multiple CAN messages with a different number of signals that were modeled after real-world vehicular network information. Moreover, the dataset has a distinct training set that has normal CAN messages and a labeled testing dataset for different types of attacks. For training and validation, we used the training dataset from [50] without any attack scenarios in an unsupervised manner. We tested our proposed *TENET* framework, and all comparison works by modeling various real-world attacks (discussed in chapter 2 subsection 2.1.4) using the test dataset in [50]. Note that *TENET* can be easily adapted to other in-vehicle network protocols such as Flexray and Ethernet, as it relies only on the message payload information. However, the lack of any openly available datasets for these protocols prevents us from showing results on them.

We used PyTorch 1.8 to model and train various machine learning models including *TENET*, and the comparison works. Our framework uses 85% of data for training and the remaining 15% for validation. We trained *TENET* for 200 epochs with an early stopping mechanism that constantly monitors the validation loss after each epoch. If no improvement in validation loss is observed in the past 10 (patience) epochs, training is terminated. We used MSE to compute the prediction error and the ADAM optimizer with a learning rate of 1e-4. We employed a rolling window approach (discussed in subsection 4.1.2) with a batch size of 256 and a subsequence length of 64. We used scikit-learn to implement the decision tree classifier, with the *gini*-criterion, and best splitter to detect anomalies based on the divergence score.

Before discussing the results, we define performance metrics in the context of anomaly detection. We classify a message as a true positive *(TP)* only if the model detects a true attack as an anomaly, and a true negative *(TN)* is when a normal message is detected as a no-attack message. When the model detects a normal message as an anomalous message it is defined as a false positive *(FP)*, whereas an actual anomalous message which is not detected is a false negative *(FN)*. Using these definitions, we evaluate our framework based on four different performance metrics:

- *Detection Accuracy*: which quantifies the ability of the IDS to detect an anomaly correctly, as defined in equation (22).

$$\boldsymbol{Detection\ accuracy} = \frac{(\boldsymbol{TP}+\boldsymbol{TN})}{(\boldsymbol{TP}+\boldsymbol{TN}+\boldsymbol{FP}+\boldsymbol{FN})} \qquad (22)$$

- *Receiver Operating Characteristic (ROC) curve with area under the curve (AUC)*: which measures the classifier's performance as the area under the curve in A plot between the true positive rate (TPR) and false positive rate (FPR) as shown in equation (23):

$$\boldsymbol{TPR} = \frac{\boldsymbol{TP}}{(\boldsymbol{TP}+\boldsymbol{FN})} \qquad \boldsymbol{FPR} = \frac{\boldsymbol{FP}}{(\boldsymbol{FP}+\boldsymbol{TN})} \qquad (23)$$

- *False Negative Rate (FNR)*: which quantifies the probability that a TP will be missed by the model (a lower value is better). It is calculated as shown in equation (24):

$$\boldsymbol{FNR} = \frac{\boldsymbol{FN}}{(\boldsymbol{TP}+\boldsymbol{FN})} \qquad (24)$$

- *False Positive Rate (FPR)*: which quantifies the probability that a TN will be missed by the model (a lower value is better). It is calculated as shown in equation (25):

$$\boldsymbol{FPR} = \frac{\boldsymbol{FP}}{(\boldsymbol{TN}+\boldsymbol{FP})} \qquad (25)$$

- *Mathews Correlation Coefficient (MCC)*: which provides an accurate evaluation of the model performance while working with imbalanced datasets, as defined as equation (26):

$$\boldsymbol{MCC} = \frac{((\boldsymbol{TP} * \boldsymbol{TN}) - (\boldsymbol{FP} * \boldsymbol{FN}))}{\sqrt{((\boldsymbol{TP}+\boldsymbol{FP})(\boldsymbol{TP}+\boldsymbol{FN})(\boldsymbol{TN}+\boldsymbol{FP})(\boldsymbol{TN}+\boldsymbol{FN}))}} \qquad (26)$$

Another metric that is sometimes used is the F-1 score, which is the harmonic mean of precision and recall. As both precision and recall do not include the true negatives in their computation, the F-1 score metric fails to represent the true performance of the classifier. Unlike the F-1 score metric, the MCC metric that we consider includes all the cells of the confusion matrix, thus providing a much more accurate evaluation of the frameworks.

4.2.2 RECEPTIVE FIELD LENGTH ANALYSIS

In the first experiment, we compare the performance of our TCNA architecture with four different receptive field lengths while the remaining hyperparameters are unchanged. We conduct this analysis to evaluate whether very long receptive lengths can help with a better understanding of normal system behavior. All the variants are evaluated based on their performance on two training metrics: average training loss and average validation loss, and the best model is selected for further comparisons.

Table 2 TCNA variants with different receptive field lengths

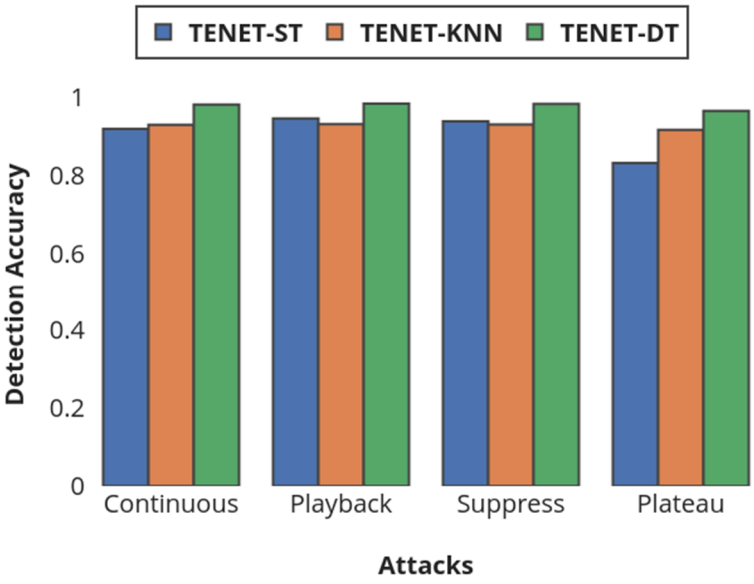
Framework	**Receptive Field Lengths**			
	16	**32**	**64**	**128**
Avg training loss	4.1e-4	3e-4	2.5e-4	6.8e-4
Avg validation loss	5.5e-4	4.3e-4	2.9e-4	9.3e-4

The average training loss value represents the average loss between the predicted behavior and observed behavior of each iteration in the training data. In contrast, the average validation loss represents the average loss between the predicted behavior and the observed behavior of each iteration in the validation data. Table 2 shows the average training and validation loss of the three variants of TCNA. We can observe that a receptive length of 64 has the lowest average training and validation loss. Therefore, we select 64 as our receptive field value, which is twice the maximum receptive field length presented in the comparison works (sequence length of 32 in [55]). This long receptive field length enables us to effectively learn very long-term dependencies in the input time series data and allows us to better understand the normal vehicle operating behavior.

