POLITECNICO DI MILANO

Facoltà di Ingegneria

Scuola di Ingegneria Industriale e dell'Informazione

Dipartimento di Elettronica, Informazione e Bioingegneria

Master of Science in

Computer Science and Engineering

Anomaly Detection System for Automotive CAN using LSTM Autoencoders

Supervisor:

PROF. STEFANO ZANERO

Assistant Supervisor:

STEFANO LONGARI

Master Graduation Thesis by:

DANIEL HUMBERTO NOVA VALCÁRCEL
Student Id n. 814247

Academic Year 2018-2019

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Automotive security has gained significant traction in the last decade thanks to the development of new connectivity features that have brought the vehicle from an isolated environment to an externally facing domain. A vehicle has several embedded Electronic Control Units (ECUs), these are sensors and actuators used from simple tasks to safety-critical systems. Controller Area Network (CAN) is the defacto protocol to communicate these ECUs, it has proven to be efficient, however, it presents several drawbacks in this new interconnected scenario. Researchers have presented vulnerabilities on ECUs, infotainment systems, and telematics units that can be exploited to gain access to the CAN bus. With access to the bus, an attacker can interfere with the normal flow of information, effectively altering the state of the car and putting the safety of drivers and others at risk.

To detect these attacks, security experts and researchers have proposed several countermeasures, including the use of Intrusion Detection Systems (IDSs) which make for a great part of the published research. Given the pervasiveness of CAN and the infeasibility of changing the protocol, IDSs provide a tool to analyze CAN traffic that does not interfere with the regular operation of the bus.

In this work, propose a data sequence anomaly detection approach that uses Long Short-Term Memory (LSTM) autoencoders. These models learn to reconstruct sequences using exclusively normal CAN traffic data. Our system uses the difference between reconstructions and real sequences to create a distribution of reconstruction errors, which describe legitimate traffic behavior. Using a set of simulated attacks, we demonstrate that sequence anomalies effectively alter this distribution and can be used for detection. We demonstrate that LSTM autoencoders are on average more effective at detecting anomalies on CAN data than the current state-of-the-art LSTM predictor-based detector; we run our anomaly tests on both detectors using a real-world dataset and the same experimental framework.

SOMMARIO

L'ambito della sicurezza informatica in campo automobilistico ha ricevuto una crescente attenzione dagli esperti e ricercatori nell'ultimo decennio. Ciò è dovuto agli sviluppi di nuove interfacce di comunicazione che hanno portato il veicolo dall'essere un ambiente isolato ad un dominio che si interfaccia con reti esterne. Ogni veicolo ha diversi sistemi embedded chiamati unità di controllo elettronico (o ECU). Queste centraline sono composte da sensori e attuatori che controllano diverse funzionalità, da quelle semplici come il controllo dei climatizzatori, a quelle critiche per la sicurezza come i sistemi anti bloccaggio dei freni. Diversi sistemi richiedono i dati forniti da più ECU, che pertanto devono essere collegate tra di loro. Il protocollo standard che collega tra di loro queste ECU all'interno del veicolo è chiamato Controller Area Network (CAN). CAN ha dimostrato di essere un protocollo efficiente che permette la comunicazione in tempo reale, tuttavia non è stato progettato tenendo conto dei requisiti di sicurezza informatica, non essendo prevedibile all'epoca l'inclusione, nel veicolo, di interfacce con sistemi esterni.

Le vulnerabilità delle ECU sono state illustrate da diversi lavori di ricerca. È stato per esempio dimostrato che i sistemi infotainment o le unità di telematica possono essere utilizzate per ottenere accesso alla rete CAN. Ottenuto l'accesso, un attaccante può alterare il normale funzionamento delle ECU all'interno del veicolo, e quindi mettere a rischio la sicurezza delle persone attorno al veicolo.

Diversi metodi sono stati proposti per rilevare i potenziali attacchi. Tra questi, l'uso di Intrusion Detection System (IDS) è quello sostenuto dalla maggior parte dei ricercatori ed esperti di sicurezza. Una delle principali problematiche dell'uso di IDS in campo automobilistico è che questi sistemi devono essere progettati per individuare attacchi, ma non è possibile ottenere esempi documentati di attacchi avvenuti realmente, si possono solo generare attacchi in laboratorio. Questa limitazione forza l'utilizzo di Anomaly Detection System (ADS), che funzionano creando un modello di riferimento che rappresenta il comportamento normale del sistema, in modo che qualsiasi deviazione da questo modello possa essere considerata come una anomalia.

Nello stato dell'arte si trovano degli ADS per sistemi automotive che monitorano la frequenza dei messaggi, le caratteristiche fisiche dei segnali delle ECU, oltre che le sequenze di dati CAN. Analizzare sequenze di dati ci permette di rilevare anomalie che possono passare inosservate con altri metodi. Molti ADS di questo tipo usano metodi di machine learning per creare i modelli di riferimento, tra i quali ci sono le recurrent neural networks con Long Short-Term Memory

(LSTM). Tra questi metodi si trovano gli autoencoders, una tecnica di apprendimento non supervisionato che permette di generare una rappresentazione compatta di sequenze di input. Questo tipo di reti neurali ci permette di creare un modello del meccanismo generatore dei dati, che impara a ricostruire le sequenze.

In questa tesi proponiamo un metodo di anomaly detection in sequenze di dati CAN usando LSTM autoencoders. Questi vengono allenati usando traffico CAN di un veicolo reale, senza bisogno di esempi di attacchi. Il nostro sistema usa la differenza tra le ricostruzioni e le sequenze reali per generare una distribuzione degli errori di ricostruzione, che descrive le caratteristiche del traffico in condizioni non-anomale.

Paragonando la distribuzione degli errori in condizioni normali con le distribuzioni ottenute con sequenze anomale simulate, dimostriamo che le seconde sono significativamente diverse rispetto alle prime. Questa differenza ci fornisce la base su cui creare un meccanismo che distingua sequenze anomale. Dimostriamo mediante esperimenti con dati CAN di un veicolo reale, condotti con gli stessi parametri sperimentali, che gli LSTM autoencoders sono in media più efficaci nel rilevamento di anomalie dei modelli di predizione su reti LSTM, che sono l'attuale stato dell'arte.

1 INTRODUCTION

The automotive industry has seen gigantic growth in the last thirty years due to the rapid development of new hardware and software components that are embedded to improve efficiency, reliability, and more user-experience oriented features; this phenomenon was driven by many factors including environmental and safety regulations, but also out of market necessity. Before this change vehicles were mainly mechanic, they included a small amount of networked electronic components used to display data on the dashboard and to execute simple tasks. Nowadays, we can find up to a hundred connected Electronic Control Units (ECUs) in a modern vehicle. These embedded systems control most of the vehicle's functionalities from simple tasks such as opening or closing the windows, to safety critical components like the engine control units, anti-lock braking systems, airbag deployment, accessing diagnostics data, etc. Many of the vehicle systems depend on signals produced by other ECUs and, consequently, units are required to communicate in an efficient manner due to the safety requirements implied by the automotive system. To interconnect these units, in 1986 Robert Bosch Gmbh presented the CAN bus specification [6], a simple but efficient solution to the problem of interconnecting ECUs that meets the requirements for real-time communication and low deployment costs. CAN is considered as the de-facto standard in automotive internal networks even to this day [4] and is also commonly found in other industrial applications.

Although the introduction of new features strongly improves safety and driving experience, they have also expanded the attack surfaces exploitable by potential malicious agents. In order to mitigate these new threats we must tackle a dual-faced problem: on one we have to manage systems that were not designed with cybersecurity in mind, on the other side these cannot be easily replaced or updated given their pervasive use and the complexity of the automotive supply-chain; each manufacturer buys components from several providers which also depend on other suppliers and so on, therefore patching or substituting specific components can be an impractical or prohibitively expensive task.

This new scenario, consisting of new potential threats and systems that are difficult to adapt, has caused an increased attention to automotive security. In the last decade, a significant amount of research work on automotive security has been published, and the attention on the topic has been steadily increasing ever since. Shortly after automotive security became an novel and interesting topic, new research was published demonstrating vulnerabilities and experimental attacks on controlled laboratory environments [32, 8, 24]. Interest from the academia and from security experts grew rapidly after researchers presented attacks carried out in real vehicles, catching attention from the industry and media outlets as well [22, 43, 44]. Countermeasures and Intrusion Detection Systems (IDSs) proposals for vehicles immediately followed applying the vast knowledge of the computer security field to the automotive world.

Intrusion Detection System (IDS) monitor the events in a computer system or network for signs of intrusions In the context of automotive security, we adopt a definition of intrusion that carries the greatest risk, which refers to malicious agents that have gained access to the CAN bus. IDS can be classified depending on the methodology they follow: signature-based, specification-based, and anomaly-based intrusion detection [35]. Signature and specification-based detection systems assume the knowledge of known attacks and of a strict description of the protocol; however, this information is not available since databases with attack data on CAN in real world scenarios are not yet available, and specifications on the signals generated by ECUs are generally not available [60]. Because of the lack of attack examples, a specification-based IDS is no longer a suitable solution. Instead, ADS attempt to characterize normal traffic data and deviations from this norm are considered anomalous and have been proposed for CAN security applications [48].

In this chapter, we begin by introducing the motivations behind the need for anomaly detection on automotive networks, specifically on the CAN bus. We then introduce an overview of the countermeasures that have been proposed to make the bus more secure. Next, we outline our proposal for the problem of anomaly detection. Finally, we briefly summarize our contributions and define the outline for this thesis.

1.1 MOTIVATIONS

1.1.1 *Vehicles as targets for cyber attacks*

The introduction of remote telematic units, Internet-connected infotainment systems, and Bluetooth interfaces have brought to the automotive market new and interesting connectivity features, but they have also brought the CAN bus to a connected domain where it is no longer an isolated network. These new externally-facing interfaces turn the vehicle into a potential target for cyber attacks. CAN was not designed with cybersecurity considerations in mind since connecting the bus to external interfaces was not considered a possibility at the time of its conception, hence the priorities at the time pushed towards following real-time system requirements and cost minimization. With a lack of authentication, encryption, and access control mechanisms, the protocol has a significant number of inherent weaknesses [32]. A combination of new external interfaces, the drawbacks of the protocol, and even oversight by some manufacturers, have given security researchers the tools to successfully compromise devices connected to the CAN bus, giving them control of the bus and in consequence effectively altering the state of the vehicle.

These security issues have been exploited by security researchers to prove the feasibility of attacks. The publication of the most notable attacks presented by Checkoway et al. [8] and Miller and Valasek [43, 42, 44] demonstrated the feasibility to carry out attacks on vehicles under real operating circumstances, this served as a concrete proof of the need for cybersecurity on vehicles. And with this, automotive cybersecurity stopped being just a theoretical possibility and turned into a real concern.

To carry out an attack, one of the most crucial steps for an attacker is successfully gaining access to the CAN bus. Once they are able to read and transmit messages they can potentially control most of the vehicle functionality including, but not limited to, controlling the steering wheel, brakes, speed, airbag deployment, Anti-Lock Braking System (ABS), and even shutting down the car entirely. For that reason guaranteeing the correct operation of the vehicle and its components is a safety-critical task. Any malicious alteration can lead to life-threatening situations not only for the driver and their passengers, but also to surrounding vehicles, pedestrians, infrastructure, the environment, and more. Additionally, the prospect of autonomous cars on the streets have created a greater demand to enhance automotive cybersecurity [69].

1.1.2 *Countermeasures*

Along with the relatively new importance given to automotive cybersecurity, researchers and security experts have proposed new detection systems and countermeasures to mitigate potential attacks. These include enhancing the protocol, tools to verify specification-compliance, ECU fingerprinting, and Intrusion Detection Systems (IDSs). Some of the proposals to enhance the CAN protocol include introducing authentication [71, 55]. However, altering the protocol itself is not considered feasible as it would require significant modifications on how the existing networks operate; leading to the non-trivial task of managing firmware patches across millions of existing vehicles and across complex manufacturer supply chains. In consequence, this leads to issues with the inclusion of authentication, encryption, and similar mechanisms used in traditional computer networks, as they would require modifications of the protocol itself.

IDSs instead do not require protocol alterations and they do not interfere with regular communication over CAN at all. They can be implemented on a single unit purposed exclusively for detection, and, as a result, backward compatibility would not be an issue. Additionally, costs are significantly mitigated with respect to patching firmware for dozens of ECUs per vehicle. These considerations render IDSs an appropriate and feasible security mechanism.

Initially, research on CAN-based detection systems focused on analyzing the frequency of messages transmitted on the bus, this choice was supported by the fact that most CAN traffic is periodic and because most attacks tend to alter the frequency at which messages appear on the bus. However, these proposals fail to detect scenarios where an attacker is able to mimic the correct frequencies and is still able to inject malicious data.

Another approach focused on attempting to create ECU fingerprints, providing a pseudo-authentication mechanism [12, 11, 9]; however, some proposals have been deemed inadequate in real operating circumstances, and others have been successfully bypassed by following research work [57].

Inspired by the extensive research on time-series anomaly analysis, researchers have developed IDSs that use machine learning algorithms to detect anomalous sequences in data streams, even in automotive applications. In this framework, only sequences of data are analyzed, looking for suspicious contents that may signal an attack. The main goal of this approach is to find anomalies in streams of data, caused by contextual inconsistencies in sequences or by anomalous data points; this implies that it is indispensable to model what characterizes normal/legitimate traffic.

Understanding the meaning of CAN signals is a crucial step when evaluating what is the most appropriate approach. Knowing the data generating mechanism and its semantics is useful when defining what differentiates legitimate and suspicious messages. However, databases that specify these semantics are usually kept confidential by vehicle and ECU manufacturers, forcing researchers to look for alternative approaches to process these signals. In consequence, developing semantics-unaware systems is an unfortunate but unavoidable requirement.

1.2 PROPOSED APPROACH

RNNs are considered good modelers for time-series data [1]. They have been proposed as a solution for anomaly detection problems, showing promising results even in semantics-unaware scenarios. Additionally, they are considered a natural fit when dealing with multivariate time-series (as is the case of CAN) [67]. Anomaly detectors using RNNs work by creating a baseline of what constitutes the normal behavior of a time-series, and any deviation from such model is considered anomalous. In the context of intrusion detection this can signal an attack.

One of the main advantages of RNNs is that they learn to model sequences of data rather than isolated observations. This allows us to detect anomalies from single data points outside the normal data distribution, and, most importantly, from data that is anomalous in the *context* of other observations. Such context is defined as sequential data, just like a time-series. Classical RNN units are rarely used in practice because they struggle with learning long sequence dependencies, this problem is caused by an uncontrolled growth or shrinkage of its gradients . Instead, these networks normally use Long Short-Term Memory (LSTM) units, which are more complex and can learn values over arbitrary time intervals, overcoming the limitations of classic RNNs. By learning how a sequence behaves, we can also learn how to reconstruct it, that is attempting to recreate sequences based on the data seen before. This has uses in many areas, including dimensionality reduction, removing noise from data [20], and even anomaly detection.

Detection approaches that use RNNs can be divided into two categories: *prediction*-based and *reconstruction*-based anomaly detection. Predictors have been used to model CAN traffic data, and have been shown to perform well in most anomaly scenarios [67, 66]. However, this approach relies on a very strong assumption stating that CAN traffic is predictable to some extent; given the semantics-unaware context of our problem, this assumption has to be re-evaluated.
Alternatively, sequence reconstruction approaches have been shown to be efficient at detecting anomalies in time-series that are inherently unpredictable [38]. Given the fact that ECUs signal semantics are unknown to us, we deemed this approach to be a suitable choice for CAN. To our understanding, this approach has not yet been implemented and tested in an automotive context.

In this thesis, we present an ADS for CAN that applies a reconstruction-based approach using LSTM autoencoders. This unsupervised approach learns a latent representation of CAN traffic data sequences using only normal/legitimate data. The resulting autoencoder is used to reconstruct sequences, and through this process, we compute the difference between reconstructed and real sequences. We demonstrate that in the presence of anomalies, reconstruction errors are significantly altered, thus we can utilize this error metric as a tool for detecting attacks. We demonstrate the effectiveness of this approach in cases where the data semantics are unknown. We compare our approach with the prediction-based detector presented in previous research, and finally we evaluate the two detectors in the context of CAN anomaly detection.

Our approach can be summarized in the following steps:

1. *CAN data evaluation.* We collect and analyze CAN traffic data, and classify it based on frequency and variability. From this process, we are able to infer some structure using an algorithm proposed in the literature.

2. *Simulating attacks.* Since CAN traffic databases with attack examples are not available, we use simulated attack data to test our system. Such data is based on attacks presented in published work, and on threat models of previous research.

3. *Predictor-based detector replication.* We implement the predictor proposed in previous research, using our own data to evaluate its performance.

4. *Reconstruction models as anomaly detection mechanism.* We evaluate the use of autoencoders to reconstruct CAN data sequences, and propose an architecture based on our dataset.

5. *Evaluation.* Finally, we test the predictor and autoencoder with the same anomalous data, evaluating and comparing their performance.

1.3 CONTRIBUTIONS

In this thesis, we propose a novel approach to anomaly detection of data sequences in automotive networks using LSTM autoencoders. Our autoencoder is trained to reconstruct the normal CAN traffic obtained from a real vehicle, and requires no prior availability of attack examples to both train the model and to fine-tune its anomaly scoring mechanism. We compare our proposal with a state-of-the-art prediction-based detector found in the literature. We show that, on average, the autoencoder outperforms the predictor under the same anomaly scenarios, demonstrating that an autoencoder can either complement the predictor in a detection system or even substitute it.

Our objectives for this work can be summarized as:

- Implement and evaluate the performance of the state-of-the-art LSTM predictor approach with our own data, employing the same semi-supervised approach as in the reference work.

- Assess the use of autoencoders as time-series modelers, for which we do not make any assumptions on its predictability, using a completely unsupervised approach.

- Compare both approaches using the same data for both training and anomaly testing. Evaluate their use in automotive intrusion detection.

The key contributions of our work are:

1. The presentation a reconstruction-based approach using LSTM autoencoders that can efficiently model normal CAN traffic.
2. The demonstration that using autoencoders in an unsupervised detection scheme can effectively detect anomalies in CAN data under different attack scenarios.
3. Proposing the use of autoencoders as a suitable addition to intrusion detection systems for vehicles that can either work together with predictor approaches or even substitute them.

1.4 STRUCTURE OF THE WORK

This thesis is organized as follows:

In Chapter 2 we introduce the necessary background on CAN, present a preliminary analysis of the CAN traffic data we use throughout this work, and introduce an approach to find potential structure in the data.

Chapter **??** introduces the relevant background on CAN vulnerabilities, an overview of the published attacks, and a review the state-of-the-art of detection systems for in-vehicle networks. We also give an overview of RNNs, LSTMs, and autoencoders. We conclude with an overview of the state-of-the-art on anomaly detection using RNNs.

Chapter 4 introduces our solution to the problem, here we describe the threat model and define how we evaluate performance. We follow with an explanation of predictor-based detection. Then, we present an autoencoder-based detection approach. We conclude by explaining how these two methods are evaluated and compared.

In Chapter 5 we present the experimental setup and how we prepare the data. We follow with an explanation of how we train the models and how we set up the post-processing. We continue by presenting the detection results, showing how detectors perform under the anomaly test scenarios.

Finally, in Chapter 6 we summarize our work, draw our conclusions, and define the path for future work.

2 THE CONTROLLER AREA NETWORK

2.1 INTRODUCTION

In this chapter we introduce the Controller Area Network (CAN), the most prevalent protocol in automotive networks, and we frame it in a cybersecurity context.

We begin by describing the protocol, including the assumptions and requirements taken into account during its design. We focus on the relevant aspects within the context of our anomaly detection problem. We then present a preliminary analysis of CAN traffic data obtained from a real vehicle. This analysis is an overview of the data we use in our experiments, used to obtain insight about the behavior of the different IDs in the bus. We conclude this chapter by presenting the *field identification algorithm*, used to find potential structures in CAN messages.

2.2 OVERVIEW OF CAN IN AUTOMOTIVE

CAN is a serial bus communication protocol, it defines the lowest layers of the Open Systems Interconnection (OSI) model, that is, the physical and data link layer. CAN was designed for distributed real-time control [6], and is suitable for embedded applications and real-time systems, providing data transfer rates up to 1Mb/s [27]. It is widely used in industrial machinery but is most commonly known for its widespread use in vehicles internal networks. In fact, it has been used as a mandatory standard for cars manufactured in Europe since 2001, and in the United States since 2008 [60]. In automotive applications, the protocol is used to connect the ECUs placed all around the vehicle. These units control several subsystems that need to exchange data to meet functional requirements while also fulfilling, among others, requirements on fault tolerance, real-time communication, arbitration policies, among others.

CAN runs on a fairly simple physical medium, typically consisting of two twisted-pair wires and one 120-Ohm resistance on each end of the bus for termination. The two wires are known as CAN-H and CAN-L, and they are also used as a fault tolerance mechanism. During transmission, the bus uses differential signaling that raises the voltage on one wire and drops it in the other [60]. Typically, buses with two lines are exclusively used for safety-critical tasks and run at high speeds. Non-critical components typically use a simpler and cheaper single wire significantly lower speeds, e. g. air conditioning,

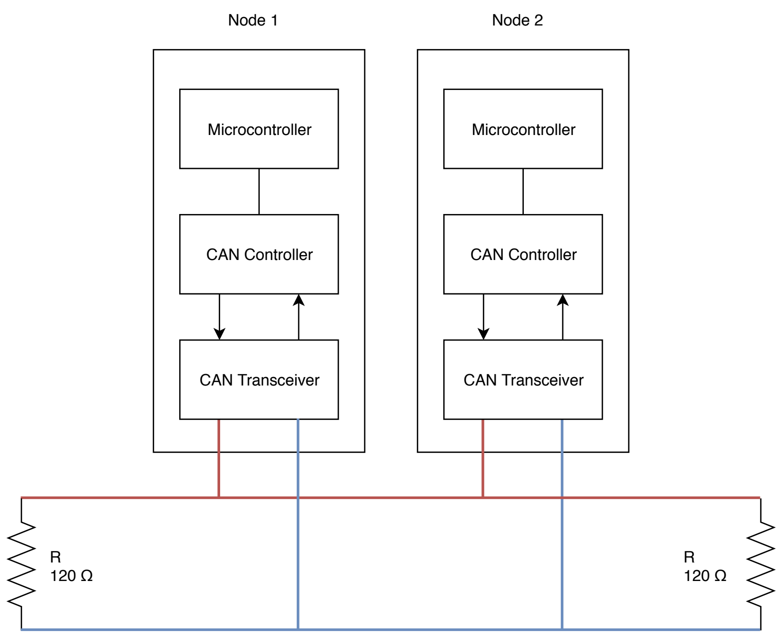
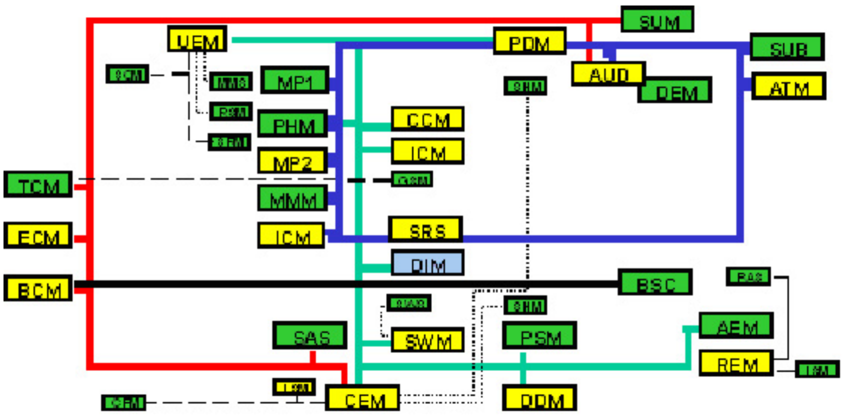


Figure 2.1: Example of a CAN network with two nodes connected using CAN-H and CAN-L.

door control modules. Traffic on the high-speed bus is usually characterized by the presence of high-frequency messages, where ECUs transmit on average every 10-20 milliseconds. Additionally, high priority units tend to be very active in the network, so high-frequency messages tend to be associated with important tasks and safety-critical modules [27], making them a higher priority when designing security systems.

Figure 2.1 illustrates a simplified view of a CAN bus, where nodes are individual ECUs with a CAN transceiver, a CAN controller, and a microcontroller.

It is common to find separate buses in a vehicle, the internal network is usually divided into subdomains each with its own bus. This provides some network segmentation that could potentially isolate issues that occur in one of the buses. However, some components depend on data that is transmitted on buses from different domains; because of this, manufacturers use gateway bridges to connect different buses [32]. For example, a Volvo XC90 has separate buses connecting up to 40 ECUs[27], as illustrated in Figure 2.2. In this network, safety-critical control modules like the Engine Controle Module (ECM), Brake Controle Module (BCM), and Transmission Controle Module (TCM) are connected to a bus running at 500 kbps. A separate network, running at 125 kbps, connects non safety-critical components like the climate control, and infotainment modules. In this vehicle, the central electronic module acts as a gateway between the two buses. Another example of network bridging can be found in



	Powertrain and chassis		Body electronics
TCM	Transmission control module	CEM	Central electronic module
ECM	Engine control module	SWM	Steering wheel module
BCM	Brake control module	DDM	Driver door module
BSC	Body sensor cluster	REM	Rear electronic module
SAS	Steering angle sensor	SWM	Steering wheel module
SUM	Suspension module	DDM	Driver door module
AUD	Audio module	PDM	Passenger door module
		REM	Rear electronic module
	Infotainment/Telematics	CCM	Climate control module
MP1,2	Media players 1 and 2	ICM	Infotainment control
PHM	Phone module	UEM	Upper electronic module
MMM	Multimedia module	DIM	Driver information module
SUB	Subwoofer	AEM	Auxiliary electronic
ATM	Antenna tuner module		

Figure 2.2: Control architecture of a Volvo XC90, network segmentation is present to separate different components. Taken from [27].

the 2010 Ford Escape, where the Accessory Protocol Interface Module and the Instrument Cluster effectively bridge both [**CAN**] buses [43].

2.2.1 *CAN packets*

CAN operates as a broadcast network connecting all ECUs that communicate on the bus, with no sender or receiver information attached. The ID, specified at the beginning of every frame, defines the meaning of the data being transmitted rather than its origin. It is important to note that in the network, an ID should be exclusively sent by only one ECU, however, this is not a guaranteed and it must be taken into account when designing systems that analyze CAN packets. IDs also define the priority of messages through arbitration, which is determined according to the frame's ID; with lower IDs indicating a higher priority message.

CAN provides four different kinds of frames, each with its own semantics and structure. There are the *data*, *remote*, *error*, and *overload* frames:

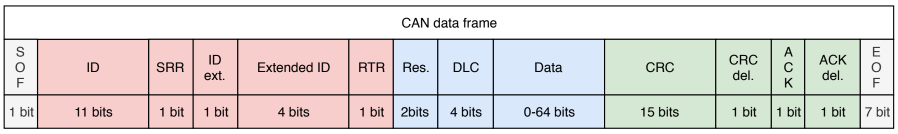


Figure 2.3: CAN Data Frame.

- *Data frames* carry data from a transmitting ECU. For example, a frame containing the steering wheel angle.
- *Remote frames* are used to request the transmission of a data frame, using the ID to signal which frame is needed. For example, A is the unit responsible for transmitting messages with ID `0x01`, another unit B can send a remote frame with ID `0x01` to request A to send a data frame. Usually these frames are not used, as data frames are typically sent at specific time intervals.
- *Error frames* are transmitted when a bus errors occur. For example, when badly formed frames are transmitted.
- *Overload frames* signal a delay of the next data frame because the transmitting ECU is overloaded at the moment.

In the context of this thesis, we only focus on the *data frame*. When reading data from the bus, the CAN controller automatically handles overload and error frames, so we only record data frames. Remote frames are rarely used, so they are practically almost never seen on the bus.

As illustrated in Figure 2.3, each frame is composed of the following fields:

1. **SOF**, start of frame that uses a dominant bit, i.e. 0.
2. An **arbitration field** that can use either an 11-bit or extended 29-bit ID, depending on the protocol version in place. This field consists of:
   a) The base ID, which is 11 bits.
   b) **SRR**, substitute remote request, typically used for extended packets.
   c) **IDE**, Identifier extension. Set to 1 if it uses an extended ID, otherwise set to 0.
   d) Extended ID, with additional 18 bits.
   e) **RTR**, Remote transmission request. Used the message is a request frame.
3. The **control field** consists of 2 reserved bits followed by a **DLC**, specifies how many bytes will be transmitted. Its value can vary between 0 and 8 bytes.

4. **Data field**, the payload of the frame. It can be 8 bytes at maximum, length specified by the DLC.

5. The **Cyclic Redundancy Checksum (CRC)** field is used for data integrity check. It is used by a receiver to check for possible data corruption. A CRC delimiter is set afterwards, with its bit always set to 1.

6. **ACK** is used as acknowledgment and is two bits long. A transmitter sets these two bits as recessive bits (i.e. 1), and a receiving unit acknowledges a successful transmission by setting the first bit to a dominant bit (i.e. 0). If the ACK fails, an error frame is sent into the bus and the transmitter of the original frame must retransmit it. The ACK delimiter is always set to 1.

7. Lastly, an **EOF** consists of seven consecutive recessive bits, indicating the end of the frame.

Additionally, data frames are separated from each other by an interspace frame, consisting of three recessive bits. During their transmission, no ECU is allowed to transmit into the bus to guarantee a minimal gap between frames.

TRANSMITTING PAYLOADS OVER 8 BYTES. In some scenarios, it is required to transmit messages with a payload bigger than 8 bytes. The ISO-TP Protocol 15765-2 specifies the transmission of bigger payloads, it works by segmenting messages into multiple frames with additional meta-data, which is used to reconstruct the entire message at the receiving unit- This protocol allows a significant payload size increase summing in total 4095 bytes per packet, and is commonly used in diagnostic tasks [18].

2.2.2 *OBD*

OBD is a subset of the Unified Diagnostic Services (UDS) protocol, used by manufacturers and technicians to provide services for diagnostics, calibration, and ECU firmware management. Some diagnostic operations include additional security mechanisms because of the risk factors involved; a challenge-response authentication mechanism is put in place to allow any modification or access to the ECU state.

Most vehicles are equipped with an OBD port, to allow an external party to run diagnostic queries. The port is directly connected to the CAN bus, and is usually found below the steering wheel [60]. One of the most common uses of the OBD port is to query and clear fault codes stored in some of the car modules. Routine checks are performed by one of the ECUs. When any subsystem encounters a fault or error, a Diagnostic Trouble Code (DTC) is stored in the Power Train Control Module (PCM), either in memory or persistent storage, and

```
          Timestamp   ID              Data
0 1549035061.89032  0FB  00121FB401000186
1 1549035061.89049  0F4  1900000000000121
2 1549035061.89073  1EF  8800000000000EDF
3 1549035061.89095  256  0000007FCC410000
4 1549035061.89120  259  000008B670000000
```

Figure 2.4: Example of individual recordings of CAN traffic data of an Alfa Giulia Veloce vehicle.

can be displayed as a warning on the dashboard. A typical example of the use of the OBD port is clearing the "Check Engine" indicator on the dashboard. Through the port it is possible to know the reasons behind this warning and to turn the indicator off.

While it was originally meant to help technicians with their repairing/diagnosing tasks, the OBD port is also used by telematics aftermarket units connected directly to it. These units provide telematics information to drivers and insurance companies [19].

2.3 PRELIMINARY ANALYSIS OF CAN TRAFFIC

For this thesis, we collect logs of CAN traffic data from a real vehicle to use on our experiments. After collection, the first step is to study the data at hand. We are able to obtain some insight into our dataset, despite the lack of signal-semantic knowledge.

In this Section, we present our dataset and study its characteristics. These include obtaining the number of IDs, packet counts and their frequency, and how the available space is used. Finally, we study the symbol counts for each ID, which gives us a basic understanding of the behavior of the data contents.

2.3.1 *Data collection*

We gathered CAN messages from a 2017 Alfa Romeo Giulia Veloce, and use them to build our dataset. We collected the data from the OBD port, which allowed us to read CAN traffic directly from the bus. The data gathering process took place during a driving session that lasted 65 minutes. As data recording was taking place, the vehicle was driven inside the city and also on the highway.

2.3.2 *Dataset*

Our dataset consists of over 6 million messages. Each message has four variables which define the columns of our dataset. We describe them below, and show an example of the recordings in Figure 2.4:

```
count        5,471,373.00000
mean                 7.49818
std                  1.18750
min                  1.00000
1%                   3.00000
10%                  6.00000
18%                  8.00000
50%                  8.00000
max                  8.00000
Name: DLC, dtype: float64
```

Figure 2.5: DLC usage in our CAN recordings dataset.

- **Timestamp**. Numerical variable that indicates the time when the message was transmitted. It represents Unix time (also known as POSIX time, or Unix epoch time), which counts the number of seconds since midnight of January 1st, 1970.
- **ID**. An hexadecimal value representing the message identifier.
- **DLC**. An integer for the data length code, it specifies the payload size.
- **Data**. Payload for a specific message in hexadecimal representation.

2.3.3 *Data Characteristics*

To have a better understanding of the characteristics of our dataset we perform a preliminary analysis on the CAN traffic logs we collected. Here we present our most relevant findings.

ID USE. We identify 83 unique IDs in our dataset, each appearing at different frequencies.

DLC USE. By studying the distribution of DLCs, we observe that most packets use all the available 8 bytes of the data frame. 89% of messages set their DLC to 8 bytes, around 10% to 6 bytes, and the rest make use of 1 to 3 bytes for their payload. Additionally, we confirmed that every ID uses a constant DLC throughout the dataset, so the size of the data field can be considered constant.

Despite the fact that most IDs use all the available payload size, several of the data buts remain constant throughout our dataset. We observe that constant bits are a common factor across IDs, albeit each one in different distributions. In table 2.1, we show the varying degrees of constant bits used across a list of IDs.

ID	DLC value	Non-constant bits
0DE	48	41
0FF	64	42
100	64	43
0F0	64	51
0EE	64	60
0FE	64	55
0FB	64	40
104	64	47
116	64	52
11C	64	49
0FC	64	37
1F7	64	50
1FB	64	25

Table 2.1: Number of bits used for each ID as set in its DLC field, and how many of those bits are non-constant in our dataset.

PACKET COUNTS PER ID. A handful of IDs produce most of the packets found in the dataset. 18 IDs alone account for 56% of the total traffic, as illustrated in Figure 2.6.

PACKET FREQUENCY. By computing the timestamp differences between messages with the same ID, we can estimate their respective transmission frequency. This allows us to determine which IDs transmit messages at a periodic rate, and which ones are aperiodic. 73 ID show seemingly constant frequencies, confirming the assumption that most CAN traffic is periodic. We illustrate the timestamp differences for each ID in Figure 2.7.

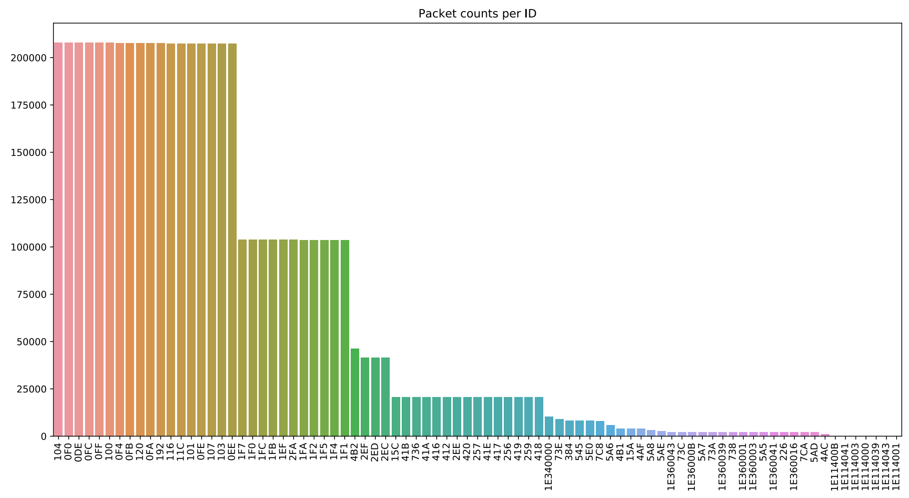


Figure 2.6: Packet counts by ID, from a sample of the CAN dataset.

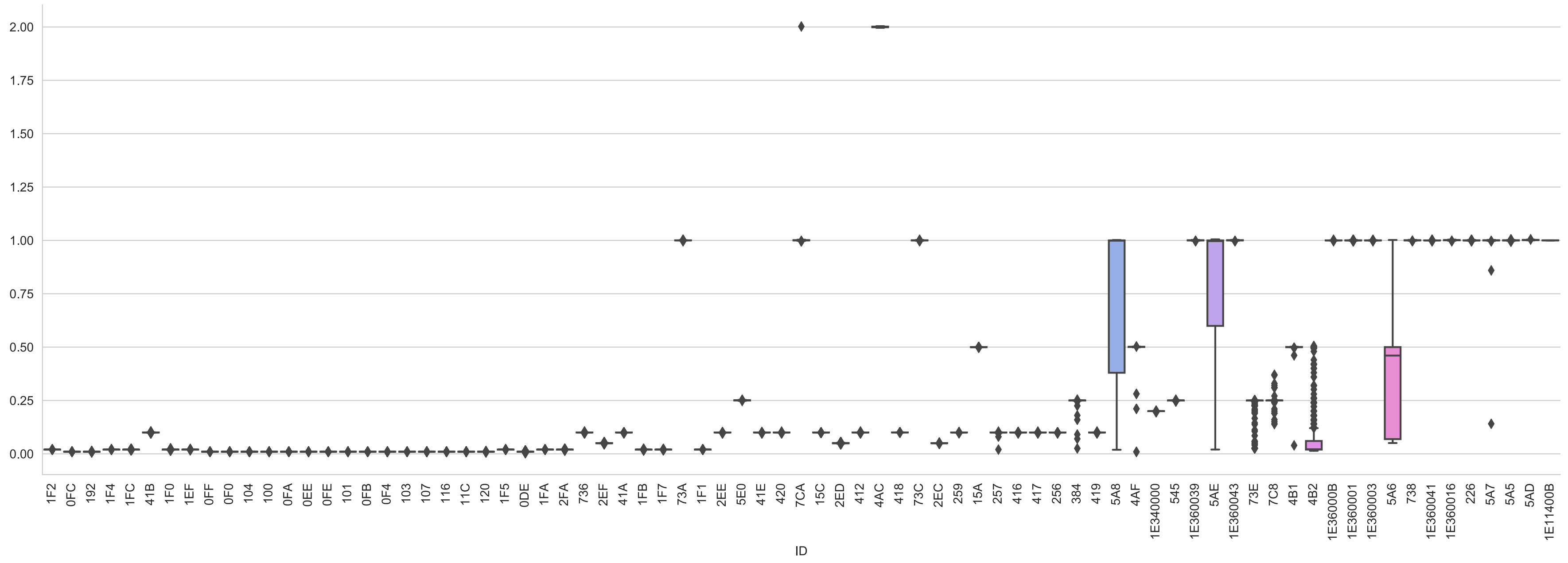


Figure 2.7: Estimated frequencies for all IDs in the CAN dataset. The x-axis represents the different IDs, and the y-axis the estimated frequency.

We divide periodic IDs into three categories depending on their estimated frequency. We present this classification in Table 2.2, along with how many IDs correspond to each category.

Classification	Estimated frequency (s)	Number of IDs
High Frequency	< 0.025	29
Medium Frequency	[0.25, 0.5]	20
High Variability	>0.5	27

Table 2.2: Classification of IDs according to their estimated frequency (in seconds) for our dataset.

SYMBOL COUNTS For our data sequence anomaly detection problem, the most important field to consider is the data field (i.e. its payload). The field can be up to 64 bits long, allowing every ID to produce at most 2^{64} unique data symbols. After analyzing the data, we can observe that several IDs only use a small set of unique symbols, while others continuously create new ones. We count the number of unique symbols to define the *variability* of each ID, and divide them into three categories: ***Low, Medium***, and ***High*** variability IDs. We follow a similar classification as used in previous research [66].
Table 2.3 shows the ranges of unique symbols for each category, and how many IDs belong to each one; we can observe that most IDs have high symbol variability. We also observed that 23 IDs only produce one unique symbol.

Classification	Number of unique symbols	Number of IDs
Low Variability	<100	23
Medium Variability	[100, 500]	7
High Variability	>500	30

Table 2.3: Classification of IDs according to their unique symbol count in the CAN dataset.

In Figure 2.8, we illustrate the symbol creation rate for four high frequency IDs. This shows that CAN IDs continually create new symbols at a seemingly steady or growing rate.

For this work, we choose 12 IDs that belong to both high-frequency and high symbol variability categories, with more than 1% of their total symbol count being unique. We chose these characteristics as they are most likely related to safety-critical systems, and also because they present the greatest challenge to model given their symbol creation rates. The chosen IDs are:

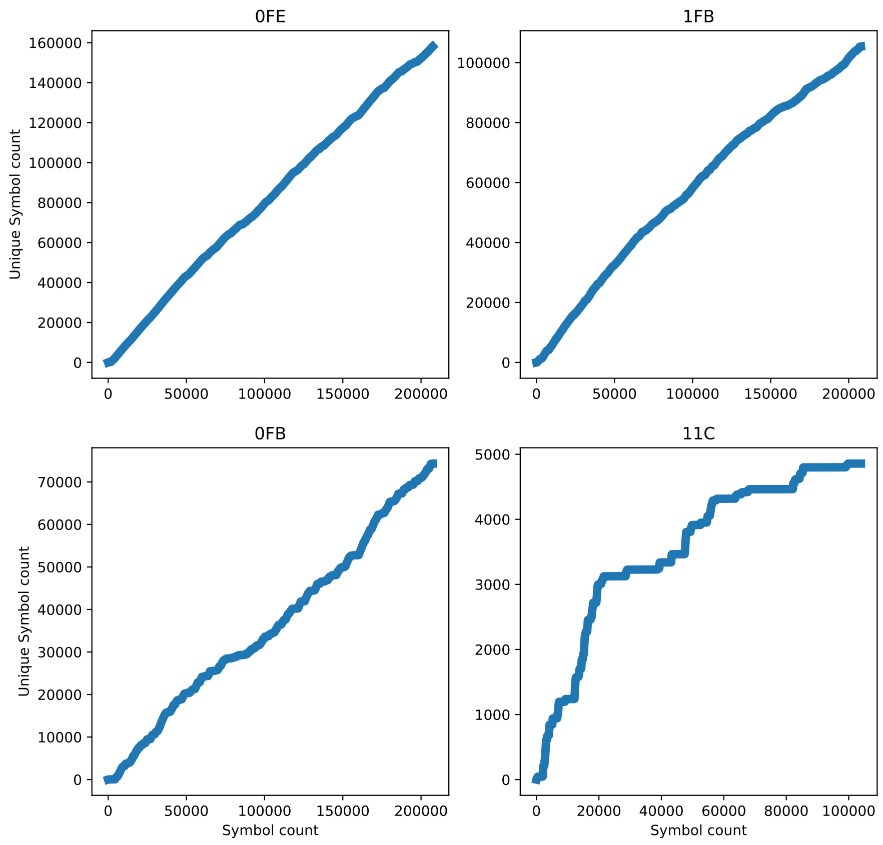


Figure 2.8: Symbol creation rate for four CAN IDs.