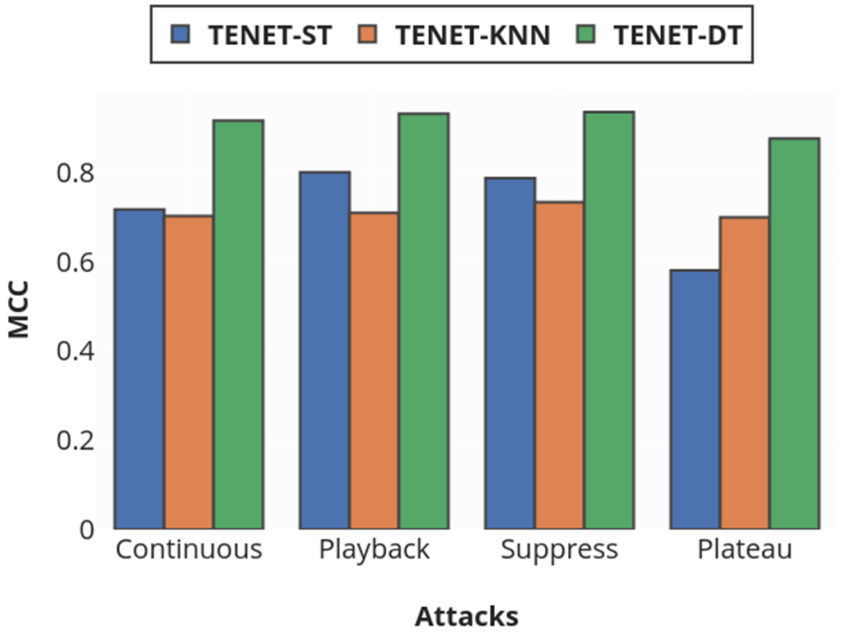
4.2.3 COMPARISON OF TENET CLASSIFIER VARIANTS

Next, we compare three different variants of *TENET* to study the impact of different classification techniques on *TENET's* anomaly detection capabilities. We considered non-linear classifiers such as decision trees (DT) and K-Nearest Neighbors (KNN), and a static threshold technique to classify a message as anomalous or normal. The three variants are labeled as *TENET-ST*, *TENET-KNN*, and *TENET-DT*. *TENET-ST* predicts whether a given message is normal or anomalous using the 99.99^{th} percentile of final validation loss as the threshold (as suggested in [54]). *TENET-KNN* uses a KNN classifier along with a kernel density estimator (KDE) to detect anomalies [55]. And lastly, *TENET-DT* uses our proposed decision tree based approach to detect anomalous messages. All variants utilize the same TCNA architecture we propose (presented in section 4.1) and the best performing *TENET* configuration from subsection 4.2.2 is used to predict the signal values of the next message instance. The detection accuracy,

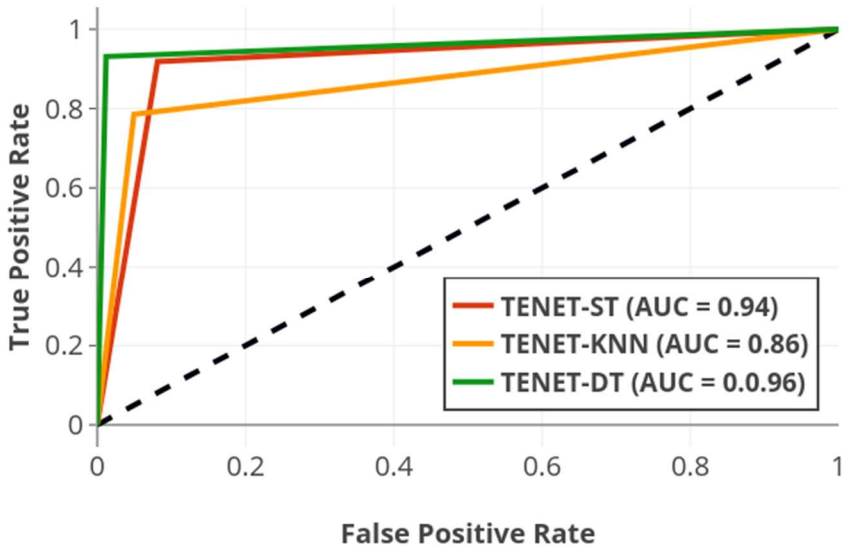
ROC curves with AUC, and MCC metrics for the three variants under four different attack scenarios are shown in Figure 22.



(a)



(b)



(c)

Figure 22 Comparison of (a) detection accuracy and (b) MCC of *TENET* variants under various attack scenarios, and (c) ROC with AUC for the playback attack.

In Figure 22(a), we can observe that the *TENET-DT* variant outperforms all the other variants for all attack scenarios by achieving an average accuracy of over 96% which is around 6.1% higher than the other two variants. Moreover, *TENET-DT* achieves an average of 19.7% increase in MCC metric compared to the other two variants for all attack scenarios, as can be seen in Figure 22(b). Figure 22(c) illustrates the ROC curves (with the area under the curve (AUC) in the legend) for the three variants in a playback attack scenario. The black dotted line in Figure 22(c) represents an AUC of 0.5 which is as good as random guessing. For brevity, we only show a playback attack for representing the ROC-AUC since it is the most difficult attack to detect. Recall that in a playback attack, the attacker tries to trick the IDS by replaying the previously observed messages. An IDS cannot easily detect these attacks unless it was properly trained to comprehend the normal sequence of messages. We can see that *TENET-DT*

outperforms other variants with an AUC of 0.96 along with the lowest false positive rate. Moreover, decision trees are a simple rule-based approach and can be trained quickly even if the input data size increases and produces fast results. Hence, we select the decision tree based classifier to detect anomalous messages in the in-vehicle network. Henceforth, we refer to *TENET-DT* as *TENET*.