`0DE, 0EE, 0FB, 0FC, 0FE, 0FF, 1F7, 1FB, 11C, 100, 104, 116.`

2.4 FINDING STRUCTURE IN CAN DATA

The data field typically has a substructure of its own. It is divided into several fields, each one with its own specific semantics. For example, a single packet could contain both data on the vehicle speed and revolutions per minute (rpm), both represented as 32-bit integers: the speed being stored on the first four bytes of the data field, and rpm in the remaining ones. Usually, not all the available bits are used, and some are set to a constant value. These bits seem to act as separators in the data frame. We can analyze the symbol variability to define potential substructures in the messages. In Figure 2.9, we illustrate the bit average values for a list of high-frequency CAN IDs, using over 200,000 logged CAN messages from the dataset. From this figure, we can visually identify some potential separators.

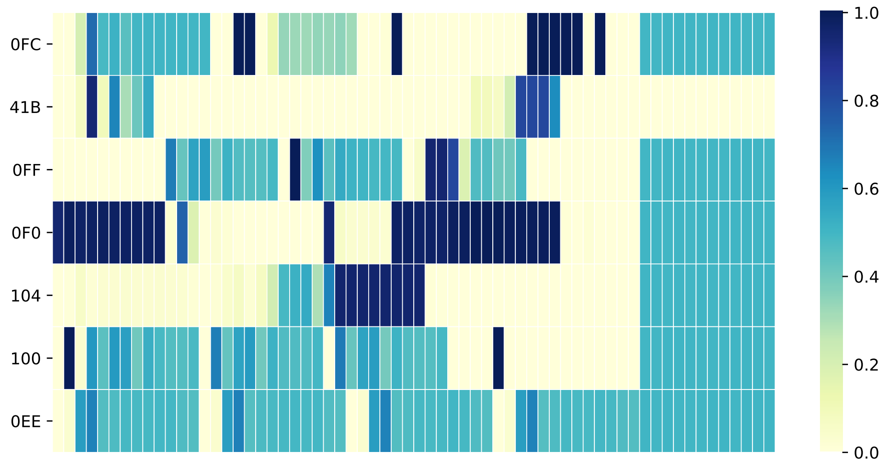


Figure 2.9: Bit average values for a single CAN IDs.

2.4.1 *Field Identification Algorithm*

To identify potential structures in CAN frames, Markovitz and Wool [41] proposed an algorithm that finds semantically meaningful structure in CANs messages when ECUs specifications are not previously known. The structure of messages consists of a concatenation of positional bit fields of fixed length. By collecting CAN traffic logs, the authors attempted to classify the fields present in the data frames. This resulted in the identification of four field types with clear semantics: *constant*, *multi-value*, *counter*, and *sensor*. This classification depends on factors such as the length of each candidate field and its respective amount of unique bit counts.

FIELD TYPES Each field can be classified as:

1. *Constant.* Sets of bits with non-changing values.
2. *Multi-Value.* Fields that exhibited a set of few different unique values. A minimum field length (denoted as $T_{L_{min}}$) and a maximum symbol count (denoted as $T_{V_{max}}$) are set as thresholds to classify fields as multi-value.
3. *Counter.* Values that behave like a cyclic counter with minimum and maximum values.
4. *Sensor.* Represent a measurement of a physical quantity, these are continuous with some noise. In practice any field with a symbol count greater than $T_{V_{max}}$.

However, in the experiments *Counter* and *Sensor* were grouped into a single class. A procedure to separate these categories was left for future research.

PROCEDURE Using sequences of CAN data as input, the algorithm consists of the following steps:

1. *Split into fields.* Create a 64×64 matrix f to represent all possible divisions. In this lower triangular matrix, columns represent the index of the fields' left bit. Rows represent each field's length. For example, a candidate in $f_{[10,4]}$ represents a field that starts at index 4 and has a length of 10 bits. Similarly, $f_{[64,1]}$ represents a candidate field that uses the whole message.

2. *Count Values.* For each candidate in f, count how many unique values k exist in the given dataset.

3. *Types and Scores.* Set a field type for every candidate and compute a score for each one. For each candidate, types and scores are set using the following criteria:
   a) Set to *Constant* if the number of unique values equals one. The score is set as the length of the field.
   b) Set to *Multi-Value* if the unique values are less than $T_{V_{max}}$, and its length is at minimum $T_{L_{min}}$. The score is set as the length of the field.
   c) Set to *Sensor* if the candidate was not set to the other types. The score is computed as $\frac{k^2}{s^l}$, where k is the number of unique values for the candidate field, and l is the length.

4. *Choose fields.* Iteratively traverse the candidates matrix following this priority scheme: *Constant* $\succ$ *Multi-Value* $\succ$ *Sensor*. When comparing candidates of the same type, its priority is set by its score, with higher scores indicating higher priority. At each iteration, the algorithm removes all overlapping fields until no candidates remain.

The algorithm outputs a list of disjoint fields, each containing values indicating its index, length, type, number of unique values, and score. An example of the output for one CAN ID is shown in Table 2.4.

However, the greedy algorithm proposed by Markovitz and Wool works by sequentially finding local optima in the group of possible candidates, and in consequence cannot guarantee that the results are optimum.

The field classification algorithm by Markovitz and Wool was proposed in a research paper in anomaly detection for automotive networks. Their ADS proposal is based on Ternary Content Addressable Memorys (TCAMs), commonly used in network switches and routers. However, their ADS proposal was not tested, and their main contribution is the field identification algorithm.

Type	Index	Length	Category	unique symbols
MULTI-VALUE	0	5	LOW VAR	10
SENSOR	5	11	HIGH VAR	197
CONST	16	8	N/A	N/A
MULTI-VALUE	24	5	LOW VAR	10
SENSOR	29	6	MID VAR	64
MULTI-VALUE	35	5	LOW VAR	7
SENSOR	40	6	MID VAR	64
CONST	48	4	N/A	N/A
SENSOR	52	12	HIGH VAR	4081

Table 2.4: Resulting field division from the field identification algorithm by Markovitz and Wool. It consists of a list of disjoint fields for a single CAN ID.

3 THREAT MODEL & RELATED WORK

3.1 INTRODUCTION

In this chapter we present the threat model and the relevant background on anomaly detection on CAN. We divided this chapter into two parts. In the first part we summarize the vulnerabilities and attacks targeting the CAN bus, and from this analysis we present the threat model. In the second part of this chapter, we present a summary of the most notable detection systems for CAN classified by their common characteristics. Then we present an introduction to RNNs, LSTM, and autoencoders; this section provides the context needed to understand some of the approaches used in anomaly detection. We conclude this chapter by presenting an overview of the solutions to anomaly detection on multivariate time-series, including their use in automotive applications.

3.2 THREAT MODEL

3.2.1 *Overview of vulnerabilities*

Security experts and researchers have been able to compromise the security of vehicles due to the significant expansion of attack surfaces on automotive systems. New attack vectors exist as a result of introducing new connectivity features, combined with several vulnerabilities that exist in the vehicle ECUs and other components connected to the bus.

Some of the earliest experimental assessments on automotive vulnerabilities were published at the beginning of the decade. Koscher et al.[32] performed an experimental security analysis of modern vehicles, showcasing the issues on their networks and demonstrating that almost every ECU in the vehicle can be compromised. They begin by highlighting the inherent security challenges that the CAN protocol presents, which include:

- The broadcast nature of the protocol.
- Its fragility to Denial of Service (DoS) attacks, easily implemented by flooding the bus with high priority messages.
- Lack of authentication.
- Weak access control. It can allow an attacker to read or modify the state of ECUs, and to reflash their firmware. Koscher et

al. showcased examples of how the challenge-response authorization mechanism in the protocol specification can be circumvented by a determined attacker.

Automotive protocols standards specify countermeasures to avoid some safety risks, for example, not allowing firmware updated while the vehicle is moving. However, in some cases, manufacturers implement them poorly or completely ignore them. Koscher et al. [32] found instances where components and units do not follow the security requirements of the protocol, which can be exploited by malicious agents to alter the state of the vehicle. The most notable issues they found include:

- Allowing unsafe commands to be accepted by ECUs while the vehicle is being driven. For example, executing the "disable all CAN communications" command and reflashing firmware.
- Non-compliant access control. It allowed the researchers to read sensitive memory and alter the state of several units. This was caused due to a poor implementation of the challenge-response authentication mechanism, which used hardcoded response keys.
- Imperfect network segmentation. Because of the dependencies between components, data has to be exchanged across buses and domains, and so gateways are put in place to bridge these networks. Researchers discovered that they were able to reprogram these gateways from the low-speed bus, going against the protocol specifications that state that gateway reprogramming must only be possible in the high-speed network.

Later, Checkoway et al. [8] studied additional vulnerabilities, on their work they classified automotive attack surfaces into three categories:

- Direct/indirect physical access to the CAN bus (e.g. OBD port, CD players, USB interfaces, infotainment systems). These present lack of encryption, authorization, and authentication.
- Short-range wireless (e.g. Bluetooth, RF-based remote keyless entry, etc.). These present buffer overflow vulnerabilities, weak encryption, and a lack of authentication and authorization
- Long-range signals (e.g. GPS, satellite/digital radio, cellular interfaces). These present weak encryption, and a weak of authentication, and weak authorization mechanisms.

Long-range surfaces have been shown to have surprisingly weak security mechanisms. Some examples of this include the use of common passwords in Internet-facing components and the re-use of encryption keys [52]. Each of these surfaces presents different vulnerabilities that can be exploited with varying levels of difficulty, but they have been experimentally proven to be feasible.

3.2.2 *Attacks on the CAN bus*

Attacks are offensive maneuvers that target information systems to gain unauthorized access or use of an asset, they are carried out by exploiting one or several vulnerabilities present in a computer system. In an automotive context, they can originate from components inside or outside the CAN bus, taking advantage of several vulnerabilities that exist in CAN, as presented in Section 3.2.1. In this section, we present some of the most notable research on experimental attacks on CAN. Our goal is to provide an overview of the attacks and their effects, and most importantly, to synthesize this information so we can build a threat model. More comprehensive lists on attacks targeting CAN can be found in literature [42, 56, 52]

Koscher et al. [32] presented several methodologies to carry out a variety of attacks. Among these, they showed an attack that exploits the vulnerabilities on the access control mechanisms found in ECUs, allowing an attacker to inject arbitrary code in the ECU memory using diagnostic messages, specifically the DeviceControl service. This allowed them to reprogram the functionality of the unit, which can be used to inject malicious messages into the CAN bus as if they were coming from a legitimate unit. Another attack consisted of injecting carefully crafted data frames which enabled them to control several components of the vehicle. The authors demonstrated it was possible to display false information on the instrument panel cluster, or displaying bogus speedometer and fuel level values. These crafted frames were also used to control some of the car functionalities, such as: opening the trunk, unlocking the doors, controlling the horns, engaging the brakes, killing the engine.

Hoppe et al. [24] demonstrated that by injecting messages at high rates, they could trick ECUs into accepting malicious input, effectively impersonating actuators. They managed to control windows, turn off warning lights, and fake the state of the airbag control system which had been removed. These attacks worked by continuously flooding the bus with fabricated messages, “drowning” the original signals. Additionally, they managed to take control of a gateway in the network, effectively gaining access to different domains and subnetworks.

Miller and Valasek [43, 42] work is one of the most notable on cyberattacks targeting vehicles, having a comprehensive assessment of vulnerabilities on real cars and documented examples of attack tests. Their experiments were carried out on real vehicles, where they were able to tamper the steering, braking, acceleration, and other functionalities. The attacks were carried out by directly fabricating specific frames using a compromised ECU. They also demonstrated a DoS attack that floods the bus with high priority messages, which causes subsystems to malfunction. In a later work [44], they found additional

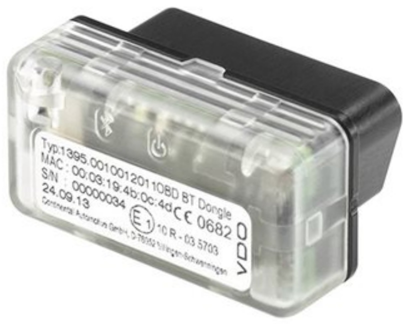


Figure 3.1: OBD Bluetooth Dongle by Continental used to transmit telematics data (available at: `https://www.continental-automotive.com/en-gl/Trucks-Buses/Interior-Cabin/Tachographs,-Telematics-Services/Tolling-Telematic-Solutions/Telematics/OBD-BT-Dongle`)

vulnerabilities that could be exploited remotely, one of the most critical included opening ports over a cellular connection through an authenticated port, and with it gain access to the CAN bus. The authors were able to take full control of a Jeep Cherokee wirelessly, allowing them to shut down the engine of a moving vehicle [22]. They also demonstrated an attack targeting the collision prevention system, where they were able to engage the brakes by impersonating the system without the need of injecting malicious messages at high rates.

Foster et al. [19] demonstrated how aftermarket telematics components can be exploited remotely by taking advantage of several vulnerabilities found in them. They studied Telematic Control Units (TCUs) with Bluetooth or cellular interfaces, which are widely available. These units are directly connected to the OBD-II port and give drivers and insurance companies data on how the vehicle is operating. Figure 3.1 illustrates an example of a TCU typically found on the market. Since these units are connected to OBD they have direct access to the CAN bus. Their vulnerabilities can be exploited locally and used externally, i.e. start with physical access which then could launch a webserver with open ports and telnet. It was verified that all units sold by a specific manufacturer used the same SSH key, meaning that an attacker could buy a unit, manually retrieve the key, and use it to gain access to other TCUs.

Cho and Shin [10] and Palanca et al. [51] took advantage of the error handling mechanism on the CAN protocol to compromise the availability of a target ECU, and even the whole bus. This approach uses the bit monitoring checks to deceive the target unit that there was an error with the message it was trying to transmit. This causes a chain of errors that eventually drive the target into bus-off mode, shutting them off the bus.

3.2.3 *Modeling attacks*

The attacks on the CAN bus presented in literature can be synthesized according to the used methodology:

- **Packet Sniffing**. Since CAN is a broadcast protocol, any ECU connected to the network can read all traffic passing through the bus. Sniffing can be used to either forward data to a malicious agent, learn the behavior of a target ECU, and for *replay* attacks.
- **Forging messages**
  - **Masquerade attack**. A compromised component with access to the network can inject data in the bus impersonating a target ECU. Attacks of this kind usually inject vast amounts of messages to fool its target into accepting malicious frames, instead of the legitimate data. They can also be executed following the same injection rates as the target, but usually require the target to be shut off from bus communications. These attacks are usually referred to as *masquerade* attacks [11]. Which approach to use depends on the goal of the attack, and also the requirements needed to successfully change the state of a target ECU, but it almost always requires revere-engineering the target unit.
  - **Replay attack**. By sniffing previously transmitted messages, an attacker can collect data and retransmit it on the bus at a later time. The purpose of a replay attack is to reenact a previously seen state by using the sniffed data.
  - **Fuzzing**. Reverse-engineering CAN messages is a necessary step to achieve specific changes in the vehicle state, e.g. forging odometer data. On the other hand, fuzzing consists of injecting random or partially random messages, and it can be used when reverse-engineering is not possible. Fuzzing has proven to be an effective tool to cause significant damage in the vehicle [32]. Fuzzing is a term used in contexts other than malicious use, however, we use this term in order to have a consistent terminology with the methodologies presented by Koscher et al. in [32].
- **DoS**. The network as a whole, or its individual components, can be targeted to compromise their availability. A DoS attack consists of flooding the bus with high priority frames is one of the most commonly used tools to shut down the CAN bus. It is also possible to target individual units and shut them off the network, as presented in [51].

An attacker could use one or more attack methodologies depending on their objective. For example, with *replay* attacks an attacker needs to first obtain examples of messages using sniffing, and using this data they can start re transmitting messages impersonating its victim. Additionally, for the attack to work, an attacker may need to shut down the unit it is trying to impersonate.

3.2.4 *Defining the threat model*

Databases with real anomalous CAN traffic are not publicly available, so in order to test any anomaly detection system we must simulate anomalies by modifying the CAN data messages we have collected. We define the threat model in this work based on the reviewed literature, and for consistency and ease of comparing results we will use a similar notation to Taylor et al. work [67, 66].

Most attacks presented in literature, as seen in Section 3.2.2, consist of altering the normal traffic data flow, that is, the expected progression of data frames in a given time-frame. The most common example of such attacks is injecting specific messages at high rates to alter the vehicle state, also referred to as *flooding*. Specifically, we consider the scenario where a compromised and a legitimate ECU send competing messages on the bus, thus creating a stream of interleaved messages. This scenario will be referred to as an *interleave* anomaly.

We consider the case where some messages are eliminated from a legitimate CAN data sequence, thus creating a gap or discontinuity in the normal data flow. This simulates the scenario where a target unit gets switched off and then an impersonator starts transmitting new but still valid sequences. This scenario is referred to as a *discontinuity* anomaly.

These two anomaly scenarios would cause an abnormality at the rate in which messages appear in the CAN bus, assuming these are periodic. Therefore they could still potentially be more efficiently detected by frequency-based methods, but we consider them for completeness as they effectively alter the normal flow of a data sequence.

3.2.4.1 *Data fields content modifications.*

Our threat model also includes cases where data frame contents are directly modified, however, we do not target the whole data frame but instead specific subsets of bits, these subsets are referred to as *fields* as we described in Section 2.4. We identify these fields using the field identification algorithm presented in Section 2.4.1. In the field modification anomaly scenario we employ a fuzzing-like approach, we evaluate fuzzing in our threat model following a similar structure and parameters as in [66]. We refer to these anomaly tests as *data field anomalies*. During experimentation, we consider three parameters to simulate these anomalies: how long the anomaly lasts, which kind of field it is targeted based on its variability, and how the target fields are modified. Regarding how fields are modified we consider: setting its value to its maximum/minimum value, a random value, a constant value, or a replayed value from another time in the sequence.

The alteration of constant bits in the sequence may signal a hidden command, however we have not included the case where constant bits are flipped since finding such anomalies is a trivial task which does not require the use of a neural network-based detection system. By not considering constant bits in the detector we present in this thesis, we can effectively reduce the data dimensionality of our problem, additionally, the performance of our model will not be obfuscated by the learning of constant data.

3.2.5 *Summary of the anomalies*

In summary, we focus on attacks that alter the normal flow of data sequences and also the contents within, hence we define our threat model using the following four anomalous scenarios:

- *Interleave* anomalies. Two legitimate sequences from different time instances are interleaved together, creating inconsistencies in the data stream. For example. given two sequences from different time instances t, and j
  $s_t = [x_t, x_{t+1}, x_{t+2}, \dots]$, and $s_j = [x_j, x_{j+1}, x_{j+2}, \dots]$
  An interleaved sequence is built as:

  $$s_{interleaved} = [x_t, x_j, x_{t+1}, x_{j+1}, x_{t+2}, x_{j+2}, \dots]$$

- *Discontinuity* anomalies. Symbols are removed around the middle of a legitimate sequence, creating a gap/discontinuity. E.g. if 2 symbols are eliminated from the sequence $s = [x_0, x_1, x_2, x_3,$ the resulting discontinuity sequence is built as:

  $$s_{discontinuity} = [x_0,\ x_3]$$

  In our experiments we use a gap consisting of 10 symbols.

- *Data field* anomalies. These anomalies consist of altering a specific target field in a CAN data frame. Such alteration is parameterized by a modifier function, its duration, and the category of the target field. We define 5 modifier functions: setting the field to its maximum, minimum, a constant, a random value, or using a replayed field (i.e. a field value previously found in the dataset).

- *Reverse* anomalies. The order of symbols in a sequence is reversed. This particular anomaly does not match any known attack and it is only used for control purposes. For example, a sequence $s = [x_0, x_1, x_2, x_3]$ is modified to
  $s_{reversed} = [x_3, x_2, x_1, x_0]$.

3.3 STATE OF THE ART AND RELATED WORKS ON DETECTION SYSTEMS FOR CAN

In this section we introduce the existing work on anomaly detection schemes for automotive networks. Systems used to detect potential attacks can be divided into two main categories depending on the approach they follow: signature-based and anomaly-based detection [2]. Signature-based detection uses a database of known attack characteristics and analyses ongoing traffic looking for messages that match these signatures. However, such a database would be manufacturer-dependent, and additionally, databases with real attack examples on CAN that are not part of laboratory tests are not publicly available. Anomaly-based detection on the other hand does not require attack signatures examples to function, this approach creates a baseline using exclusively normal traffic. During the operation of the detector, the messages going through the network are compared with this baseline so that deviations from it are considered anomalous and may signal an attack. Anomaly detection is the most commonly used approach in automotive security research, given the unavailability of attack signatures databases. From a machine learning perspective, anomaly-based detection can be considered as an unsupervised or semi-supervised approach. Unsupervised when no attack samples are used, and semi-supervised when it is possible to use some simulated attack examples to fine-tune the detectors.