4.2.4 PRIOR WORK COMPARISON

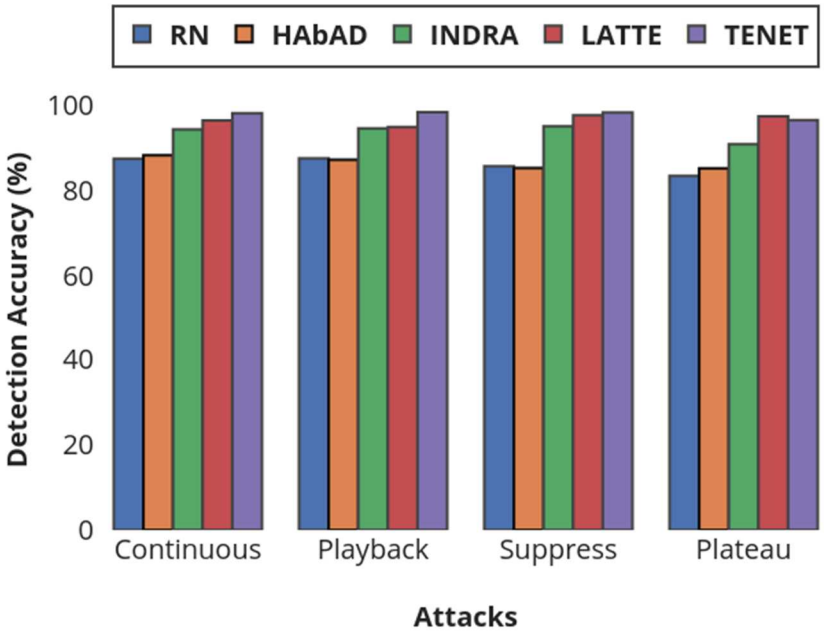
In this subsection, we compare our proposed *TENET* framework with various state-of-the-art prior works RN [56], INDRA [54], HABAD [55], and LATTE [16]. We evaluated the performance of the *TENET* framework, and the comparison works using detection accuracy, FPR, ROC curves with AUC, and MCC metrics, under various attack scenarios, and the results are as shown in Figure 23. From Figure 23 (a)-(e), we can observe that our *TENET* framework outperforms all comparison works in detection accuracy and MCC metrics for all attack scenarios except plateau attack. Moreover, LATTE [16] and *TENET* shows similar performance for the ROC-AUC metric. In addition, it can be observed that *TENET* achieves the second lowest FPR and FNR.

LATTE [16] shows better performance for detection accuracy, MCC, FPR, and FNR metrics only for plateau attack. However, LATTE [16] has relatively large memory footprint and model parameter (discussed in detail in subsection 4.2.5) compared to *TENET*. Moreover, INDRA [54] outperforms *TENET* for the FPR metric for all attack scenarios. However, the model shows poor performance for the FNR metric which is equally important because it tells the probability of an IDS to miss an actual attack. Table 3 summarizes the average percentage improvement of our *TENET* framework over the comparison works for all attack scenarios. A negative percentage

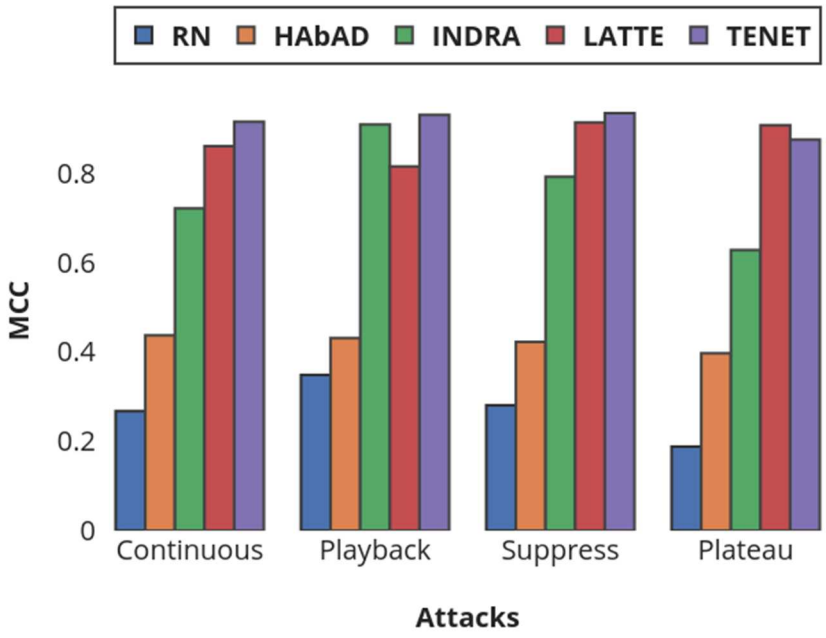
value indicated that TENET has underperformed for that metric and work. From Table 3, we can notice that our *TENET* framework achieves an improvement of up to 69.47% for the FNR metric, up to 64.30% for the MCC metric, up to 37.25% for the ROC-AUC metric, up to 9.48% for the detection accuracy, and up to 5.19% for the FPR metric for all attack scenarios.