In this section we present an overview of the most commonly used patterns for detection, and present the current state of the art on detection systems for CAN.

3.3.1 *Patterns used for detection*

As the first research papers on vulnerabilities and attacks in automotive were published, so did the work on how to mitigate them. Hoppe et al. [24] showcased examples of security threats in CAN and were among the first to propose countermeasures to mitigate these threats. The authors identified four attack scenarios to analyze their safety risks: unauthorized actuation, spoofing signals, impersonation of other ECUs, and unauthorized network bridging. From these scenarios, the authors identified three patterns that can be used to detect potential anomalies:

1. *Increased message frequency.* The first two scenarios require the attacker to send enough traffic into the bus to “drown” the original signal, this means sending more messages than the legitimate ones to force the target unit to accept the malicious data as input. This increased frequency of contradicting messages can be used as an indication of an attack, however, it is only applicable when these are sent periodically.

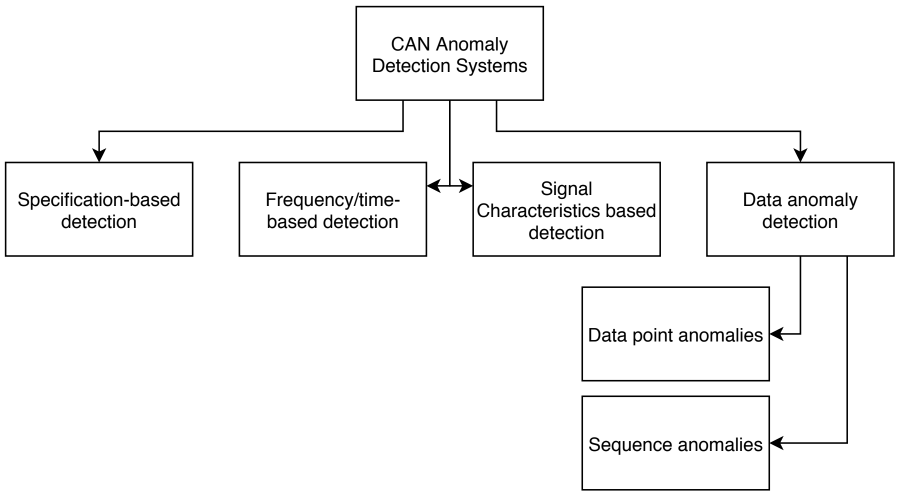

CAN Anomaly Detection Systems
Specification-based detection
Frequency/time-based detection
Signal Characteristics based detection
Data anomaly detection
Data point anomalies
Sequence anomalies

Figure 3.2: Categories of anomaly detection systems for CAN.

2. *CAN ID misuse.* Because of the broadcast nature of CAN, every ECU can monitor if other units send frames using one of its exclusive IDs. Therefore, each unit can theoretically detect messages that are trying to impersonate it.

3. *Low level communication characteristics.* Use data from the physical layer (i.e. the electrical signal generated by the transceiver) to detect anomalies from an arbitrary bus location. Used to validate if a message comes from the expected source, acting as a pseudo-authentication mechanism.

Based on these scenarios and attack patterns, many researchers have suggested detection systems that attempt to generalize what constitutes normal and anomalous traffic. Results are promising, however, not one single method or system can be used to detect all possible anomalies since they manifest in different ways. Therefore, a complete intrusion detection system must employ several techniques to successfully identify attacks.

Next, we present the most notable research on detection systems for CAN. These proposals can be divided into four categories depending on the attack patterns they analyze, these are: *Specification-based*, *frequency/time-based*, *signal characteristics-based*, and *data* anomaly detection. Additionally, data anomaly detection is divided into two subcategories, data point detection which analyzes single data points, and sequence detection which analyzes a time-series of data observations. We illustrate these categories in Figure 3.2.

3.3.2 *Specification-based detection.*

With this approach, legitimate CAN traffic is specified by a set of explicitly defined rules. This holistic scheme checks for correctness, compliance, formality, consistency, and plausibility of the contents of every message. From a database, the system retrieves what are the allowed ranges, variations, domains, frequencies, etc. for a given CAN ID.

Müter et al. [48] define this specification-compliance verification process through the use of a set of sensors. These check for formality, location, range, frequency, correlation, protocol compliance, plausibility, and consistency. The results from these sensors are integrated to produce an estimation of a potential anomaly using weighted values to indicate criticality.

Salman and Bresch [59] proposed a detection system that uses a signal database (i.e. the `dbc` file) to check for compliance, and sets a threshold that determines how many abnormal messages are allowed before triggering an attack alarm. This database specifies a set of allowed behavior for each ID, for example: in which domains it operates, minimum and maximum values, allowed change rates in data, message frequency, etc.

Dagan and Wool proposed Parrot [13], an anti-spoofing system which would be installed on every ECU. It works by monitoring the network for ID misuse by other units, and if needed, it creates collisions as a counter attack to take the offending unit offline. Naturally, it requires knowledge as to which units can in fact be taken offline and not cause further damage.

However, many attacks presented in literature follow the follow the behavioral specification for a given CAN ID, several of the attacks presented in literature use the allowed ranges and expected message injection rates. A specific example that matches this description is a *replay* attack. In practice, the databases containing the specifications of CAN signals are rarely available since they are kept confidential, they are also different across models and manufacturers. All other research made on this topic assumes this database is unavailable and therefore it is not possible to know the original signal semantics.

3.3.3 *Frequency/time-based detection.*

Several attacks presented in literature consist of an attacker injecting malicious messages at high rates. Researchers have proposed detection systems that record how often messages are seen on the bus, deriving an estimated frequency for each ID, which is then analyzed for anomalous variations. A crucial assumption in this scenario is that the monitored IDs sends messages periodically with a somewhat constant frequency. Outliers in the frequency distribution are not anomalous by nature, a common scenario where this occurs is when packets lose arbitration to higher priority traffic, which is considered normal. Because of this, IDSs that use this approach must account for occasional and legitimate outliers.

Here we present some of the frequency/timing-based detection systems found in literature, even though similar methods were discussed in previous research [43], these works include experimental tests of their detection systems.

Taylor et al. [68] proposed a detection mechanism using two methods: training a One Class Support Vector Machine (OCSVM) using statistical data on message frequency as features, and computing the frequency for every ID using a sliding window and then getting an anomaly score based on computing a t-test over the historical data. The two presented methods performed significantly well, achieving perfect performance in many tests. Since they are computationally inexpensive, and no method was better in all test cases, the authors suggest using a mix both for more reliable detection.

Song et al. [61] proposed a lightweight IDS that analyses time intervals between messages. First, the system estimates these intervals, and in the detection phase, an alarm is triggered when an interval is shorter than half of what it normally is. The system then creates a score based on how many consecutive messages have abnormal arrival times, effectively reducing false positives. This part is an important aspect of the system, as it can handle real variations that occur like lost arbitrations. A significant advantage of this approach is that It requires only a small number of samples and low computational demands, it achieves almost perfect performance with the proposed threat model, achieving up to 100% accuracy with no false positives.

Cho and Shin [11] proposed CIDS, an IDS that creates fingerprints for every ECU based on their clock skews. The main assumption of this work is that clock quartz crystal oscillations in each ECU create unique clock skews. This ADS aims to detect messages from malicious units that attempt to impersonate other units, regardless of their injection rate. However, in practice, it has been proven that this detection system can by bypassed. The clock offsets are computed based on the messages arrival times to the detection unit, rather than solely on hardware characteristics, and so they can potentially be spoofed as shown by Sagong et al. [57].

Lee et al. [33] proposed OTIDS, a detector that periodically sends remote frames to all ECUs in the network and detects anomalies based on the time they take to send a response to these frames. This response time is theorized to be a good fingerprinting tool, as it depends on hardware characteristics that are assumed unique for each unit. However, the actual system only evaluates the time at which the required frames arrive at the detector. During set-up, this system learns the distribution of response times for every transmitting unit, to then create the baseline that the detector uses to look for anomalous response times.

Even though these proposals can find anomalies with fairly good performance, they can only detect attacks that require the use of high injection rates. Therefore, malicious changes in the data frames using seemingly normal injection frequencies (as presented in [44]) would not be detected.

3.3.4 *Signal characteristics-based detection.*

Some detection systems make use of the signal characteristics of CAN transceivers to create fingerprints of each unit in the network. These characteristics are theorized to be unique for each ECU and could be used to effectively identify them. Fingerprints are built by analyzing the physical changes in the bus for each ID that is transmitted. These systems aim to detect intrusions in cases where a message is sent from a unit with a different fingerprint than the expected one; this aims to detect attacks where an attacker impersonates a given unit but manages to mimic other aspects of its behavior.

Murvay and Groza [46], Choi et al. [12], and Spicer [63] proposed systems that derive the signal characteristics of a message by measuring the CAN-H and CAN-L voltage values during transmission, extracting several features from the time and frequency domains, which are then used to train classifiers. This approach is used to derive a fingerprint for each ECU, so when an impersonation occurs, the anomalous messages would show characteristics that are different from the expected fingerprint. However, the approach presented in these works follow a batch learning-based approach which assumes that the characteristics do not change during the operation of the vehicle; this is not the case in automotive environments since several factors, like changes in temperature, can alter these signals.

Cho and Shin [9] proposed an approach that measures the voltage use of each unit in the network, these measurements are used to create voltage profiles for every transmitting ECU. However, its goal is not intrusion or anomaly detection, but rather work as a tool for attacker identification. One of its most important features is adaptability, this system uses an adaptive signal processing approach so that the system can account for environmental or operational changes that affect the voltage profiles of every unit in the vehicle.

Palanca et al. [51] proposed a mechanism that aims to detect when a new ECU is connected into the network. It measures differential internal resistance of all the units, which influences the total bus load that a transmitting units uses to correctly send bits. When a unit is added, this load changes, thus detection works by measuring changes in the current necessary for transmission. This was proposed as part of their work presenting a novel attack in CAN networks, it was presented as a possible countermeasure so no real test were conducted; we included this work since it could be implemented as part of a complete security solution.

3.3.5 *Data anomaly detection.*

Some of the attacks in literature only modify the payload contents of CAN messages, using the same frequency as a target ID and following the protocol specification. For instance, Cho and Shin [11] demonstrated a scenario where a target ECU is taken offline and then a compromised unit sends malicious messages impersonating the target. In this scenario. Frequency-based detection systems would not be able to detect the attack, in consequence, an approach that analyses data contents is needed.

Some researchers have proposed solutions to find anomalies in the data frame contents. Müter and Asaj [47] and Marchetti et al. [39] proposed an information theory approach, where the packet message entropy is used to construct a norm model. This strategy showed promising results, but it was only tested in insertion attack scenarios, e.g. flooding, DoS, and targeted injection. However, these systems only compare the entropy of single messages and not sequences, therefore it can only detect single point outliers.

Kang and Kang [29] proposed a probability-based IDS that uses a deep neural network model that captures the statistical features of CAN traffic. First, the system computes the probability of each given symbol, then p is defined as the result of applying a logistic regression on such probabilities. Then, a feature vector is built by applying a xor operation between two consecutive time instances, that is $v(n) = p_v(n) \oplus p_v(n-1)$. The neural network is trained using these feature vectors and using a supervised approach, using samples of both normal and anomalous traffic. The model learns to estimate a certainty metric that a given feature vector is anomalous or not. However, it is difficult to evaluate the use of this approach for several reasons, the experimentation was fairly limited and omitted important details. The nature of the messages was also very limited since it only considered subsets of the data frame bits. Additionally, the threat model used was not specified so it's impossible to know under which anomaly scenarios the IDS was tested on.

3.3.6 *Data sequence anomaly detection.*

CAN traffic data can be seen as a time-series, with its own natural temporal ordering, meaning that analyzing anomalies in CAN traffic, is a time-series anomaly detection task. One of the main advantages of this approach is that it aims to detect abnormalities in messages that can be considered normal as individual observations, but that are actually anomalous in the *context* of surrounding messages. It can detect values that are outside a norm model (outliers) and also when they are out of context; for example sudden unrealistic changes in speed but otherwise normal absolute speed values, e.g. 1km/h to 100km/h in a second. Another advantage of this approach is that it can be used regardless of the periodicity of the CAN ID to be monitored.

Detecting anomalies in time-series is a well-researched topic, detection can be done using several techniques including finite-state automata, information theory, proximity by mapping the series onto a space, or creating models that aim to reproduce the time-series [1]. Cluster analysis has also been proposed to analyze time-series subsequences, however, research has shown this approach to be inadequate yielding poor results [30]. Choosing a detection approach greatly depends on how many unique data symbols are produced. For a limited number, approaches like Hidden Markov Models (HMMs) [28, 21], or Conditional Random Fields (CRF) [62] could be appropriate solutions.

Regarding time-series anomaly detection on automotive networks, Narayanan et al. [50], and Levi et al. [34], presented ADSs that use classifiers trained using HMMs. They analyze sequences of events for anomalous behavior, transforming time-series data into state changes with transition probabilities. Levi et al. used the absolute value of known variables, specifically rpm and speed to build the HMM. Narayanan et al. on the other hand, created a new layer of abstraction of the data by defining *stories*, which are general descriptions of events, examples of stories are: *car unlock, door open, door close, seat belt on*. However, as we demonstrated in Section 2.3, the number of symbols that CAN IDs can produce is not necessarily restricted to a fixed dictionary, rendering HMMs and CRFs inadequate in most cases.

In time-series analysis, RNNs have been shown to be effective modelers. They are considered to be a natural fit when dealing with multivariate sequences, with no natural restrictions on the symbols' dictionary size, and even allowing the use of arbitrary sequence lengths when using specialized units. Taylor et al. [67] proposed a system that trains models as predictors of CAN traffic data, then it classifies unexpected predictions as a potential anomalies. This model works by applying a semi-supervised learning approach, the ADS uses LSTM neural networks to learn the normal sequences in the data, these models are trained to predict data symbols given a set of normal sequences. An anomaly score is computed based on the difference between the real and predicted data frame, i.e. the prediction error, and it is used as the anomaly detection metric. In their later work [66], the authors also compared the predictor with a multi-step mul-

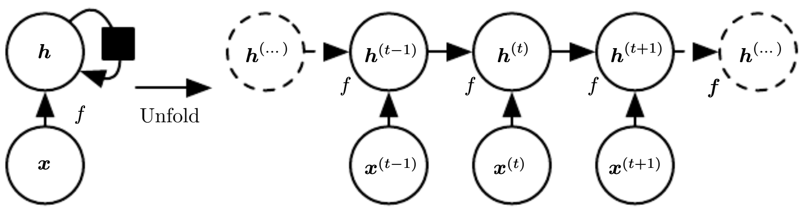


Figure 3.3: Unfolding a RNN. Figure taken from [20].

tivariate HMM, which is adapted to handle high dimensional data. However, the performance of the new HMM was significantly worse than the LSTM predictor. Among the conclusions drawn from their work, it was noted that the detector struggled with several IDs with low word-variability, as well as, short-lasting anomalies, leaving room for improvement and future research.

In the following sections we introduce RNNs and LSTM, so we can introduce the concept of multivariate time-series anomaly detection, which is the main framework of this thesis.

3.4 RECURRENT NEURAL NETWORKS

Neural Networks are statistical models that aim to approximate a nonlinear function, however, classical neural networks cannot process sequential data. Recurrent Neural Networks (RNNs) on the other hand, specialize in handling data sequences and they are powerful modelers even at arbitrary sequence lengths [20]. Sequences are generalized by sharing the parameters of the network through sequences of nodes, which translates into sharing them across time. These networks use the notion of recurrence to represent a sequence or chain of events. This can be represented as the unfolding a computational graph, where information from an input series x is processed by incorporating the hidden state $h^{(t)}$ to the input $x^{(t)}$ at time t, and passed forward to $t+1$. This process is illustrated in Figure 3.3. RNNs are akin to finite state automata, and they do not make a Markovian assumption [1] (a property of a stochastic process that states that the future state of a process depends only upon the current state and not past events). Additionally, they do not demand dimensionality reduction of the input, rendering them a natural fit for multivariate data.

We proceed to present an overview of how these networks can represent sequences of data. Given the input $x^{(t)}$, the state $h^{(t)}$ of a recurrent unit is computed as:

$$h^{(t)} = f(Wx^{(t)} + Uh^{(t-1)} + b)$$

where f is a nonlinear activation function (e.g. sigmoid hyperbolic tangent, etc.) , W and U are weight matrices, and b is a bias term that exist in each unit. The output for each cell is computed as:

$$o^{(t)} = W_{ho}h^{(t)} + b_o$$

where $o^{(t)}$ at time t depends on the current state $h^{(t)}$, a weight matrix W_{ho} connecting input and output with a separate bias term for the output b_o. All these computations are repeated over a sequence over time t so that the RNN learns to generalize the sequence. Even though RNNs can learn arbitrary time lengths, the input must always have the same size as this determines the structure of the neurons in the network.

Similarly to classical neural networks, RNNs require the definition of a loss function to compute the gradients during training. However, classical back-propagation is not adequate anymore because of the flow of the gradient does not only go through neurons, but also time. To solve the issue of computing the gradients through time, the back-propagation algorithm is applied to the unrolled computational graph, in a process called Back-Propagation Through Time (BPTT).

This makes RNNs quite difficult to train in practice. This difficulty is due to the more complex units, the fact that the network now has to optimize significantly more parameters and so forward-propagation requires computing significantly more outputs/states, and BPTT is a complex operation on its own. These networks have another important disadvantage, even though they theoretically can learn long-term dependencies, this is not the case in reality because of two problems known as *vanishing* and *exploding* gradients [53]. During back-propagation, the feedback loops can cause the gradients to shrink or grow significantly.

3.4.1 *Long Short-Term Memory*

Hochreiter and Schmidhuber [23] presented Long Short-Term Memory (LSTM) neural networks, a solution to the gradient problems present in RNNs. This solution adds forget-gates and a complex, but powerful, state flow that allows the network to actually learn long-term dependencies. The most crucial contribution of LSTMs is the presence of a linear self-loop that gradients can flow through without exploding or vanishing [1]. The state computation is significantly more complex than in classic RNNs, however, they achieve great improvements in performance.

Now we describe the equations used in he calculation flow of an LSTM, this process is also illustrated in Figure 3.5.

Input and forget gates (i and f respectively) are crucial components of LSTMs, they control how much of the hidden space is dependent on both the new input and the state at $t-1$. They depend on the

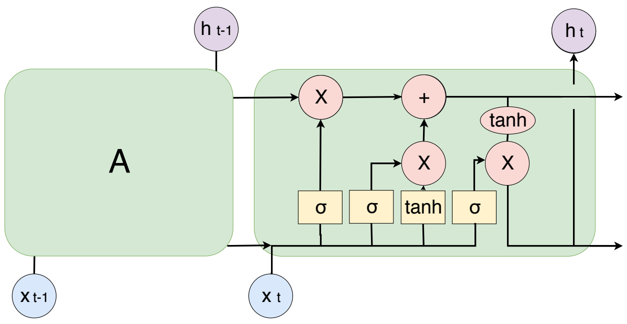


Figure 3.4: Structure of a LSTM in a network.

input $x^{(t)}$ and the output sate from the previous step. The signal of these gates is determined by the equations:

$$i^{(t)} = \sigma(W_{input}x^{(t)} + U_{input}h^{(t-1)} + b_{input})$$
$$f^{(t)} = \sigma(W_{forget}x^{(t)} + U_{forget}h^{(t-1)} + b_{forget})$$

W_{input} , U_{input} and W_{forget} , W_{forget} are weight matrices for the input and forget gates, while b_{input}, b_{forget} are their respective bias terms. σ corresponds to the sigmoid function.

A linear combination of the outputs of i and f is used to determine the new state of the cell $c^{(t)}$ using the previous state $c^{(t-1)}$ and a candidate state $g^{(t)}$. The new state is computed as:

$$g^{(t)} = \tanh(W_c x^{(t)} + U_c h^{(t-1)} + b_c)$$
$$c^{(t)} = f^{(t)}c^{(t-1)} + ig^{(t)}$$

Where W_c, U_c are the weight matrices of the c. Now the cell output is controlled by an output gate $o^{(i)}$:

$$o^{(t)} = \sigma(W_o x^{(t)} + U_o h^{(t-1)}) + b_o$$

Where W_o, U_o are the weight matrices of the output gate, and b_o its bias. Finally we can compute the output state $h^{(i)}$:

$$h^{(t)} = o^{(t)} \tanh(c^{(t)})$$

3.4.2 *Autoencoders*

An autoencoder is a type of neural network that learns a latent representation of training data, this results in a fixed-sized representation that usually has a smaller dimension than the input. Autoencoders consist of two main components, an *encoder* and a *decoder*. In brief, the input is described by a hidden state h, the encoder acts as a function so that $h = f(x)$. The decoder then reconstructs the input from the hidden state $r = g(h)$, the general structure of an autoencoder is

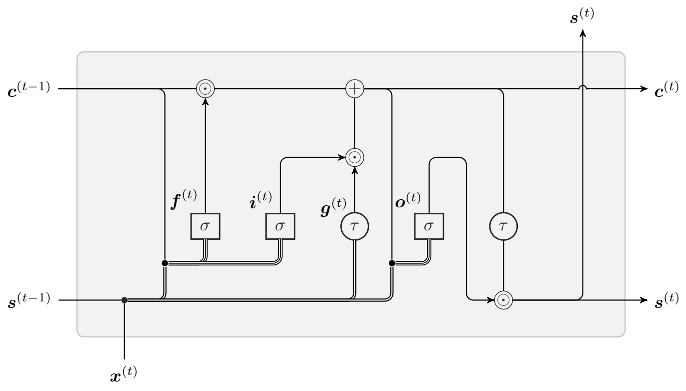


Figure 3.5: LSTM calculation flow, not including bias terms. Taken from [1]

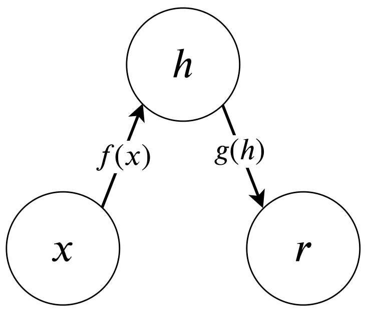


Figure 3.6: General structure of an autoencoder. An input x is represented as a space h through function $f(x)$, and then it is reconstructed to output r using function $g(h)$.

illustrated on Figure 3.6. Autoencoders have been traditionally used for feature learning or dimensionality reduction [20, 3]. Denoising autoencoders for example, are commonly used to remove noise that is corrupting an input, it receives corrupted data and is trained to predict the original data.