Table 3 Average percentage improvement of TENET over comparison works

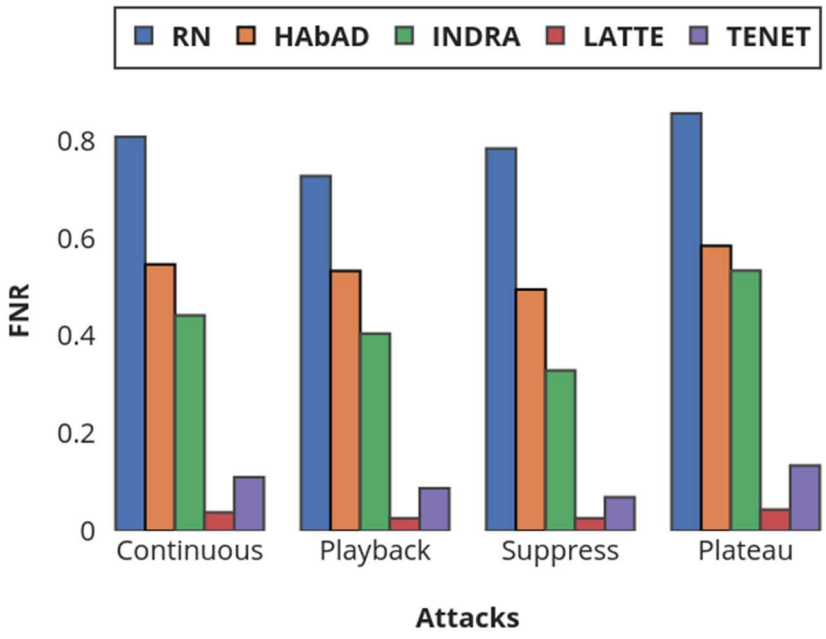
Prior Works	Detection accuracy	ROC-AUC	MCC	FPR	FNR
LATTE [16]	1.26	0.00	3.95	1.96	-6.63
INDRA [54]	3.32	17.25	19.14	-1.33	32.70
HABAD [55]	9.07	26.50	49.26	**5.19**	44.05
RN [56]	**9.48**	**37.25**	**64.30**	1.41	**69.47**



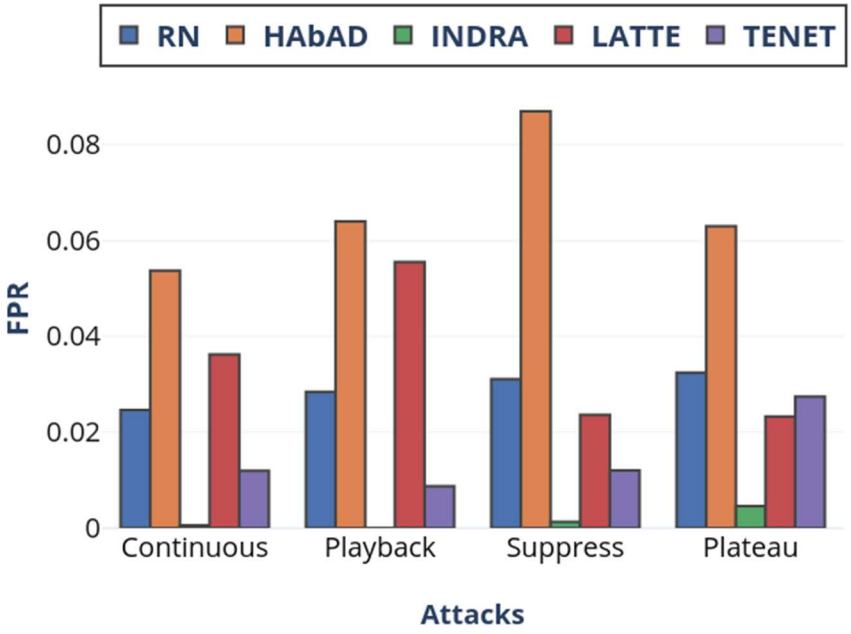
(a)


RN
HAbAD
INDRA
LATTE
TENET
MCC
0
0.2
0.4
0.6
0.8
Continuous
Playback
Suppress
Plateau
Attacks

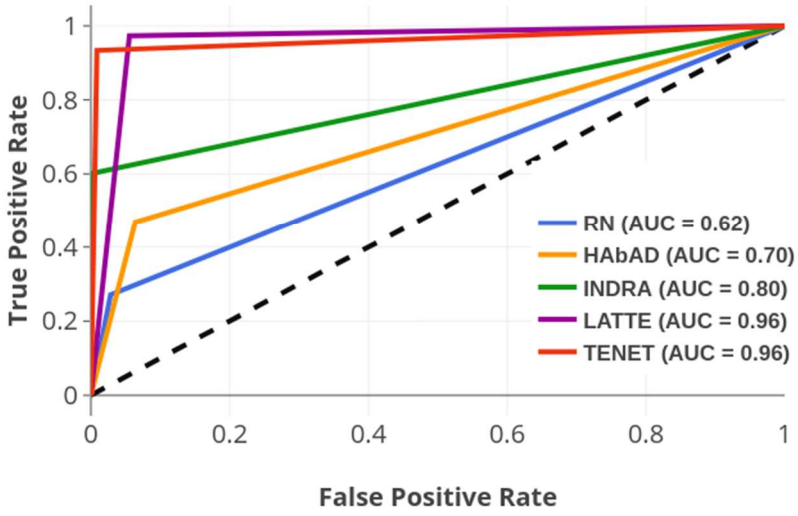
(b)


RN
HAbAD
INDRA
LATTE
TENET
FNR
0
0.2
0.4
0.6
0.8
Continuous
Playback
Suppress
Plateau
Attacks

(c)



(d)



(e)

Figure 23 Comparison of (a) detection accuracy, (b) MCC, (c) FNR, (d) FPR of *TENET* and comparison works under various attack scenarios, and (e) ROC with AUC for playback attack.

In a nutshell, our *TENET* framework with a novel TCNA network outperforms all the recurrent architectures without attention in detection accuracy and MCC metrics (from Table 3), due to its ability to capture very-long term dependencies in time-series data. Moreover, the attention mechanism within the TCNA improves the quality of the outputs of the TRB enabling efficient learning of very-long term dependencies. Thus, our novel TCNA network with the decision tree classifier results in a formidable anomaly detection framework.

4.2.5 TENET MEMORY OVERHEAD ANALYSIS

Lastly, we compare the number of trainable parameters and the memory footprint of the *TENET* framework, and the comparison works to evaluate the memory overhead of these models. It is important to consider the memory overhead of IDS models because automotive ECUs are resource constrained and it is crucial to have a model that does not interfere with the normal operation of safety-critical applications. Figure 24 illustrates the fast and lightweight nature of *TENET* and the earlier mentioned comparison works with the number of model parameters (on the x-axis), the memory footprint (on the y-axis), and the color bar represents the detection latency of comparison works in microseconds (μ_s). It can be seen from Figure 24 that *TENET* has the second lowest number of model parameters and memory footprint over all the other comparison works except RN [56]. Even though RN [56] has the least number of model parameters and memory footprint, it fails to capture the temporal dependencies between messages effectively, resulting in poor performance as seen in Figure 23 (a)-(e). Moreover, *TENET* (grey color) achieves the least detection latency among all compared works, shown in Figure 24.

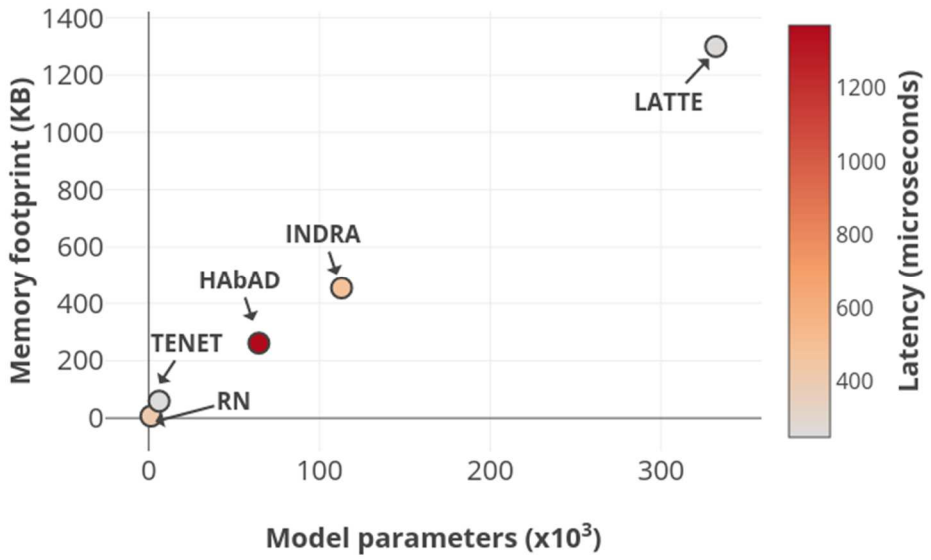


Figure 24 Comparison of memory overhead of *TENET* and the comparison works.

TENET achieves an average of 86.73% reduction in memory footprint against all comparison works, up to 98.17% reduction in the number of trainable model parameters. Moreover, the model memory footprint of *TENET* is 86.95% smaller compared to the model memory footprint of INDRA [54]. *TENET* achieves high performance with a very lower number of trainable parameters because of the fewer filters used by each DCC layer in the TCNA network. The attention block improves the quality of the outputs of each TRB thus eliminating the need for more filters, which in turn reduces the number of trainable parameters required by the model. Moreover, *TENET* also has the lowest inference time with an average of 34.24% reduction against all comparison works. *TENET* is able to achieve faster inferencing because, unlike recurrent architectures, *TENET* employs CNNs to process multiple subsequences in parallel, which helps reduce the inference time. Thus, *TENET* is able to achieve superior performance across various attack scenarios in automotive platforms with minimal memory and computational overhead.

4.3 CONCLUSION

In this chapter, we proposed a novel Temporal Convolutional Neural Attention (TCNA) network based anomaly detection framework called *TENET* for automotive cyber-physical systems. We also proposed a metric called the divergence score (DS), which measures the deviation of the predicted signal value from the actual signal value. We presented a sensitivity analysis for the selection of receptive field size and the detection classifier. We then compared the best variant of our proposed framework with the best-known prior works that employ a variety of sequence model architectures. *TENET* achieves up to 69.47% improvement in the FNR metric, up to 64.30% improvement in the MCC and up to 37.25% increment in the ROC-AUC metric over all the comparison works and for all attack scenarios. Moreover, *TENET* achieves up to 98.17% reduction in the number of model parameters and an average of 86.73% decrease in model memory footprint over all the comparison works. Therefore, these promising results indicate a compelling potential for utilizing our proposed approach in emerging automotive platforms.

5. CONCLUSION AND FUTURE WORK

5.1. RESEARCH CONCLUSION

In this thesis, we proposed efficient techniques to deploy machine learning based IDSs for in-vehicle network security. We started by conducting a comprehensive survey on state of art on different types of in-vehicle network IDSs. This survey determined that machine learning based IDSs were a realizable solution to the automotive security problem.

These findings were used to design two solutions for in-vehicle network security. The first technique uses a novel stacked LSTM with a self-attention framework called *LATTE* that learns the normal system behavior by learning to predict the next message instance under normal operating conditions. It also used a one class support vector (OCSVM) based detector model to detect cyber-attacks by monitoring the message deviations from the normal behavior. *LATTE* was able to achieve an average of 18.94% improvement in accuracy, 19.5% improvement in F1 score, 37% improvement in AUC and 79% reduction in false positive rate, and up to 47.8% improvement in accuracy, 37.5% improvement in F1 score, 76% improvement in AUC and 95% reduction in false positive rate. Therefore, *LATTE* offers a lightweight, scalable, and reliable solution to the intrusion detection problem.

Nonetheless, the increased automotive system complexity today has resulted in very long-term dependencies between messages exchanged between ECUs that cannot be effectively captured using LSTMs. This is mainly because the current time step output of LSTMs is heavily influenced by the recent time steps compared to past time steps, which makes it hard to capture

very long-term dependencies, e.g., for sequence lengths of 50 or more. Processing very long sequences also exacerbate the memory overhead of LSTMs. We want an IDS to efficiently capture and understand the very long-term dependencies to accurately detect anomalies in the network. Therefore, in the second technique we propose a novel anomaly detection framework called *TENET* based on Temporal Convolutional Neural Attention (TCNA) networks to learn very-long term temporal dependencies between messages in the in-vehicle network with low memory and computational overhead. *TENET* achieves an average improvement of 1.8% in FNR, 5.78% in detection accuracy, 20.25% in ROC-AUC, 34.16% in MCC, and 34.89% in FNR metric with 95.01% fewer model parameters, 86.73% decrease in memory footprint, and 34.24% lower inference time.