Autoencoders are usually set to have a single layer for the encoder and one for the decoder. Nonetheless, similarly to feed-forward networks, autoencoders with deep architectures can offer greater representational power. Additionally, depth can significantly decrease the amount of training data that is needed to learn some functions [20].

Autoencoders can also be used to learn latent representations of sequential data using recurrent units in the encoder/decoder layers, where the encoder maps an input sequence into a vector representation. In this scenario the hidden state h is obtained from the last step in the encoder. The dimensionality of this latent representation is determined by the number of hidden units in the recurrent network. In natural language processing, sequence-to-sequence models can be implemented using autoencoders.

3.5 RNNS IN MULTIVARIATE TIME-SERIES ANOMALY DETECTION

Complex control systems that record measurements from different sensors create streams of multivariate data, in other words, a time-series composed of more than one dependent variable. Systems with multiple inputs and outputs are often difficult to model, fewer variables make for an easier modeling task but it is likely that their accuracy will worsen. Industrial systems, aircrafts, and automobiles are some examples of systems that create multivariate time-series because of the vast amount of sensors included within them. In the case of automotive CAN, not only each ECU produces unique streams of data packets, but also each of them can have up to 64 variables (i.e. the number of bits in the data field).

Several approaches have been presented for multivariate time-series anomaly detection applications, these include graph-based models [54], energy-based methods like restricted Boltzmann machines [17, 72, 14], and recurrent neural network models. In this context, RNN models have been shown to be effective time-series modelers.

There are two main approaches when performing anomaly detection tasks using RNNs: *predictor* and *reconstruction*-based detectors.

3.5.1 *Prediction-based detection*

A predictor-based detection approach [37] trains a recurrent neural network model to predict future observations using normal (i.e. non-anomalous) time-series sequences. The model consists of stacked layers of recurrent units (either LSTMs, or Gated Recurrent Units (GRUs)) and it is trained to predict either a single observation or a whole sequence. This training is carried out with the goal of reducing the prediction error over a separate set of validation data. A simplified representation of the stacked LSTM architecture used for anomaly detection is presented in Figure 3.7. During operation, this type of detector takes as an input a time-series and tries to predict its next observation. The difference between the prediction and actual value is computed as the prediction error which is used as an anomaly metric, high prediction errors are likely to indicate an anomaly in the data sequence.

This approach has been applied in anomaly detection in a wide variety of contexts, including but not limited to electrocardiography data [7], industrial control systems [15, 16], aircraft and spacecraft data [49, 25], video [36], and also automotive CAN [67] as we presented in Section 3.3.6. Neural networks with dilated convolutions have been proposed as an alternative to recurrent neural networks in prediction tasks, they are significantly easier to train with similar or better prediction performance in some scenarios [5], this solution was inspired by the generative approach used in Wavenet [70].

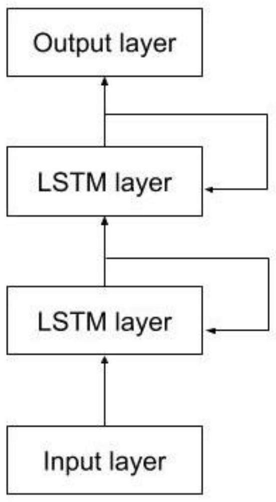


Figure 3.7: Simplified architecture of a stacked recurrent neural network predictor with LSTM units.

3.5.2 *Reconstruction-based detection*

The second approach to anomaly detection is reconstruction-based detection, it consists of training a RNN for representation learning, typically using an autoencoder. A reconstruction consists of recreating a sequence based on the characteristics of the signal used to train the model. This model characterizes the data sequences in a compressed representation so they can be later reconstructed, the reconstruction error is used as a metric to differentiate anomalies [58]. Reconstruction models, specifically LSTM autoencoders, have also been proposed to model these sequences, Malhotra et al. [38] showed that an encoder-decoder approach for multi-sensor anomaly detection is more efficient than predictor-based models at detecting anomalies when the underlying data is inherently unpredictable. They carried our their experiments to test this hypothesis under different datasets, with both predicable and unpredictable time-series data. A similar experiment showing supporting results to this hypothesis was also presented in [1]. This reconstruction approach using autoencoders has been shown to be effective in different time-series anomaly detection settings, including industrial machinery [3], acoustic data [40], video sequence analysis [58].

4 APPROACH: PREDICTOR AND AUTOENCODER DETECTORS

4.1 INTRODUCTION

In this chapter, we describe the approach of our work. First, we present an overview framing our problem and explain the motivation behind our choice to use the LSTM autoencoder approach as a solution. We then define the performance scores that we later use as a metric to compare how the predictor and autoencoder perform under the same experimental framework. Then, we present the approach used in the state-of-the-art predictor-based ADS for CAN. Finally, we describe our proposal of an autoencoder-based approach for anomaly detection on CAN, we define the components and architecture of the detector, and its method to use a reconstruction approach to find anomalies.

Both detectors are described in a way that allows for experimental comparison of these two approaches.

4.2 APPROACH OVERVIEW

The approach we propose is a data sequence anomaly detection system for CAN data. This detection problem has several restrictions that limit the solutions that can be applied. An important limitation comes from the number of unique symbols that can exist in CAN traffic data, since most IDs in the high-speed bus create new symbols at a seemingly growing rate, rendering approaches that assume a fixed-size dictionary inadequate. Another issue with HMM and similar approaches is that they make the assumption that the data generating mechanism behind the time-series can be modeled through a Markovian process, which is a strong assumption given our domain-unaware scenario. To our knowledge, the predictor-based ADS proposed by Taylor et al. [67] is the only work done on data sequence anomaly detection on CAN that deals with effectively infinite unique symbols. From their results and conclusions, we observe that performance varies significantly between IDs, although, not knowing the signal semantics hinder a complete interpretation on what factors influence these results. However, some insights are obtained from the parameters used in the anomaly tests. For instance, the test cases with low performing detection results had some common characteristics, they targeted low variability data and they used short lasting anomalies.

This approach also presents some inherent limitations that come from the assumptions made during its design, one of the most crucial ones being that CAN traffic is thought to be predictable. However, this assumption can be challenged given the semantics-unaware context of the problem. Some approaches on anomaly detection do not rely on the predictability assumption: in Section 3.5.2, we reviewed literature presenting the use of autoencoders for anomaly detection, which can be used as an alternative to prediction-based models. Malhotra et al. [38] presented a reconstruction-based detector using a LSTM encoder-decoder model, which was proposed as an alternative to the LSTM predictor approach by the same authors in [37]. By experimenting on different datasets, this model proved to perform better on unpredictable time-series. For predictable time-series, the predictor-based approach still achieved better performance. In this work we will refer to this encoder-decoder approach simply as an autoencoder, because both the encoder and decoder structure are similar and because they have use the same input and output space, effectively rendering the model as an autoencoder.

We have chosen to apply a reconstruction-based approach using autoencoders to our problem given the promising results in contexts that do not assume the time-series to be stationary, and therefore a series that may be difficult to predict or inherently unpredictable. Further work is needed in order to apply a LSTM autoencoder to our multivariate problem, Malhotra et al. only effectively tested their model using univariate test sets, the only multivariate data on their experiments was reduced to an univariate time-series using principal component analysis.

Our work consists on evaluating the use of a detection system for CAN using LSTM autoencoders and showcase its advantages with respect to a state-of-the-art detector for data sequence anomalies. We first replicate the LSTM prediction-based detector as presented by Malhotra et al. [37] and Taylor et al. [67] and test it with our own dataset. Then, we implement an autoencoder detector that reconstructs sequences of the multivariate time-series by exclusively learning normal traffic data. Since one of our main goals is to compare the two approaches, we will use similar parameters in both our implementation and experiments that we present in Chapter 5.

Our approach can be summarized into the following steps:

1. Define the performance scores. First we define how we evaluate the performance of an anomaly detector, the metrics to do so are defined in Section 4.3.

2. Replicate the predictor-based detector for CAN. To do so, several further steps must be followed, we describe this process in Section 4.4.

3. Evaluate the use of autoencoders for data sequence anomaly detection on CAN. This process is described in Section 4.5.

4. Finally, test the predictor and autoencoder detectors under the same anomaly scenarios and compare their performance. This step will give us the results needed to effectively compare both approaches, and is explained in Section 4.6.

4.3 PERFORMANCE SCORES

In this section we define how we measure performance for the methods evaluated in this thesis. Anomaly detection is a classification task and we must evaluate it as such, a CAN data sequence can be considered either as anomalous or normal. Both approaches presented here output a single score value that is used to determine how to classify a sequence. Choosing a decision threshold depends on many factors, in our scenario it is determined by the nature of each unit given its criticality on the network.

Precision, and Recall

To determine the performance metric we use in our classification task, first we have to introduce an evaluation measure from the context of information retrieval. When classifying documents, or in this case CAN traffic sequences, we need to measure the *relevance* of our results. The possible results in a classification task are summarized in Table 4.1:

		True	
		Normal	*Anomalous*
Predicted	*Normal*	True Positive (TP)	False Positive (FP)
	Anomalous	False Negative (FN)	True Negative (TN)

Table 4.1: Definitions of the results in a binary classification task.

From these definitions we can define two performance metrics for classification, the *True Positive Rate* and the *False Positive Rate*, defined in Table 4.2:

True Positive Rate (TPR)	$\frac{TP}{FN+TP}$
False Positive Rate (FPR)	$\frac{FP}{TN+FP}$

Table 4.2: Definitions of true positive and false positive rate, performance metrics for classification tasks.

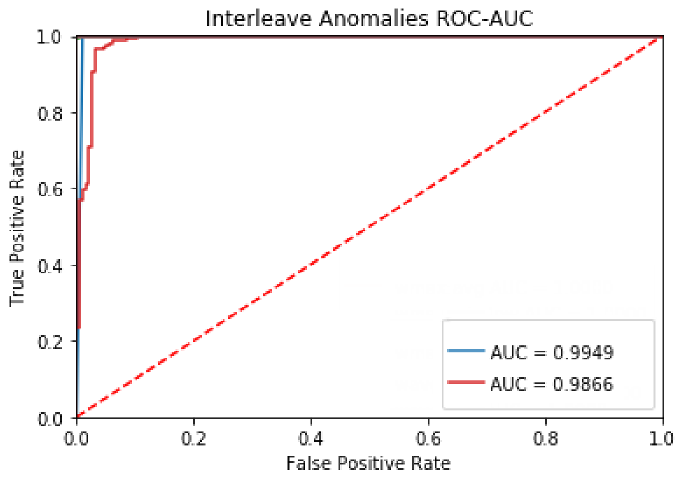


Figure 4.1: Example of a ROC curve showing AUC values for two detectors.

4.3.1 *ROC and AUC*

The *Receiver Operating Characteristic (ROC)* is a widely used graphic to display the TPR and FPR for all possible thresholds, where the overall performance of the classifier is given by the *Area Under the Curve (AUC)* of the ROC [26]. The AUC is the detector performance metric used throughout this work. An ideal classifier would show a ROC curve that sits very closely to the top left corner with AUC values close to 1. A classifier that is no better than random choice would have an AUC of 0.5. Which threshold value to choose depends on the problem domain at hand, whether we want to maximize precision or minimize false positives, we must choose a threshold that meets the domain's needs. We use an example of one of our anomaly tests to illustrate the ROC in Figure 4.1, which shows fairly good results of two classifiers with high AUC values.

It is important to note that this approach to measuring performance is better used in cases where classes are somewhat equally distributed in the test dataset. So in order to use them properly during our experiments, we test the detectors on datasets that are not strongly dominated by normal traffic, keeping a more balanced sequence type distribution.

4.4 PREDICTION-BASED DETECTION

In this work we replicate the prediction-based detector and evaluate its use with our dataset. In our understanding, the predictor system presented by Taylor et al. is the only approach aiming to detect anomalies in CAN data sequences following the limitations we have described in Sections 2.3 and 3.3, specifically dealing with data generating mechanisms that steadily create new symbols that are not limited to fixed-size dictionaries.

The intuition behind this approach is that a RNN can learn to model the characteristics of a time-series of CAN recordings. Since only non-anomalous data is used, the RNN exclusively learns to model what constitutes normal CAN data. Consequently, in the presence of anomalous traffic the RNN would output erratic results from which we can infer the presence of anomalies. The detector consists of two main components, a predictor and an output processing mechanism. The predictor is a neural network that is modeled so it can forecast a time-series based on an input sequence, in our context the model takes as input a sequence of CAN data recordings up to a time t and aims to predict the next CAN data symbol. The detection system then measures how accurate the predictions are, and then it compares the real symbol from the time-series at $t + 1$ with the predictor's output to compute a prediction error value. This error works as an indication to differentiate potential anomalies. These errors are then sent to an output processing module for further analysis.

The output processing module analyses a stream of prediction errors to produce an anomaly signal. The need for this module comes from the fact that predictors will likely never be able to perfectly characterize the data generating mechanism behind the time-series, this can be caused by many factors but most importantly by the presence of noise and stochastic processes that we are unaware of given the domain-unaware nature of our problem. When designing this module two important challenges appear: how to process prediction errors to produce a single error metric (called anomaly signal), and then how do we determine at which values are these metrics considered anomalous. Figure 4.2 illustrates an overview of the predictor-based ADS.

Since each data generating mechanism behind each ID is different, it is necessary to implement a single predictor for each one, in our case resulting in 12 different predictors for each of our 12 IDs.

In summary, to replicate the predictor-based ADS we must follow these general steps:

1. Define the predictor architecture and train the model.
2. Implement the output processing module, which defines how the anomaly signal is generated.

4.4.1 *Model Architecture*

The predictor system proposed by Taylor et al. is heavily inspired by the model proposed in Malhotra et al. work [37]. The authors designed a stacked architecture and prove that by stacking recurrent hidden layers with LSTM units, the resulting model can learn long term patterns of arbitrary length and also learn higher level temporal

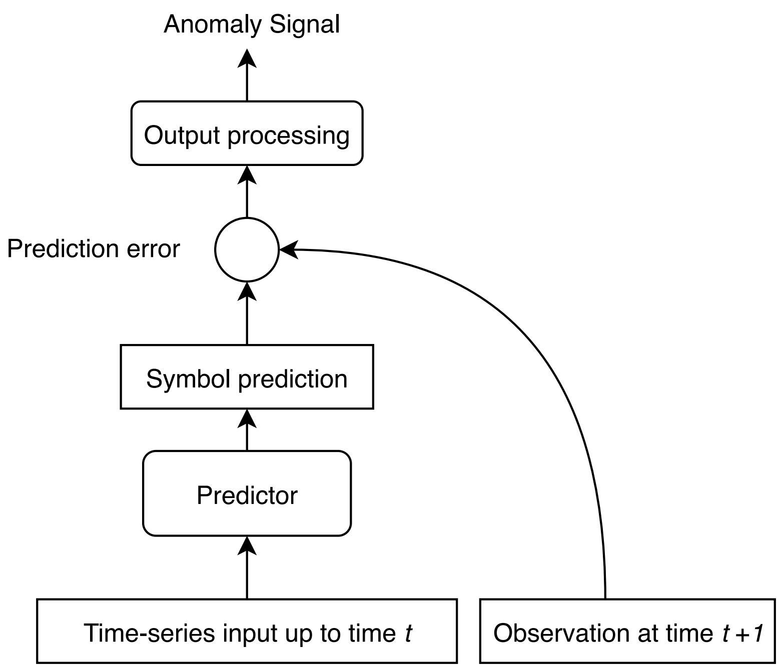


Figure 4.2: Overview of a predictor-based CAN anomaly detection system.

features. The structure of the model is defined as a neural network with four hidden layers, as shown in Figure 4.3. This architecture is based on the one presented in [37] but with a small variation in the hyperparameters obtained by running a random hyperparameter search process.

The predictor model is structured as:

1. An input layer that specifies the input sequence dimensions, equal to the number of non-constant bits for its respective CAN ID.

2. Two fully connected dense feed-forward hidden layers, with 128 units using `tanh` activation functions.

3. Two recurrent layers follow, both using `tanh` activation functions and 512 LSTM units.

4. The output layer outputs the prediction, so it has the same amount of units as the series dimensionality for the given ID, It uses sigmoid activation functions to scale back the prediction to a [0 - 1] range.

Each layer, except for the output one, is followed by batch normalization layer. This layer normalizes the activations of the previous layer at each batch and is said to speed up convergence time.

All layers in the network have a Dropout value of 0.2. Dropout is a tool used when training neural networks that aims to avoid overfitting by dropping some units in the network. Each unit in each layer

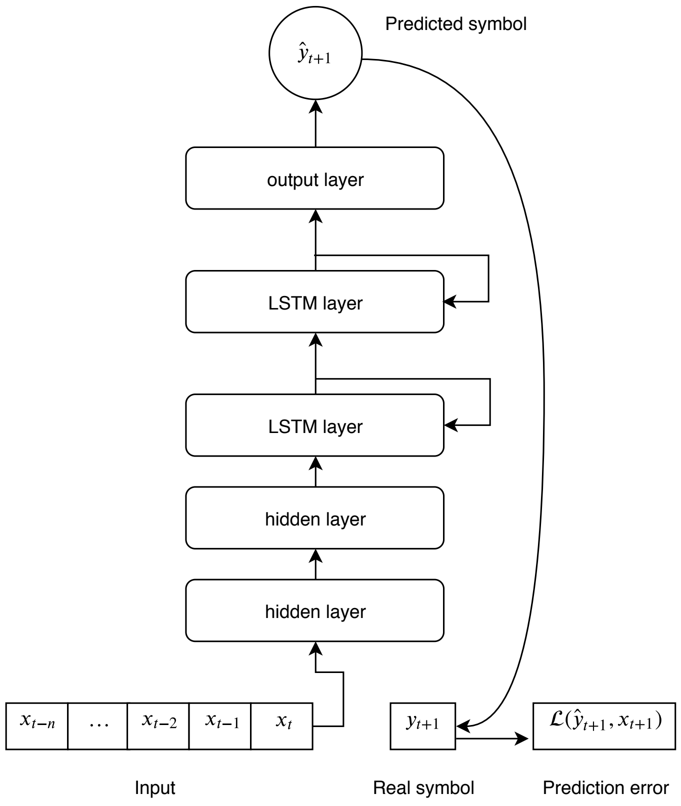


Figure 4.3: Structure of the prediction-based anomaly detector for CAN proposed by Taylor et al.

has a probability D to be taken out the network, D is a hyperparameter.

INPUT. For each time step, the input is given by sequences of CAN data of length n in binary format with dimensionality k. Each input sequence is therefore a $n \times k$ vector. n is a hyperparameter that can be chosen depending on application constraints. However, its value must be tuned in order to improve prediction accuracy. Instinctively the more context (i.e. data frames) are fed into the model, predictions improve. In our experiments we set $n = 20$ to be consistent with the reference work [67]. k represents the dimensionality of our time-series, with a maximum value of 64, corresponding to the 64 bits available per data frame. However, as we evidenced in Section 2.3 all IDs use only a subset of the total amount of available bits, removing constants reduces the dimensionality for each ID and slightly lessens the model's memory usage.

OUTPUT. An output layer is added to bring the prediction back into the domain of the original data. This linear layer uses sigmoid activation functions to scale the values between 0 and 1, and uses a total of k units. The final output is a single symbol prediction $\hat{x}_{i+1}$ with dimensionality k, it can be interpreted as a measure of certainty about the prediction, even though its presentation is similar to a probability it cannot be considered as one. For example, a predicted bit value of 0.9 indicates that the neural network predicts with high confidence that the real bit value is 1. While values around 0.5 indicate that the model is not confident about its prediction.

Training

A single model is trained for every ID, only IDs that present both high frequency and high symbol variability are considered, accounting for 12 CAN IDs as discussed in Section 2.3.

The model itself consists of a deep neural network that is trained to predict a symbol that follows a given sequence. To find the parameters that better achieve this task the network computes an error metric between the predicted symbol and the real one. To train the model for prediction, we begin by defining the sequence $S_i = x_t, x_{t+1}, \ldots, x_n$ where x_t is a single CAN data frame at time t, and n is the sequence length (also called a window). The neural network receives two elements as inputs: a set of sequences and a set of individual symbols y following those sequences so that the network learns what it is meant to predict. Briefly, training inputs consist of a set of S sequences that are continuous in time to preserve data flow continuity, and a set of target symbols, e.g.:

$$S_0 = x_0, \ldots, x_n \quad y_0 = x_{n+1}$$

$$S_1 = x_1, \ldots, x_{n+1} \quad y_1 = x_{n+2}$$

In the spirit of replicating Taylor et al. approach we use the same number of timesteps for the input sequence (i.e. how many symbols per sequence) which we set to $n = 20$.