5.2. FUTURE WORK

Despite the many promising state-of-the-art AI-based IDS techniques that show compelling results, several issues still need to be addressed to make future autonomous vehicles truly secure. We present some key open challenges that represent promising opportunities for researchers to assist with achieving security goals in future autonomous vehicles.

5.2.1 INTRUSION RESPONSE SYSTEMS (IRS)

Detecting intrusions is the first step in the process of providing complete security to the in vehicle network. To achieve total security the identified threat needs to be handled in such a way that it does not cause any further undesirable events. Therefore, we need a response mechanism in place to handle the operation of the vehicle after detecting intrusion. One such mechanism is called an intrusion response system (IRS). IRSs have been employed in various fields to add an

extra layer of security. In the automotive domain, few works focus on intrusion response. In [60] authors proposed an intrusion tolerant architecture for autonomous driving. Their proposed approach used a simplex architecture to tolerate partially compromised automotive software on the ECU by extensively replicating critical services to hide the actions of a minority of compromised components. However, this work cannot handle a fully compromised ECU. A three-layer intrusion response framework was proposed in [61]. The proposed IRS module works in combination with the intrusion prevention system (IPS) in the first layer and IDS in the second layer. Their IRS is designed to use a unique predefined security strategy for different types of threats. However, this may fail to handle a novel threat scenario. With the increasing complexity of automotive cyber-physical systems and attack vectors, more comprehensive response strategies are required for future vehicles. Therefore, this domain represents a research gap that can be explored more in the future.

5.2.2 DATA PROTECTION AND PRIVACY

Data theft is a rapidly growing concern in today's world and is a prevalent issue across various industries. In 2020 alone, the average cost of a single data breach was around $3.86 million [1]. This is also a concern in future autonomous vehicles as the vehicles collect and operate on large volumes of data of different types. Data thefts have varying levels of safety, security, and economic impacts depending on the type and severity of the breach. Such thefts can compromise individual user data as well as intellectual property data of vehicle OEMs. With the recent introduction of next-generation driverless ridesharing services in places such as Phoenix and San Francisco, the stakes for user data privacy now is higher than ever. For instance, attackers can use stolen user information to launch more effective socially engineered attacks.

The issues of security, trust, and privacy in autonomous vehicles are presented in detail in [62]. Techniques such as Confidentiality Integrity Availability (CIA) and Distributed Immutable Ephemeral (DIE) models need to be adopted in the automotive domain to ensure data protection and privacy of future autonomous vehicles.

5.2.2 TAMPER-PROOF AI

AI algorithms have shown superior performance in IDS and ADAS subsystems for autonomous vehicles. However, these algorithms are vulnerable to carefully crafted adversarial attacks [8]. Moreover, with the rollout of increasingly connected vehicles, we envision that Black-hole DDoS attacks (where communication between vehicles is blocked) and Sybil attacks (where a vehicle operates with multiple identities) will become increasingly common. Such attacks will result in confusing AI algorithms, potentially causing failure across vehicle subsystems. Recent model inversion attacks [63] that try to reconstruct training data from the model parameters are gaining popularity. Such attacks pose a great threat to the proprietary data of the automakers that are used to train the AI models. Moreover, with newer and scalable learning approaches for large AI algorithms such as with federated learning in datacenter environments, the need for creating new approaches for tamper-proof and adversarial attack-resilient AI algorithms becomes even more imperative.

5.2.3 SECURING AUTOMOTIVE IC SUPPLY CHAINS

As different semiconductor integrated circuit (IC) components in a vehicle are manufactured in different parts of the world today, it is crucial to have a secure supply chain. Any vulnerability induced from the supply chain in any component of the vehicle will have

disastrous effects on autonomous vehicles. This issue is further exacerbated with the increasing demand for the RSUs and 5G infrastructure to enable intelligent transportation systems. A comprehensive list of IC supply chain concerns and a logic obfuscation technique to overcome them is presented in [64]. Techniques such as digital watermarking, IC fingerprinting, IC metering, etc., need to be further explored to ensure a secure supply chain.

5.2.4 ADOPTING EMERGING TECHNOLOGIES

In recent years, researchers have started looking into using WiGig networks that use IEEE 802.11ad multiple gigabit wireless system (MGWS) standard at 60 GHz frequency for in-vehicle network communication. The ability to support high data rates and enable low latency applications can transform the prospect of both in-vehicle networks and future self-driving applications. Another disruptive technology that could revolutionize future autonomous vehicles is blockchain technology. The blockchain's decentralized ledger provides accurate and simultaneous access to different types of data, such as traffic information and better vehicle tracking information for ride-sharing applications. However, as these technologies are still in their infancy in the automotive domain, they need to be meticulously scrutinized by characterizing vulnerabilities and exploring security mechanisms to enhance security.

BIBLIOGRAPHY

[1] "Upstream Security's 2021 Global Automotive Cybersecurity Report," 2021. [Online]. Available: https://upstream.auto/2021report/.

[2] V. K. Kukkala, J. Tunnell, S. Pasricha and T. Bradley, "Advanced Driver-Assistance Systems: A Path Toward Autonomous Vehicles," *IEEE Consumer Electronics Magazine,* vol. 7, pp. 18-25, 2018.

[3] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental Security Analysis of a Modern Automobile," in *IEEE Symposium on Security and Privacy*, 2010.

[4] C. Valasek, and C. Miller, "Remote Exploitation of an Unaltered Passenger Vehicle," *Black Hat USA,* 2015.

[5] F. D. Garcia, D. Oswald, T. Kasper, and P. Pavlidès, "Lock it and still lose it—on the (in) security of automotive remote keyless entry systems," in *USENIX security symposium*, 2016.

[6] L. Wouters, E. Marin, T. Ashur, B. Gierlichs, and B. Preneel, "Fast, Furious and Insecure: Passive Keyless Entry and Start Systems in Modern Supercars," *IACR Transactions on Cryptographic Hardware and Embedded Systems,* vol. 2019, no. 3, pp. 66-85, 2019.

[7] L. Wouters, B. Gierlichs, and B. Preneel, "My other car is your car: compromising the Tesla Model X keyless entry system," *IACR Transactions on Cryptographic Hardware and Embedded Systems,* vol. 2021, no. 4, pp. 149-172, 2021.

[8] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust Physical-World Attacks on Deep Learning Visual Classification," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2008.

[9] J. Petit, B. Stottelaar, M. Feiri, and F. Kargl, "Remote Attacks on Automated Vehicles Sensors: Experiments on Camera and LiDAR," *Black Hat Europe,* 2015.

[10] "Experimental Security Research of Tesla Autopilot," 2019. [Online]. Available: https://keenlab.tencent.com/en/2019/03/29/Tencent-Keen-Security-Lab-Experimental-Security-Research-of-Tesla-Autopilot/.

[11] Z. Hau, K. T. Co, S. Demetriou, and E. C. Lupu, "Object removal attacks on lidar-based 3d object detectors," 2021. [Online]. Available: https://arxiv.org/abs/2102.03722.

[12] R.P Weinmann and B. Schmotzle, "TBONE – A zero-click exploit for Tesla MCUs," 2020. [Online]. Available: https://kunnamon.io/tbone/tbone-v1.0-redacted.pdf.

[13] "Kia Motors America suffers ransomware attack, $20 million ransom," 2021. [Online]. Available: https://www.cpomagazine.com/cyber-security/kia-motors-america-suffers-a-20-million-suspected-doppelpaymer-ransomware-attack/.

[14] C. Sitawarin, A.N. Bhagoji, A. Mosenia, M. Chiang, and P. Mittal, "DARTS: Deceiving Autonomous Cars with Toxic Signs," 2018. [Online]. Available: https://arxiv.org/abs/1802.06430.

[15] J. Dürrwang, J. Braun, M. Rumez, R. Kriesten, and A. Pretschner, "Enhancement of Automotive Penetration Testing with Threat Analyses Results," in *Society of Automotive Engineers (SAE)*, 2018.

[16] V. K. Kukkala, S. V. Thiruloga, and S. Pasricha, "LATTE: LSTM Self-Attention based

Anomaly Detection in Embedded Automotive Platforms," *in ACM Transactions on Embedded Computing Systems (TECS),* vol. 20, no. 5s, pp. 1-23, 2021.

[17] S. V. Thiruloga, V. K. Kukkala, and S. Pasricha, "TENET: Temporal CNN with Attention for Anomaly Detection in Automotive Cyber-Physical Systems," in *IEEE/ACM Asia & South Pacific Design Automation Conference (ASPDAC)*, 2022.

[18] "Isms family of standards," International Organization for Standardization, Geneva, 2015.

[19] I. Studnia, E. Alata, V. Nicomette, M. Kaâniche, and Y. Laarouchi, "A language-based intrusion detection approach for automotive embedded networks," *International Journal of Embedded Systems (IJES),* vol. 10, 2018.

[20] P. Waszecki, P. Mundhenk, S. Steinhorst, M. Lukasiewycz, R. Karri and S. Chakraborty, "Automotive Electrical and Electronic Architecture Security via Distributed In-Vehicle Traffic Monitoring," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 36, pp. 1790-1803, 2017.

[21] G. C. DiDomenico, J. Bair, V. K. Kukkala, J. Tunnell, M. Peyfuss, M. Kraus, J. Ax, J. Lazarri, M. Munin, C. Cooke, E. Christensen, L. Peltz, N. Peterson, L. Wolfe, Z. Vinski, and D. Norris, "Colorado State University EcoCAR 3 Final Technical Report," in *Society of Automotive Engineers (SAE)*, 2019.

[22] J. Schmidhuber, "Habilitation thesis: System modeling and optimization," 1993.

[23] J. F. Kolen and S. C. Kremer,, "Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies," in *A Field Guide to Dynamical Recurrent Networks*, Wiley-IEEE Press, 2001, pp. 237-243.

[24] K. Cho, B. V. Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y.

Bengio, "Learning phrase representations using RNN encoderdecoder for statistical machine translation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[25] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano and K. J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE Transactions on Acoustics, Speech, and Signal Processing,* vol. 37, pp. 328-339, 1989.

[26] S. Bai, J. Z. Kolter, and V. Koltun, "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling," 2018. [Online]. Available: https://arxiv.org/abs/1803.01271.

[27] H. M. Song, H. R. Kim and H. K. Kim, "Intrusion detection system based on the analysis of time intervals of CAN messages for in-vehicle network," in *International Conference on Information Networking (ICOIN)*, 2016.

[28] M. Marchetti and D. Stabili, "Anomaly detection of CAN bus messages through analysis of ID sequences," in *IEEE Intelligent Vehicles Symposium (IV)*, 2017.

[29] T. Hoppe, S. Kiltz, and J. Dittmann, "Security threats to automotive CAN networks—Practical examples and selected short-term countermeasures," *Reliability Engineering & System Safety,* vol. 96, no. 11, pp. 11-25, 2011.

[30] M. Gmiden, M. H. Gmiden and H. Trabelsi, "An intrusion detection method for securing in-vehicle CAN bus," in *International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*, 2016.

[31] H. Lee, S. H. Jeong and H. K. Kim, "OTIDS: A Novel Intrusion Detection System for In-vehicle Network by Using Remote Frame," in *Annual Conference on Privacy, Security and Trust (PST)*, 2017.

[32] U. E. Larson, D. K. Nilsson and E. Jonsson, "An approach to specification-based attack detection for in-vehicle networks," in *IEEE Intelligent Vehicles Symposium*, 2008.

[33] M. Aldwairi , A. M.Abu-Dalo and M. Jarrah, "Pattern matching of signature-based IDS using Myers algorithm under MapReduce framework," *EURASIP Journal on Information Security,* 2017.

[34] E. W. Myers, "An O(ND) difference algorithm and its variations," *Algorithmica,* vol. 1, pp. 251-266, 1986.

[35] T. Hoppe, S. Kiltz, and J. Dittmann, "Applying intrusion detection to automotive IT-early insights and remaining challenges," *Journal of Information Assurance and Security (JIAS),* pp. 226-235, 2009.