PARAMETERS. The models are trained with a binary cross entropy loss function using the Adam optimizer, using a batch size of 128. It also includes early stopping as an overfitting countermeasure, specifying that training stops if the network performance does not improve after 3 epochs.

The resulting models are saved on separate files and stored in disk.

USAGE. Once the predictor models have been trained, we have to specify how they are used in the context of analyzing sequences. The resulting models receive as an input a set of sequences S, for each sequence $s_i = \ldots, x_{t-2}, x_{t-1}, x_t,\ s \in S$ a symbol prediction is the output $\hat{y}_{t+1}$ which is then compared to the real value x_{t+1}.

4.4.2 *Output Processing and anomaly signal*

With the model trained, we can use data sequences to obtain an output which is sent to the processing module to evaluate the predictions and create an anomaly signal. With a set of predictions made by the neural network the system uses a binary cross entropy loss function to evaluate each single bit of the prediction, high values for incorrect but confident predictions, and low for incorrect but middling ones. Output processing is carried out using the following procedure:

1. **Compute prediction error**. Compute the binary cross entropy loss e between the symbol prediction $\hat{y}_t$ and the real symbol x_t. This results in an error vector e with the same dimensions as x and $\hat{y}$.

2. **Create a symbol score** (word output score following Taylor's notation). Compute a single score from vector e, possible options include: finding the maximum, minimum, or average bit loss.

3. **Symbol score combination.** The symbol scores for a sequence must be combined into a single value, indicating the "anomaly signal value" for the sequence. Using the symbol output from the previous step, the combination can be done using any of the following methods: taking the mean, maximum, log sum value, or using a rolling window that takes the average score over 0.1s windows and from these it retrieves the maximum value.

Since each ID has its own model, the score results obtained from this module vary between them. Therefore, every ID has its own output processing procedure. In order to check what is the best symbol score and symbol combinations, the detector has to be tested on validation data, using a set of anomalous traffic the best combination is selected based on which one scores the highest AUC on average.

DETECTING AN ANOMALY. With the above methods the output processing module provides a unique anomaly score for every given sequence. From this it is possible to establish a *threshold* that discriminates between anomalous and normal sequences. In the reference work, the authors used an additional validation set with anomalies to set this threshold, however, in this work we omit this last step. The goals of these thesis are two, the first one is to evaluate the use of an unsupervised approach to anomaly detection, by doing so we do not use any anomalous data to set detection thresholds. The second goal is to compare the predictor and autoencoder models, in order to do that they must have the same performance metric so we can compare their results, by using the AUC we evaluate their performance under different threshold values.
In summary, in the predictor-based ADS we only evaluate the output of the symbol combination method without comparing it to a given threshold. By doing this, we can directly compare its results with the autoencoder-based approach that we present next.

4.5 AUTOENCODERS-BASED DETECTION

Based on the arguments presented in Section 4.2, we implement a reconstruction-based approach using autoencoders for anomaly detection on CAN. Based on the research work done by alDosari[1], Assendorp [3], Malhotra et al. [38], we deemed this approach to be feasible and potentially efficient in our context. In our understanding no work has been published discussing the application of autoencoders in automotive security.

In a scenario where we want to learn the high level representation of a time-series, in our particular case model normal CAN traffic, autoencoders are a more intuitive approach than using predictors. We aim to build a model that is able to replicate sequences of normal CAN data so that if it fails in doing so, it can indicate that an attack is taking place. An autoencoder learns a vector representation of the time-series so it can learn what constitutes 'normal' traffic without the need of attack examples, then it reconstructs such sequences based on the representation that it learned. This means that a perfect model outputs exactly the same sequence that was used as an input. Once the autoencoder has been trained, we need to know if a given sequence can be considered anomalous or not, we can make this classification based on how well the model is able to reconstruct the input sequence. The intuition behind this is that given that the

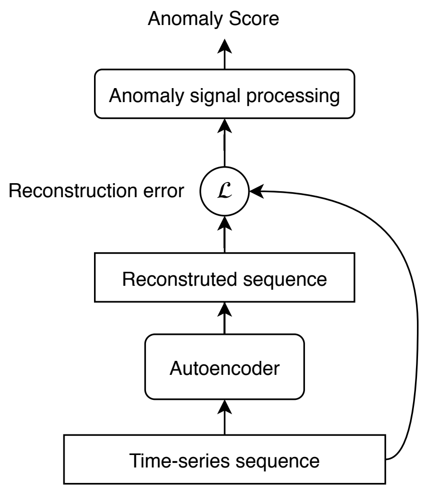


Figure 4.4: Overview of the autoencoder-based ADS.

autoencoder only knows how to reconstruct normal data, if given an anomalous input it would create a bad reconstruction. We use the reconstruction error, that is the difference between the input and reconstructed sequences, to evaluate the performance of the model: low error values indicate that the autoencoder managed to reconstruct the sequence rather well, while if errors are high it indicates that the given sequence is anomalous and potentially an attack.

The anomaly detection system we present here consists of two main components: the autoencoder, used to reconstruct traffic, and the anomaly signal mechanism. The second component is responsible of computing an anomaly score given the autoencoder reconstruction errors. Figure 4.4 shows an overview of the autoencoder-based ADS we propose, it receives sequences of CAN data as input and outputs an anomaly score. The errors computed using the reconstructed sequence from autoencoder only tell us how well the reconstruction is done, but we still need to translate this into a measure of certainty that a given sequence is normal or not. Perfect reconstructions are impossible due to many factors, including the bias of the model itself, and most importantly the noise generated by the data generating mechanisms behind CAN signals.

The anomaly signal mechanism works by computing the statistical characteristics of reconstruction errors over a separate validation dataset, then it assigns a distance score that indicates how far a given reconstruction error is from the expected normal/legitimate distribution. In principle both detector approaches train the models with legitimate/normal data, however with autoencoders we will not be using any validation data with anomalous traffic to tune the anomaly signal, rendering this a completely unsupervised approach to anomaly detection on CAN. We specify how this distribution and distance are computed in Section 4.5.3.

Similarly to the predictor approach, we train an autoencoder for every ID in our dataset, which results in 12 autoencoders in total.

4.5.1 *Model Architecture*

To design the model architecture we begin by following the model proposed by Malhotra et al. It consists of an encoder and a decoder, the encoder consists of an input layer that receives the input sequence $x^{(t)}$ and a single recurrent hidden layer with LSTM units. The decoder consists of a single recurrent hidden layer with LSTM units and an output layer, the state of the decoder recurrent layer is initialized using the final hidden state from the encoder h_E^t. The decoder output layer transforms the state back into the same dimension as the encoder input, scaling its values back to the [0, 1] range and so outputting the sequence reconstruction.

The encoder input is a sequence S as in the predictor case, however, the target used for training is also a sequence. Nonetheless, the same sequence S is not used as a target as it would lead the network to just memorize sequences. Instead, the target sequence is reversed as it has been shown to improve reconstruction performance in sequence-to-sequence modeling applications [65] [1]. An overview of the model is illustrated in Figure 4.5 showing the inference steps for an input $x^{(t)}$ to reconstruct $x'^{(t)}$.

INPUTS The autoencoder uses a similar input to the predictor specified in Section 4.4, a $n \times k$ vector where n is the sequence length with dimensionality k. n is a hyperparameter that can be tuned depending on practical considerations of the ADS. We preemptively set its value to $n = 20$ the same sequence length used by Taylor et al. in their predictor: we made this decision in order to have similar parameters when confronting the predictor and autoencoder models. As we discussed in the input description of the predictor approach, k can be at most 64 but thanks to the removal of constant bits it is reduced depending on the respective ID.

ENCODER AND DECODER HIDDEN LAYERS We performed a random hyperparameter search over this model to obtain an architecture that could properly address our multivariate problem. Malhotra et al. model, even though proposed for multivariate time-series, was not effectively tested with multivariate data. In their experiments they used 4 different datasets, 3 of them univariate, the remaining one was re-

1 Even though Malhotra et al. cite Sutskever et al. to justify the use of reversed sequences, the former used a reversed source sequence with a normal target sequence, while the latter did the opposite, that is a reversed target but regular source sequence. There was not any explanation as for why this change was done, we verified in a test scenario that the reconstruction performance is similar regardless of what sequence gets reversed.

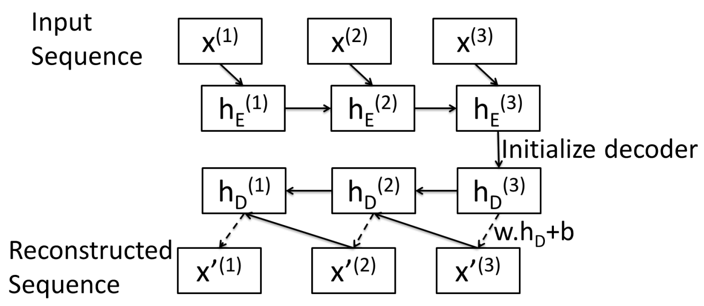


Figure 4.5: Overview of the Encoder-Decoder model proposed by Malhotra et al., taken from [38]

duced to an 1-dimensional series using principal component analysis. Because of this we instinctively assume that the model must be more complex in order to properly characterize the data dimensionality. While conducting the random hyperparameter search, we selected the model with the lowest reconstruction error on a validation set so that the resulting model doesn't just overfit our training data. The search is done using the data from a single CAN ID that was randomly chosen from our training dataset[2]. We began this process by considering a similar architecture to Malhotra et al. model, starting with a model with an input layer, a single LSTM recurrent hidden layer for the encoder and one for the decoder. Our search included additional recurrent and non-recurrent layers, and varying number of units for each layer, from 64 to 512, and both GRU and LSTM units for the recurrent layers. During the search we tested mechanisms to avoid overfitting, including Dropout [64], using L1/L2 regularizers [26] for different layers as well as using none at all.

The final architecture is illustrated in a simplified manner in Figure 4.8, and a more complex representation in Figure 4.9, the model consists of two main components: an *encoder* and a *decoder*.

- An input layer expects $n \times k$ input matrix of each ID. n being the length of the sequence to reconstruct and k the dimensionality.
- A dense feed-forward layer with 256 units ($C = 256$) with hyperbolic tangent activation functions. Dropout is added to this layer with a probability set to 0.2.
- A recurrent layer follows with 128 LSTM ($L = 128$). units, with hyperbolic tangent activation functions.
- The last layer is another recurrent layer with 128 LSTM units. This layer returns the output from the encoder (which is not

2 Tuning the hyperparameters for each ID could in theory result in a better performing model, since the data generating mechanisms are different, albeit a considerable time-consuming task.

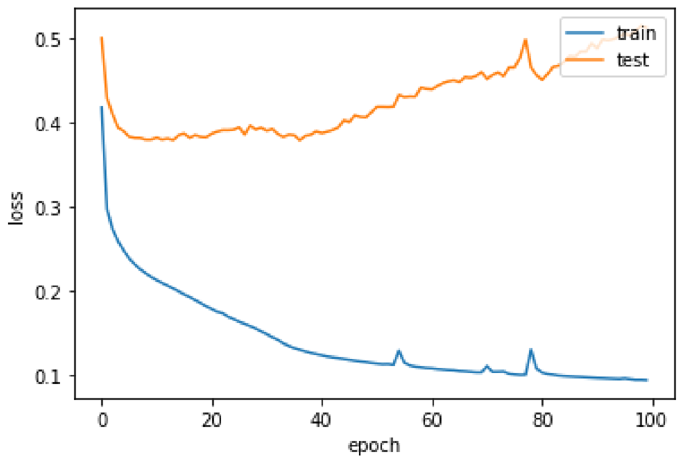


Figure 4.6: Training loss with a validation dataset for a random ID using Malhotra's encoder-decoder model. The model overfits because of the lack of regularization.

Our search began using the same architecture as Malhotra et al.: when training this model we noticed that it can lead to overfitting because there are no regularization processes in place. In Figure 4.6 we can see the model loss for a random ID using Malhotra's model, we observe that after epoch 40 the model starts to perform worse on validation data while the test error kept decreasing, signaling that model is overfitting. In general, a deeper architecture performed better, as it was expected, and we noticed this trend early even during training. As we can observe in Figure 4.7, the reconstruction error of the deep autoencoder is significantly lower for the same ID using exactly the same training and validation data.

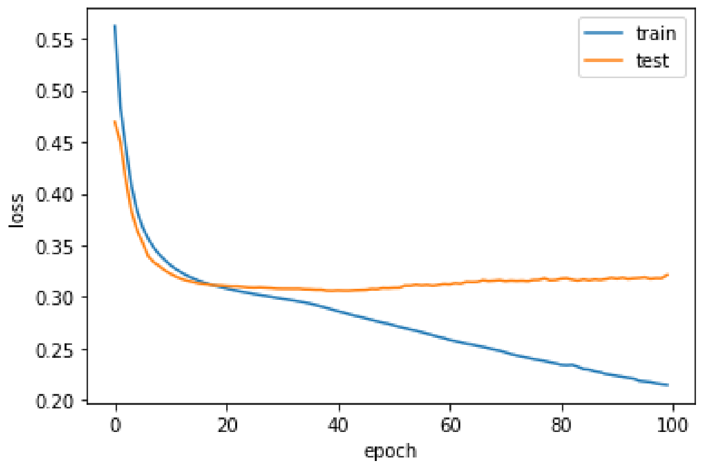


Figure 4.7: Reconstruction error during training for the deep autoencoder model.

used) and the hidden state h. Technically LSTMs output two states: the usual h from classic RNNs and the cell state c, but we denote them both as h for simplicity.

The decoder is consists of:

- A recurrent layer with 128 LSTM units with hyperbolic tangent activations. As an input it expects a vector with similar dimensionality as the decoder output, and the encoder states h to initialize the layer.
- Another recurrent layer follows, similarly with 128 LSTM units with hyperbolic tangent activations.
- A final dense layer that uses sigmoid activation functions to scale the result back into the [0, 1] range. It's output has the same shape as the encoder input.

At each time-step the encoder outputs two hidden states at the last step, these hidden states h_E are used to initialize the decoder.

4.5.2 *Training*

The autoencoder is trained to reconstruct CAN data sequences exclusively using normal traffic data. During training, the autoencoder needs to receive as an input a source and target sequence, i.e. what it is supposed to reconstruct. Instinctively, one would use the same sequences S for both source and target, however Sutskever et al. [65] demonstrated that sequence reconstruction performance improves if the symbol order of the source sequence is reversed, we apply the same principle here. Additionally, the autoencoder needs another input during training, the first LSTM layer of the decoder cannot be left without an input, so similarly to Malhotra et al. the decoder previous outputs are used as the decoder input. During training the model is run on one time-step t, its output reconstruction $s'^{(t)}$ and respective states are stored and reinjected in the next iteration, so the reconstructed sequence $s'^{(t)}$ and its states are used as an decoder input when processing $s^{(t+1)}$. This reinjection is performed only during training, during the normal usage of the models, the decoder input is simply set as a vector of zeros.

In summary, training is done using the following data:

- A *source* set of sequences S, where its symbols x order is <u>reversed</u>:

$$S_0 = x_n, \dots, x_0$$

$$S_1 = x_{n+1}, \dots, x_1$$

$$\dots$$

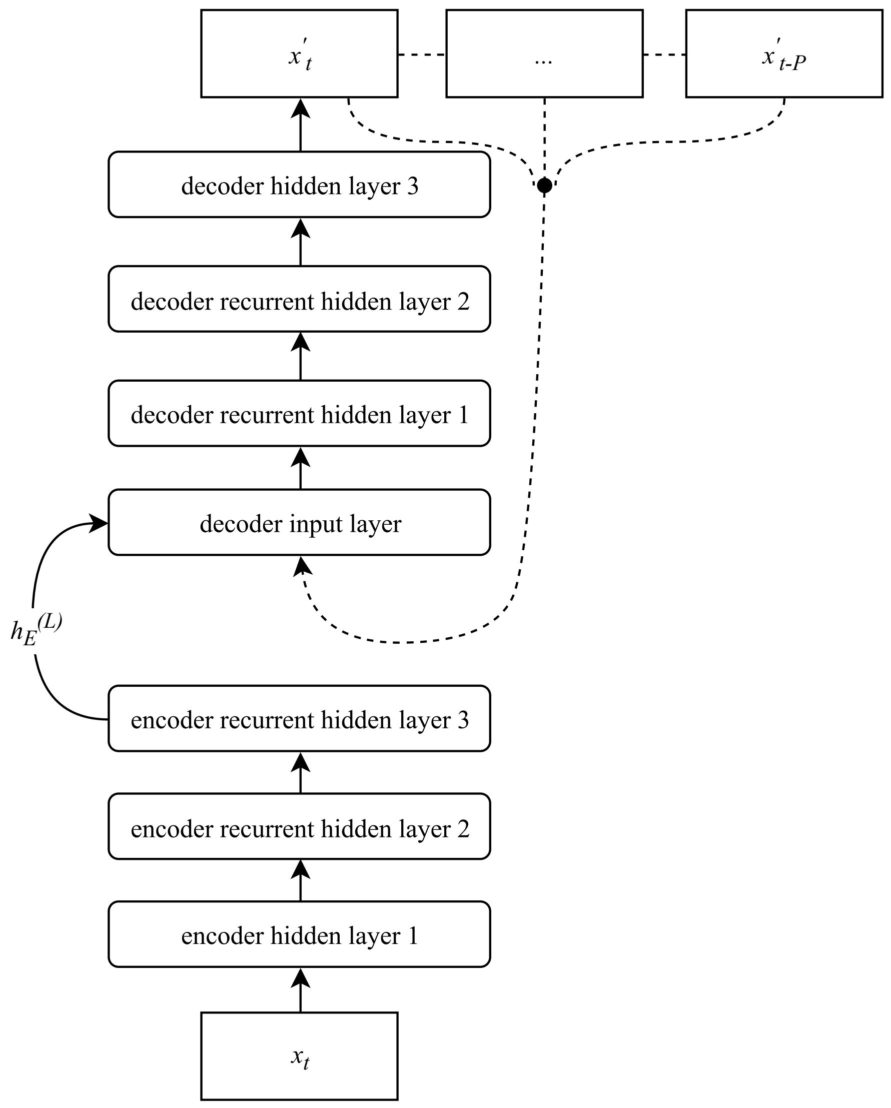

x'_t
...
$x'_{t\text{-}P}$
decoder hidden layer 3
decoder recurrent hidden layer 2
decoder recurrent hidden layer 1
decoder input layer
$h_E^{(L)}$
encoder recurrent hidden layer 3
encoder recurrent hidden layer 2
encoder hidden layer 1
x_t

Figure 4.8: General architecture of the autoencoder. The dashed lines indicate that the decoder input is set to the previous reconstructions only during training. Otherwise use a 0-vector. P corresponds to the number of sequences used only during training.

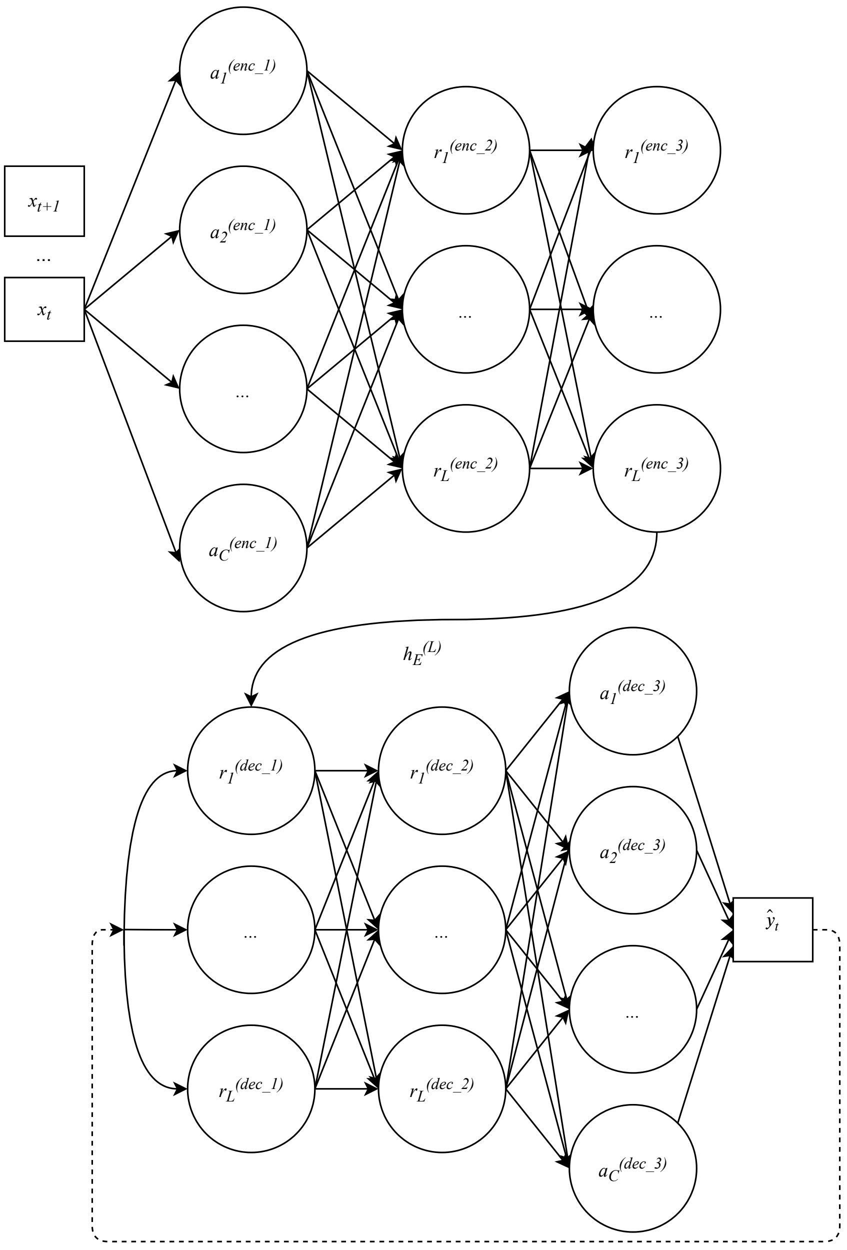


Figure 4.9: Autoencoder model. x_t represents a CAN sequence at time t. $a_C^{(enc/dec)_l}$ is the unit C of linear layer l of either the encoder or the decoder. $r_L^{(enc/dec)_l}$ is the LSTM unit L for the recurrent hidden layer l of either the encoder or decoder. $h_E^{(L)}$ is the encoder's final hidden state used to initialize the decoder hidden states. $\hat{y}_t$ is the reconstructed sequence given x_t. The re-injection is performed only during training, the resulting model the decoder input is set to a 0 vector.

- A set of *target* sequences Y, which is the same as the source sequences but with the symbols in their original order.

$$Y_0 = x_0, \dots, x_n$$

$$Y_1 = x_1, \dots, x_{n+1}$$

$$\dots$$

- The *decoder's input*, initially set to a vector of zeros but as training occurs the model uses S_t and Y_t to output a reconstruction $\hat{Y}_t$, at the next time-step $t+1$ the decoder's input is set to $\hat{Y}_i$. Note that the decoder input LSTM states are still being initialized based on the latest encoder hidden states at the *same* time-step.

PARAMETERS. The autoencoder is trained using mini batch optimization with a batch size of 128 with Adam as the optimizer [31]. As a loss function, we use binary cross entropy, this results in models that harshly penalize incorrect but confident predictions. We also include an early stopping check that stops training if the reconstruction error from a validation set fails to improve after 10 consecutive epochs. As soon as training starts, the decoder input is set to a 0-vector, but as the first decoder outputs are produced, they are continuously reinjected into the decoder input for the rest of the training process.

USAGE. To use a trained autoencoder we need to define its input so it can be used to analyze sequences, these inputs are:

- *Encoder inputs.* A sequence s_t which is composed of L symbols in reversed chronological order $s_t = [x_{t+L}, \dots, x_t]$, and x, a symbol of dimensionality k, $\quad 0 < k \leqslant 64$.
- *Decoder inputs.* A 0-vector of size $1 \times k$

With these inputs the model outputs a reconstruction of the sequence

$$\hat{y}_t = [\hat{x}_t, \dots, \hat{x}_{t+L}]$$

Where $\hat{x}$ is a single reconstructed symbol of size k.

4.5.3 *Anomaly signal processing*

Once we have a model that is able to reconstruct sequences, we need to find a procedure to generate an anomaly score that is be used as a discriminator between normal and anomalous sequences. First we must define how to compute the error of a reconstruction. Unlike Malhotra et al. approach, we do not use the absolute error function, instead we use the binary cross-entropy function to compute the reconstruction error:

$$-(b_k \log(\hat{b}_k + \epsilon) + (1 - b_k)\log(1 - \hat{b}_k) + \epsilon)$$

Where b_k is the k-th bit in the reconstructed sequence y_i and b_k is the k-th bit of the source (or input) sequence.

By computing the reconstruction error of a single sequence, we obtain an error vector with the same cardinality as the input sequence, that is using original and reconstructed sequences of size $n \times k$ will result in a $n \times k$ reconstruction error vector. The next step is to determine how we can establish what are the normal reconstruction errors that occur in normal data sequences.

The processing step yields different results for each autoencoder that we trained, as the errors significantly vary between IDs. Therefore, the anomaly signal processing procedure must be carried out for each CAN ID.

NORMAL RECONSTRUCTION ERROR DISTRIBUTION. To determine if a reconstruction error for a sequence can be considered anomalous, we must first establish what is a 'normal' error using legitimate data sequences. Using a separate validation dataset we compute the reconstruction errors of a set of normal sequences which result in a $s \times n \times k$ error vector, where s is the number of sequence samples, n is the length of each sequence, and k is the dimensionality of the symbols in that sequence. From such errors we can compute the statistical characteristics of the distribution, but first we need to reshape the vector so we can compute these values. We reshape the $s \times n \times k$ error vector into $s * n \times k$, thus obtaining a continuous list of reconstruction errors.

Successively, we fit these errors into a multivariate Gaussian distribution, which gives us the mean μ of dimension k and covariance matrix Σ with $k \times k$ dimension.

ANOMALY SCORE. Now that the normal error distribution is available we need to compute the likelihood of an anomaly. We use a similar approach to [37, 38], an anomaly score a is derived from the Mahalanobis distance measuring the distance between a reconstruction error and the distribution of normal/legitimate errors, and it is computed as follows:

$$a = (e - \mu)^T \Sigma^{-1} (e - \mu)$$

Where e is the reconstruction error vector of dimensions $s * n \times k$.

We need to test how correct is the assumption that these scores are different under anomalous scenarios. Just for the purpose of illustration we present the variations in the score distributions when

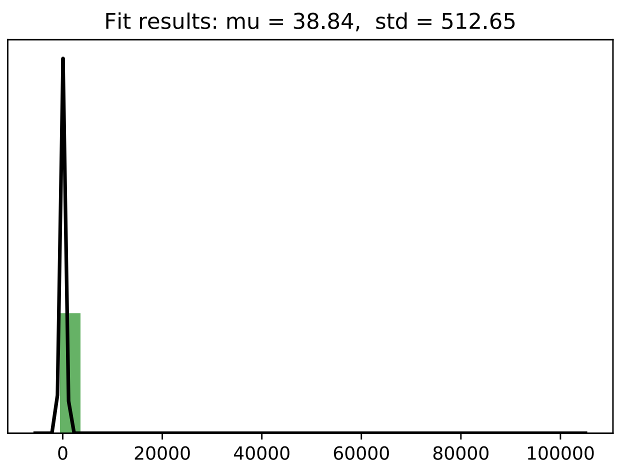


Figure 4.10: Anomaly score distribution for test sequences with normal data.

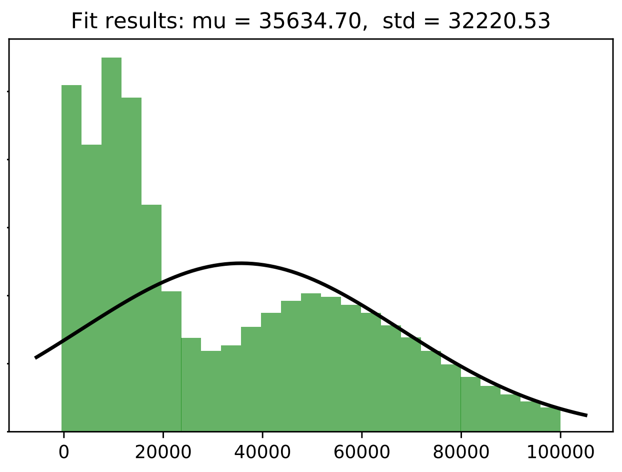


Figure 4.11: Anomaly score distribution for test sequences with an interleave anomaly.

anomalies are present. In Figure 4.10 we show the anomaly score distribution using only legitimate sequences along with its mean and standard deviation. Figure 4.11 illustrates an example using an *interleave* anomaly, we can see the significant change in the scores distribution. In Figure 4.12 we illustrate the changes when executing a *replay* anomaly targeting a high-frequency field, they are not as drastic as in the interleave case but we can still see evident changes in the scores distribution. This gives us an idea of how drastic anomalies alter the reconstruction error distribution of a set of test sequences.

4.6 DETECTORS EVALUATION

We can now use this anomaly score to compute the ROC AUC of the autoencoders using our simulated anomalies. We have specified

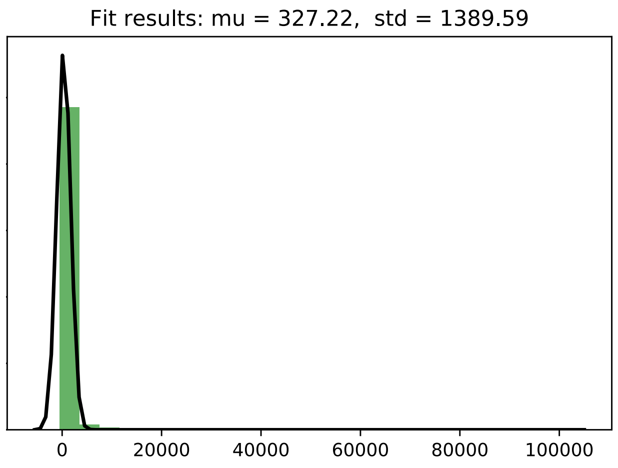


Figure 4.12: Anomaly score distribution for test sequences with a replay attack targeting a high variability field.

the proposal that uses a predictor-based detector and we also defined how our autoencoder detector operates. In the next chapter we evaluate how these two approaches compare to each other. We theorized that the autoencoder approach could yield better results given the fact that it does not make any assumptions on the data generating mechanisms in place, and in doing so removing the strong assumption that CAN traffic data is predictable. We use similar implementation and experimental parameters to ensure that the detectors are being compared under the same scenarios. For example, both detectors use the same data sequence as input with the same sequence length.

Both detectors are tested using the same simulated anomalies with the same anomaly parameters, during the anomaly test phase we compute the ROC AUC of the detectors so we can evaluate their performance and how they compare.

5 EVALUATION

5.1 INTRODUCTION

In this chapter we present our experiments and results for our data sequence anomaly detection task.

We begin by explaining our experimental setup and data, including how our datasets are divided. We explain our data pre-processing procedure and then present the results of the post-processing for both detectors. Afterwards, we present the process used to simulate anomalies.

We show the results of our anomaly experiments divided by the two main categories of anomalies we consider. For each one, we present an overview of the results, and in the case of data field tests, we also show the results in the context of different parameter combinations and offer some insight on them.

With our experimental evaluation we set three main goals:

1. Show that autoencoders provide a good model for anomaly detection in data sequences.
2. Demonstrate that autoencoders can be used as a complementary model or even substitute predictors entirely in an anomaly detection system. This is demonstrated using the performance scores we defined.
3. Evaluate the shortcomings of the autoencoder approach so they can be improved in future work.

5.2 EXPERIMENTAL SETUP

The software tools were chosen because of the wide variety of libraries for fast prototyping with a wide collection of machine learning algorithms, including neural networks, data analysis, among others. We used the following software throughout the course of this work:

- Python 3.6[1]
- Tensorflow 1.12 with Python API [2]
- Keras 2.2.4 with Tensorflow back-end[3]

1 Available at: https://www.python.org/
2 Available at: https://www.tensorflow.org/
3 Available at: https://keras.io/

- Pandas 0.24.1 (Library for Python) [4]
- Numpy 1.15.4 (Library for Python) [5]
- scikit-learn 0.20.2 (Library for Python) [6]
- Jupyter 4.4.0 (for preliminary analysis and interactive data exploration) [7]

5.3 EXPERIMENTAL DATA

Datasets

In our experimental work we use a database with more than 6 million messages of logged CAN traffic, the data is obtained from a Alfa Romeo Giulia Veloce vehicle as we described in Section 2.3.2. It is saved on disk as a text file and it contains the columns Timestamp, ID, DLC, and Data.

Overall, we did not identify any missing information or data that could be considered as corrupted or that may hinder our work. Nonetheless, the only issue came from a couple of small temporal gaps in the readings. Some gaps were present which are a couple seconds long, and one is 7 minutes long. Temporal gaps are problematic if not managed correctly as they cause data discontinuities in the data streams and this would negatively affect the performance of the models as we train them. Despite of that, we took advantage of these gaps to divide our data logs into disjoint datasets, which we describe bellow.

We divide the CAN recording files according to the different data needs we have identified, each dataset is disjoint from one another so there are no overlaps between them:

- $\mathtt{Data}_{\mathtt{train}}$ is a dataset used for training both our predictor and autoencoder models. This dataset corresponds to around 60% of all our available data.
- $\mathtt{Data}_{\mathtt{train\_val}}$ is used as a validation set during training. It is used to check the performance of our models as they are being trained, which is effectively used for *early stopping* to avoid overfitting. This dataset corresponds to around 10% of the total available data.
- $\mathtt{Data}_{\mathtt{scores}}$ is a dataset used to configure the anomaly signal mechanisms of both detectors. In the case of the predictor, it is used to find the best symbol output and symbol combinations (half of the data is replaced with anomalous data, albeit non

4 Available at: https://pandas.pydata.org/
5 Available at: https://www.numpy.org/
6 Available at: https://scikit-learn.org/stable/
7 Available at: https://jupyter.org/

permanently). For the autoencoder, this dataset is used to fit the distribution of reconstruction errors and the anomaly scores that are present in normal CAN data. This dataset corresponds to round 10% of the total available data.

- $Data_{anomalies}$ is a dataset used exclusively for anomaly tests, during the evaluation of our systems 40% of this dataset is modified to include anomalies while the rest is left for normal/legitimate traffic. This dataset corresponds to 20% of all our available data.

Following Taylor's approach, we only consider CAN IDs with high frequency, high symbol variability, and non-trivial symbol complexity (i.e. the percentage of unique symbols is higher than 1%). The IDs in our dataset that match these characteristics are:

```
0DE, 0EE, 0FB, 0FC, 0FE, 0FF, 1F7, 1FB, 11C, 100, 104, 116.
```

5.3.1 *Data Preprocessing*

CAN data traffic, as directly logged from the bus, is not yet suitable for training neural networks and to be used with the predictor and autoencoder models we have presented. Additionally, it also not usable to simulate the anomalies we have described. For these reasons, we must first follow pre-processing procedures to prepare our dataset.

One of the first steps during pre-processing is to create a binary representation of the data field. To avoid converting data every time we load the dataset, we add a new column to our dataset named `DataBin` which is a binary representation of the hexadecimal data present at each CAN message. This process also creates a new column named `DataBin_nc`, which represents the data of `DataBin` but with the constant bits removed. However, this requires that the whole dataset is used in order to guarantee that the constant bits are consistent across files and datasets.

5.3.1.1 *Field Classification*

As part of the data pre-processing process we implement the Field Classification Algorithm presented by Markovitz and Wool[41] and discussed in Section 2.4.1. Before executing the algorithm, it requires that the data is represented as a sequence of bits rather than a hexadecimal string. It is also relevant to note that all data across files is passed to the algorithm so the results are consistent throughout our datasets As an example of the importance of using the entire dataset, we observed that during our preliminary tests some of the bits deemed constant in the training dataset later changed in our test sets, this rendered our models incompatible across different datasets

as they had inconsistent dimensionality when constant bits are removed.
There are parameters that have to be set prior to the execution of the algorithm. To classify fields as either *sensor* or *multi-value*, we used the same parameters for maximum amount of unique symbols and candidate field length used in the reference work $T_{V_{max}} = 12$ and $T_{L_{min}} = 4$, these values were selected to minimize the false positive rate. Since we do not have any ground truth to test the identification algorithm we could not verify the accuracy of the resulting fields.

The algorithm follows this general procedure: as an input, the algorithm uses the 64 bits of the data frame, executes the identification process, and outputs a list of disjoint fields for each ID. The output includes the leftmost-bit (i.e. index), length, type (multi-value, sensor, constant), category (low/medium/high variability), and the number of unique symbols. The output is then saved as a file on disk containing the list of disjoint fields and their attributes for every ID.

The full list of IDs and their identified fields are found in the Appendix A.

5.4 MODEL TRAINING

We trained 24 models in total as there are 12 CAN IDs we consider and 2 detectors, and each model is saved on a separate `.h5` file. One training process is executed for each given the available memory space we had on the machine we used. However, no GPU was available so training was conducted only using the CPU. Because of this, training took a significant amount of time, requiring 2-3 days to train each predictor and 3 to train each autoencoder (which was expected since autoencoders are known to take longer to train). In our case, the total training time to obtain all models was 6 days. For future work and replication we encourage the use of GPUs for training, and if the right infrastructure is available, follow Tensorflow's data input pipeline performance [8].

5.5 POST-PROCESSING

5.5.1 *Predictor post-processing*

There is no single best match of symbol scores and combination methods for all IDs. Using anomalous traffic samples from the validation set $Data_{scores}$, we choose the combination that maximizes the AUC, this is the step that renders Taylor's method a semi-supervised learning approach. In their early work [67], the maximum loss symbol score was considered to perform best for all CAN IDs, in their later work [66] results slightly changed but in general this mechanism still

8 Available at: https://www.tensorflow.org/guide/performance/datasets

scored better overall. However, when we replicated the experiment with our own data the results were more varied. For each ID we computed the AUC for every anomaly type using all possible output processing methods, the combination that performed best overall was chosen. The final results included a combination of the symbol scores: maximum and average, with combination scores: rolling window, log sum, maximum, and average.

The best combination results can be seen in Table 5.1. These are the anomaly signal outputs which indicate an anomaly score for a given test sequence.

ID	Symbol output	Symbol combination
0DE	Max bit loss	Rolling window
0EE	Max bit loss	Rolling window
0FB	Max bit loss	Rolling window
0FC	Max bit loss	Rolling window
0FE	Max bit loss	Maximum symbol loss
0FF	Max bit loss	Rolling window
1F7	Max bit loss	Maximum symbol loss
1FB	Max bit loss	Average symbol loss
11C	Max bit loss	Rolling window
100	Max bit loss	Rolling window
104	Max bit loss	Rolling window
116	Average bit loss	Average symbol loss

Table 5.1: Symbol score and combination results for every ID..

5.5.2 *Autoencoder post-processing*

As we explained in Section 4.5.3, the post-processing procedure for the autoencoder is fairly simple and it does not require evaluating parameter combinations.

To produce an anomaly score we follow the same mechanism used by Malhotra et al.[37, 38]. For every CAN ID, we find the reconstruction error distribution using only normal/legitimate data from our validation set $Data_{scores}$. This results in a mean μ and a covariance matrix Σ that indicates us the multivariate Gaussian distribution of non-anomalous data. We create a file for each ID specifying the respective values for μ and Σ.

To determine if a given test sequence is anomalous, we compute the Mahalanobis distance between the distribution we obtained previously and the reconstruction errors of the test sequence.

5.6 ANOMALY TESTS RESULTS

5.6.1 *Defining the test sequences*

Before simulating anomalies we need to describe how our test sequences are defined. They are created using continuous and non-overlapping data sequences from the $\mathrm{Data}_{\mathrm{anomalies}}$ dataset. Each sequence is 3 seconds long and contains 300 observations, since all the CAN IDs we considered have a message frequency of 10 milliseconds. The constant bits for each ID are not ignored since they are needed when simulating anomalies targeting specific fields, constant bits are removed after the anomaly sequences are created so we can use the models we trained. Test sequences are therefore a 3-dimensional matrix consisting of:
`number_of_sequences` $\times$ 300 $\times$ 64, for example:

$$
\begin{aligned}
s_{\mathrm{anom},0} &= x_0, x_1, \ldots, x_{299} \\
s_{\mathrm{anom},1} &= x_{300}, x_{301}, \ldots, x_{699} \\
&\ldots
\end{aligned}
$$

Where x_i is a single data frame with dimensionality 64.

For every anomaly type, we create a new set of test sequences. We keep 60% of the original sequences as legitimate sequences, the remaining 40% is altered using the anomaly functions

5.6.2 *Simulating anomalies*

In this Section we describe the implementation of the anomalies that we presented on the Threat Model in Section 3.2.4. To recap, the anomalies *Interleave* and *Reverse* modify the test sequences but not the contents of each symbol. Therefore, the only parameter needed to create such anomalies is the test sequence itself. The *Discontinuity* anomaly has an additional parameter, which is the number of symbols to remove from the sequence. We set this parameter to 10.

On the other hand, *field anomalies* have additional parameters and we must evaluate the detectors using different combinations. The parameter combination process is performed in the following order:

1. *Anomaly function*. There are 5 field modification functions we consider, *set to maximum, set to minimum, set to a constant, set to random value, replay field.*

2. *Field variability*. We consider all the 3 variability types (low, medium, and high).

3. *Target field*. If a field with the chosen variability exists, we randomly select a field for the target ID.

4. *Duration*. How long is the anomaly in the test sequence. 4 durations are evaluated: 0.2, 0.5, 1, and 1.5 seconds. The anomaly starting time is chosen at random, but not earlier than one third of the total length of the test sequence and early enough to accommodate the entire anomaly duration. This allows us to evaluate the impact of the duration time for every anomaly.

By selecting the target field before the duration, we guarantee that we are consistently evaluating the anomaly under the same parameters (anomaly function and variability type). We also note that in our experiments we used exactly the same fields when testing the predictor and the autoencoder, this was done in order to have a consistent parameter choice between the detectors.
Taylor in their experiments implemented field anomalies in a way that only targets *Sensor* fields, in this work we removed this restriction so we can better evaluate detector performance using a greater range of fields.

Note that there are potentially 64 anomaly test cases for every CAN ID. However, in our experiments most IDs do not have fields for every variability category therefore in most cases the actual number of parameter combinations is 40.

5.7 ANOMALY RESULTS

In this section we present the results from the anomaly detection tasks using both detectors. In all cases we use the AUC as the performance metric to compare them.