[36] D. Stabili, M. Marchetti and M. Colajanni, "Detecting attacks to internal vehicle networks through Hamming distance," in *AEIT International Annual Conference*, 2017.

[37] K. T. Cho and K. G. Shin, "Viden: Attacker identification on in-vehicle networks," in *ACM SIGSAC Conference on Computer and Communications Security*, 2017.

[38] K. T. Cho and K. G. Shin, "Fingerprinting Electronic Control Units for Vehicle Intrusion Detection," in *25th USENIX Security Symposium*, 2016.

[39] X. Ying, S. U. Sagong, A. Clark, L. Bushnell and R. Poovendran, "Shape of the Cloak: Formal Analysis of Clock Skew-Based Intrusion Detection System in Controller Area Networks," *IEEE Transactions on Information Forensics and Security,* vol. 14, pp. 2300-2314, 2019.

[40] M. Yoon, S. Mohan, J. Choi and L. Sha, "Memory Heat Map: Anomaly detection in real-time embedded systems using memory behavior," in *IEEE Design Automation Conference*

(DAC), 2015.

[41] M. Müter and N. Asaj, "Entropy-based anomaly detection for in-vehicle networks," in *IEEE Intelligent Vehicles Symposium (IV)*, 2011.

[42] W. Wu, Y. Huang, R. Kurachi, G. Zeng, G. Xie, R. Li, and K. Li, "Sliding window optimized information entropy analysis method for intrusion detection on in-vehicle networks," *IEEE Access,* vol. 6, pp. 45233-45245, 2018.

[43] M. Müter, A. Groll and F. C. Freiling, "A structured approach to anomaly detection for in-vehicle networks," in *International Conference on Information Assurance and Security*, 2010.

[44] A. Taylor, N. Japkowicz and S. Leblanc, "Frequency-based anomaly detection for the automotive CAN bus," in *World Congress on Industrial Control Systems Security (WCICSS)*, 2015.

[45] F. Martinelli, F. Mercaldo, V. Nardone and A. Santone, "Car hacking identification through fuzzy logic algorithms," in *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2017.

[46] T. P. Vuong, G. Loukas and D. Gan, "Performance Evaluation of Cyber-Physical Intrusion Detection on a Robotic Vehicle," in *IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, 2015.

[47] M. Levi, Y. Allouche and A. Kontorovich, "Advanced analytics for connected cars cyber security," in *IEEE Vehicular Technology Conference (VTC Spring)*, 2017.

[48] M. Kang and J. Kang, "A novel intrusion detection method using deep neural network for

in-vehicle network security," in *IEEE 83rd Vehicular Technology Conference (VTC Spring)*, 2016.

[49] S. F. Lokman, A. T. Othman, S. Musa, and M. H. Abu Bakar, "Deep contractive autoencoder-based anomaly detection for in-vehicle controller area network (CAN)," *Progress in Engineering Technology. Advanced Structured Materials,* vol. 119, 2019.

[50] M. Hanselmann, T. Strauss, K. Dormann and H. Ulmer, "CANet: An Unsupervised Intrusion Detection System for High Dimensional CAN Bus Data," *IEEE Access,* vol. 8, pp. 58194-58205, 2020.

[51] G. Loukas, T. Vuong, R. Heartfield, G. Sakellari, Y. Yoon and D. Gan, "Cloud-Based Cyber-Physical Intrusion Detection for Vehicles Using Deep Learning," *IEEE Access,* vol. 6, pp. 3491-3508, 2018.

[52] A. Taylor, S. Leblanc and N. Japkowicz, "Anomaly detection in automobile control network data with long short-term memory networks," in *IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, 2016.

[53] M. D. Hossain, H. Inoue, H. Ochiai, D. Fall and Y. Kadobayashi, "LSTM-based intrusion detection system for in-vehicle can bus communications," *IEEE Access,* vol. 8, pp. 185489-185502, 2020.

[54] V. K. Kukkala, S. V. Thiruloga and S. Pasricha, "INDRA: Intrusion Detection using Recurrent Autoencoders in Automotive Embedded Systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD),* vol. 39, 2020.

[55] M. O. Ezeme, Q. H. Mahmoud and A. Azim, "Hierarchical Attention-Based Anomaly Detection Model for Embedded Operating Systems," in *IEEE International Conference on*

Embedded and Real-Time Computing Systems and Applications (RTCSA), 2018.

[56] M. Weber, G. Wolf, B. Zimmer, and E. Sax., "Online Detection of Anomalies in Vehicle Signals using Replicator Neural Networks," in *ESCAR USA*, 2018.

[57] M. Weber, S. Klug, B. Zimmer, and E. Sax, "Embedded Hybrid Anomaly Detection for Automotive CAN Communication," in *European Congress on Embedded Real Time Software and Systems (ERTS)*, 2018.

[58] S. Tariq, S. lee, and S. S. Woo, "CANTransfer: Transfer learning based intrusion detection on a controller area network using convolutional lstm network," *ACM Symposium on Applied Computing,* pp. 1048-1055, 2020.

[59] 2018. [Online]. Available: https://assets.vector.com/cms/content/products/gl_logger/Docs/GL1000_Manual_EN.pdf.

[60] M. Vöelp and P. Esteves-Verissimo, "Intrusion-Tolerant Autonomous Driving," in *IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*, 2018.

[61] M. Hamad, M. Tsantekidis, and V. Prevelakis, "Intrusion Response System for Vehicles: Challenges and Vision," in *International Conference on Vehicle Technology and Intelligent Transport Systems*, 2021.

[62] G. Muhammad and M. Alhussein, "Security, Trust, and Privacy for the Internet of Vehicles: A Deep Learning Approach," in *IEEE Consumer Electronics Magazine*, 2021.

[63] S. Chen, M. Kahla, R. Jia, and G. J. Qi, "Knowledge-Enriched Distributional Model Inversion Attacks," in *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[64] K. Shamsi, M. Li, K. Plaks, S. Fazzari, D. Z. Pan, and Y. Jin, "IP protection and supply

chain security through logic obfuscation: A systematic overview," *ACM Transactions on Design Automation of Electronic Systems (TODAES),* vol. 24, pp. 1-36, 2019.