While presenting the results, we first show how the detectors perform when analyzing *interleave*, *discontinuity*, and *reverse* anomalies. Then we evaluate the results in *data field* attack scenarios, where we evaluate the performance using the set of parameters used to create the anomalies.

5.8 ANOMALY SCENARIO 1: GENERAL SEQUENCE ANOMALIES

5.8.1 *Interleave*

We begin this evaluation by presenting the detectors results when testing for *interleave* anomalies. The AUC for every ID is presented in Table 5.2 as well as the performance average for both detectors. These anomalies were among the easiest to detect because of the noticeable impact they have in the data stream, half of the symbols in the sequences are drawn from a pool from a different point in time, and so each anomalous sequence has half of its symbols from a completely different context.

Both detectors score very high AUC values and there does not seem to be a a clear winner in this case.

	Predictor	Autoencoder
0DE	1	1
0EE	1	0.9846
0FB	1	1
0FC	1	1
0FE	1	1
0FF	1	1
1F7	0.9935	1
1FB	0.9894	1
11C	0.9982	1
100	1	1
104	0.9965	1
116	1	0.9997
avg	0.9981	0.9986

Table 5.2: Interleave anomaly AUC for all IDS.

	Predictor	Autoencoder
0DE	0.9996	1
0EE	0.9939	0.9437
0FB	0.9938	1
0FC	0.9921	1
0FE	0.9998	1
0FF	0.9911	1
1F7	0.9563	1
1FB	0.9961	1
11C	1	1
100	0.9957	1
104	1	1
116	0.9974	0.7892
avg	0.9930	0.9777

Table 5.3: Discontinuity anomaly AUC for all IDS.

5.8.2 *Discontinuity*

Next, in Table 5.3 we present the results for the *discontinuity* anomaly case. Both detectors yield significantly good results overall. However, on average the predictor obtained a higher AUC for all IDs, albeit slightly. The main factor contributing to this result came from a specific ID (116), for which the autoencoder performed poorly with an AUC of 0.789 while the predictor achieved 0.997. However, we were not able to find factors that could indicate why the autoencoder performs worse for this specific ID but almost perfectly for all other cases.

	Predictor	Autoencoder
0DE	1	1
0EE	1	0.9622
0FB	0.9999	1
0FC	0.9996	1
0FE	1	1
0FF	0.9991	1
1F7	0.9930	1
1FB	1	1
11C	1	1
100	1	1
104	1	1
116	0.9960	0.9587
avg	0.9989	0.9934

Table 5.4: Reverse anomaly AUC for all IDS.

5.8.3 *Reverse*

Finally, the *reverse* anomaly results are presented in Table 5.4. Both detectors can detect these anomalies with very good performance, as it was expected. This scenario does not match any attacks found in literature, but it serves as a control experiment as done in [67].

Overall, we can see that both detectors perform well at detecting these types of anomalies. This result is to be expected given the fact that the all bits for every anomalous symbol are out of context, meaning we can have a maximum of 64 anomalous bits per symbol. This causes a chain of errors that significantly grows as we compute the anomaly flow, from bit to symbol to sequence error. However, this is not the case in data field anomalies which we will present next. In this context only a subset of the symbols bits are used and these can vary significantly, being either 6 bits or even 52 bits long.

Predictor	Autoencoder
0.932265	0.967711

Table 5.5: Average AUC results for all data field anomalies across all IDs

5.8.4 *Caveats*

These results are overall satisfying, as performance is quite high. However, all of these anomalies have something in common, they directly alter the rates at which messages are seen on the bus. As we introduced in Section 3.3.3, frequency detection methods have been proven to be quite efficient at detecting these changes, achieving perfect performance in many cases. Which approach yields better performance remains to be evaluated, however these *frequency-based* methods are effectively much simpler to design, implement, and execute, requiring significantly less demanding computing demands.

5.9 ANOMALY SCENARIO 2: TARGETING FIELDS

In this section we analyze the results of the data field anomaly test cases. To begin, in Table 5.5 we show the average AUC for all IDs over all data field anomalies, we see that performance is not that much different with our autoencoder detector, achieving an AUC average improvement of 0.034.

5.9.1 *Overall results*

One of the parameters that have the greatest impact in the results is the identifier, each data generating mechanism is different and in consequence so is the capability of the models to characterize them. In Figure 5.1, we illustrate a direct comparison between the detectors for each ID, we observe that in 7 out of 12 cases performance was very similar as there's an AUC difference less than 0.02, as shown in Table 5.6. However, for 4 IDs the difference was more significant, with AUC improvements ranging from 0.05 to 0.19.

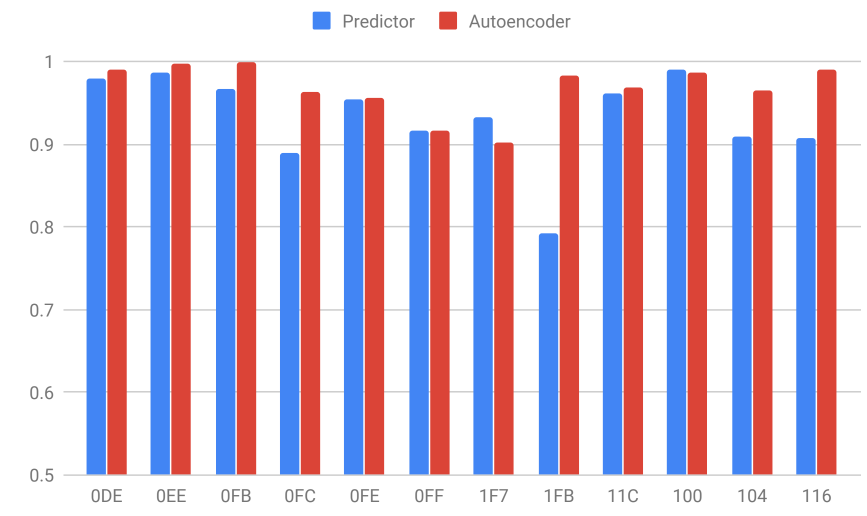


Figure 5.1: Average AUC per ID for all data field anomalies.

ID	AUC difference
0DE	0.0090
0EE	0.0093
0FB	0.0327
0FC	0.0729
0FE	0.008
0FF	0.006
1F7	-0.0305
1FB	0.1913
11C	0.0067
100	-0.0049
104	0.0551
116	0.0822

Table 5.6: AUC difference between detectors for all data field anomalies. A negative value indicates the predictor performs better.

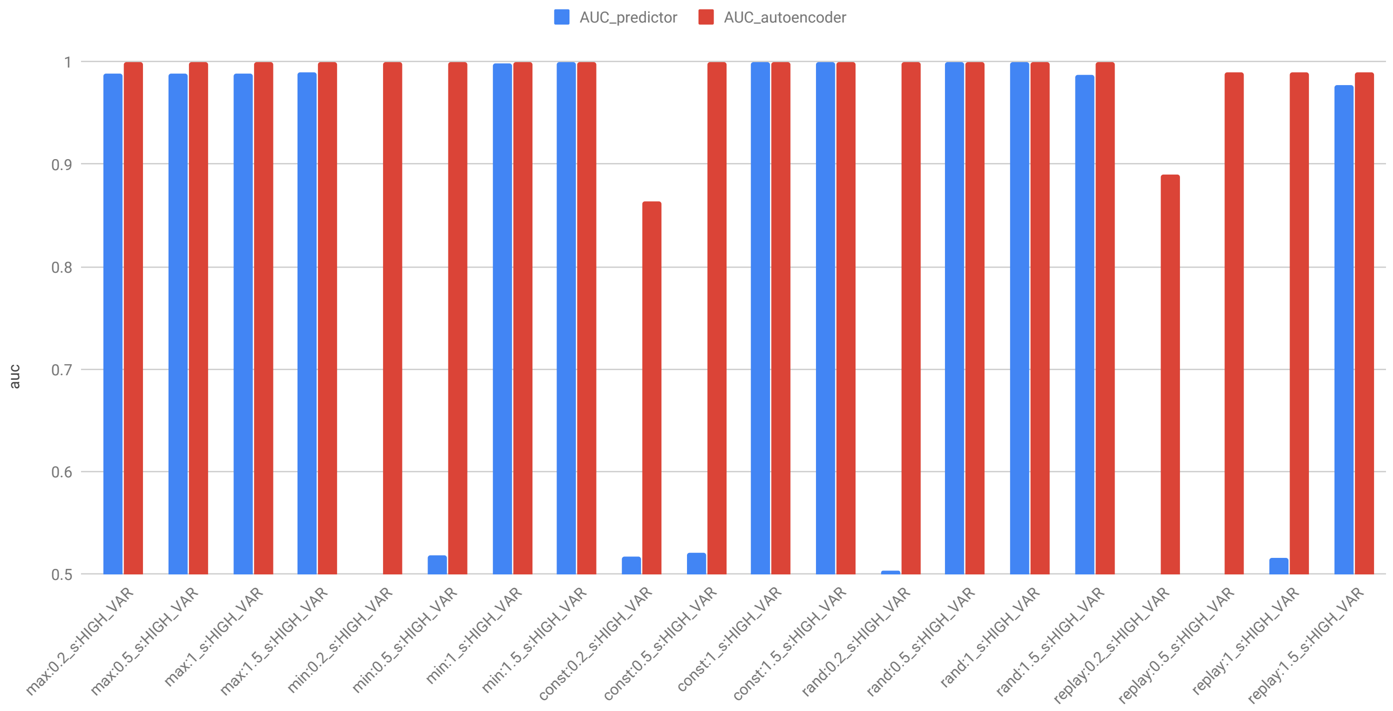


Figure 5.2: AUC results for ID 1FB. The x-axis indicates the parameters for each test as specified in Section 5.6.2

Anomaly function	**Predictor**	**Autoencoder**
Max	0.9341	0.9440
Min	0.8986	0.9264
Constant	0.9508	0.9514
Random	0.9908	0.9944
Replay	0.8087	0.9521

Table 5.7: Overall AUC of the detectors grouped by anomaly function.

Looking at the results for each ID allows us to see trends that are otherwise obfuscated by overall results, particularly when the detectors have similar performance across the parameter space. To better illustrate the detectors performance throughout the test parameters, we take ID 1FB as an example, which is illustrated in Figure 5.2. We chose this ID as it presents the greatest improvement when switching from a predictor to our autoencoder. We see that the type of anomaly influences the detectors performance, in this particular example the autoencoder performs significantly better for every duration when testing for *field replay* anomalies. The detectors share good results in several scenarios, in the example we can observe that the predictor eventually catches up with the autoencoder with longer lasting anomalies, but the performance gap at shorter durations is quite significant, even yielding an AUC difference of 0.49. In this particular case, the predictor performs slightly better than just making a random choice on whether or not the sequences are anomalous.

In the following sections, we present an evaluation considering three important parameters that influence the detection results: we begin analyzing the AUC by the anomaly function used, then we analyze the results depending on the target field variability, and finally on the duration.

5.9.2 *Influence of the anomaly function*

We evaluate if there is a relation between the anomaly function and the detectors performance. We combine the AUC of all IDs and grouping them by the anomaly function, as presented in Table 5.7. We can observe that performance is fairly similar between the detectors, yet the autoencoder always resulted as the highest performer overall . However, we can point out a clear pattern, autoencoders are significantly better at detecting *replay* anomalies than predictors, the difference between their respective AUCs is 0.15.

Field variability	Predictor	Autoencoder
Low	0.9239	0.9741
Medium	0.8837	0.9106
High	0.9594	0.9754

Table 5.8: Overall data field AUC anomaly results divided by field variability.

5.9.3 *Influence of field variability*

We now evaluate the influence of symbol field variability as a in the detectors performance. In Table 5.8, we see the average AUC results for all IDs and grouped by the field target variability. We see that in this framing, the autoencoder on average performs better. One of the most remarkable results is the improvement of attacks targeting low variability fields. As a matter of fact, Taylor et al. highlighted that their detector performed worse using fields of low variability, and on high variability cases it performs best, our results partially confirm this conclusion.

However, in our tests, mid variability fields performed worse than the others. By studying all the medium variability fields in our dataset, which are presented in Appendix A, we see that these fields tend to have a very short length, usually between 4 and 7 bits long. They are also not particularly common, found in only 5 of our 12 IDs. As the length is short with respect to other fields, the errors are much more obfuscated by the surrounding legitimate bits.

We also evaluate the detection performance relation between variability and the anomaly function. In Figure 5.3 we illustrate the AUC results by grouping variability and anomaly function, we observe that performance is fairly similar for both detectors with the autoencoder having a marginal advantage. From these results we can observe that autoencoders have significant better performance at detecting replay anomalies when the target fields have low or medium variability, with high-variability fields they have better performance on average but the difference is not as significant as the other cases.

5.9.4 *Influence of duration*

How long the anomaly lasts has a clear impact on the detectors performance, intuitively one would assume that the longer the anomaly the better the chances to detect it. Indeed, we confirm this intuition as can be seen in Table 5.9. For both detectors longer anomalies result in a higher AUC values. By grouping variability and duration, as illustrated in Figure 5.4 and in Table 5.10, we can confirm that autoencoders perform significantly better on average for all short-lasting

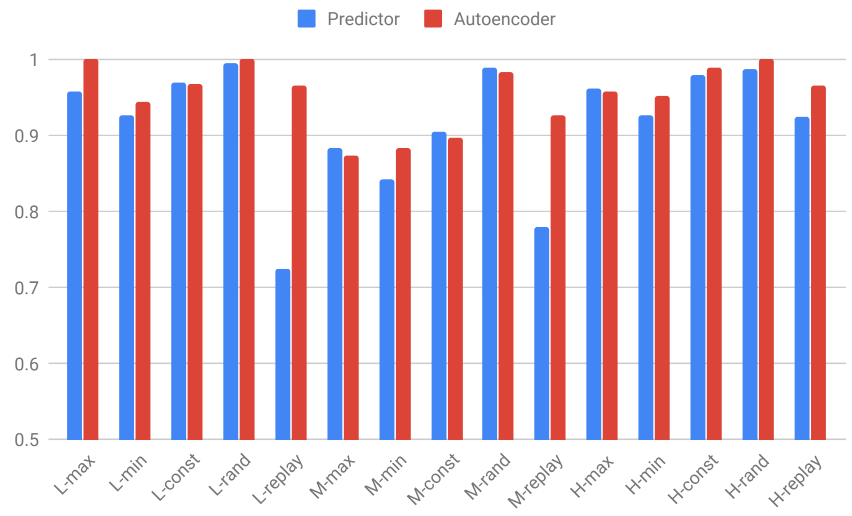


Figure 5.3: Overall detection AUC, grouped by anomaly function and variability. The first letter stands for the target field variability (Low, Medium, and High), and it is followed by the anomaly function.

Anomaly duration (s)	Predictor	Autoencoder
0.2	0.9003	0.9260
0.5	0.9084	0.9535
1	0.9241	0.9648
1.5	0.9336	0.9704

Table 5.9: Overall detector AUC anomaly grouped by attack duration.

anomalies that target low variability fields. On the other cases performance is relatively similar, but we can see the gap closes as anomalies last longer.

5.10 OBSERVATIONS ON POORLY PERFORMING CONTEXTS

In the preceding sections, we mostly considered average results for our anomaly detection tests. By analyzing different levels of abstraction over the results we managed to draw some patterns and findings on the factors that contribute to detection performance, as well as to directly confront two anomaly detection mechanisms.

We look at some of the specific scenarios where both detectors performed poorly as to know the reason why this is the case. Here are some examples and findings:

- ID 104 (see appendix B.11) shows that both detectors perform poorly when using a "set field to minimum" anomaly function on a high-variability field. By looking at the average bit values

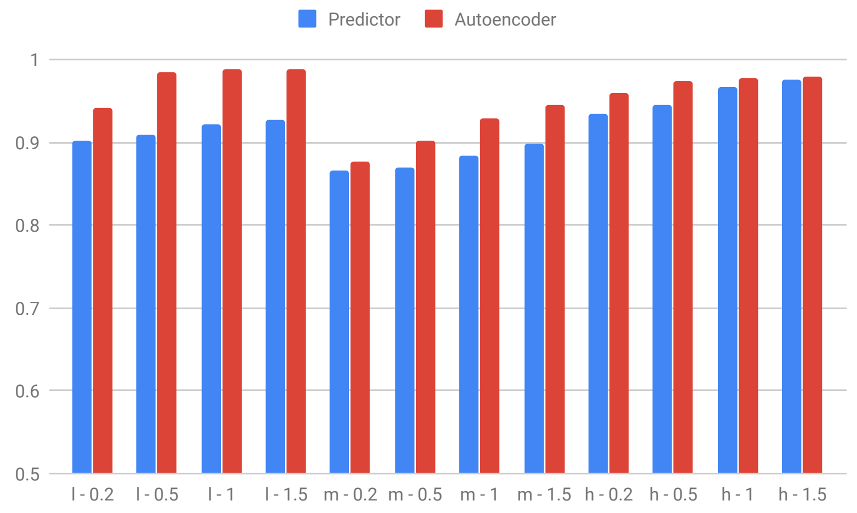


Figure 5.4: Overall detection AUC, grouped by variability and duration. The first letter stands for the target field variability (Low, Medium, and High), and it is followed by the anomaly duration in seconds.

		Low var.				Mid var.				High var.			
ID		0.2(s)	0.5(s)	1(s)	1.5(s)	0.2(s)	0.5(s)	1(s)	1.5(s)	0.2(s)	0.5(s)	1	1.5(s)
0DE	P.	0.9721	0.9725	0.9772	0.9795					0.9520	0.9764	0.9879	0.9914
	AE.	0.9819	0.9944	0.9996	0.9997					0.9560	0.9832	0.9897	0.9910
0EE	P.									0.9870	0.9749	0.9878	0.9888
	AE.									0.9911	0.9957	1.0000	0.9994
0FB	P.					0.9561	0.9553	0.9013	0.9640	0.9903	0.9905	0.9920	0.9940
	AE.					0.9989	1.0000	0.9983	1.0000	1.0000	1.0000	1.0000	1.0000
0FC	P.	0.9376	0.9419	0.9456	0.9475	0.7119	0.7104	0.7396	0.7566	0.9855	0.9924	0.9933	0.9941
	AE.	0.9942	0.9976	0.9951	1.0000	0.8490	0.8974	0.8837	0.9203	1.0000	1.0000	1.0000	1.0000
0FE	P.	0.9252	0.9377	0.9432	0.9504	0.8965	0.9019	0.9047	0.9107	0.9966	0.9998	0.9991	0.9997
	AE.	0.9044	1.0000	1.0000	1.0000	0.8289	0.9063	0.9913	0.9924	1.0000	1.0000	1.0000	1.0000
0FF	P.	0.9502	0.9603	0.9635	0.9667	0.7677	0.7515	0.8253	0.8245	0.9573	0.9737	0.9803	0.9827
	AE.	0.9754	0.9997	1.0000	0.9993	0.7768	0.7519	0.8136	0.8352	0.9304	0.9893	0.9951	0.9981
1F7	P.	0.9326	0.9367	0.9439	0.9521	0.9081	0.9425	0.9734	0.9744	0.8734	0.8760	0.8796	0.8778
	AE.	0.9039	0.9157	0.9198	0.9133	0.8442	0.8793	0.9148	0.9397	0.8306	0.8475	0.8565	0.8706
1FB	P.									0.5970	0.7031	0.9005	0.9909
	AE.									0.9507	0.9980	0.9979	0.9979
11C	P.	0.9168	0.9149	0.9191	0.9212					0.9879	0.9894	0.9918	0.9934
	AE.	0.8272	0.9841	0.9978	0.9998					0.9918	0.9990	0.9999	1.0000
100	P.									0.9822	0.9882	0.9906	0.9954
	AE.									0.9745	0.9778	0.9873	0.9964
104	P.	0.8179	0.8174	0.8205	0.8218	0.9559	0.9579	0.9609	0.9607	0.8939	0.8881	0.9001	0.9026
	AE.	0.9884	0.9995	1.0000	1.0000	0.9670	0.9759	0.9719	0.9802	0.8941	0.9014	0.9020	0.9020
116	P.	0.7599	0.7954	0.8579	0.8748					0.9973	0.9976	0.9978	0.9993
	AE.	0.9508	0.9854	0.9938	0.9869					0.9982	0.9988	0.9995	0.9977
Avg	P.	0.9016	0.9096	0.9214	0.9268	0.8660	0.8699	0.8842	0.8985	0.9334	0.9458	0.9667	0.9758
	AE.	0.9408	0.9846	0.9883	0.9874	0.8775	0.9018	0.9289	0.9446	0.9598	0.9742	0.9773	0.9794

Table 5.10: AUC results grouped by anomaly duration and target field category. Blank cells indicate that a relevant field does not exist. P stands for the predictor results, and AE. stands for the autoencoder results.

over the target field, we observed that the average values for that field are in fact very low in the unaltered dataset, with many bits having average values ranging from 0.1 to 0.3.

- ID 0FF (see appendix B.6) is a particular case since it shows that the autoencoder struggled significantly when detecting some field anomalies with medium variability. We briefly inspected the relevant fields that yielded these results, the field in question was very short in length using 6 bits of the total 64. But most importantly, we noticed that 3 of its bits remain constant in the dataset we used for training, while they effectively change in the test datasets. In brief, the detectors effectively learned some bits as constant. Form this we can draw a hypothesis that the autoencoder may be more sensible to these constant bits, from which we can infer that it needs more training data to overcome this issue.

5.11 SUMMARY

Through our experiments results we demonstrated that overall the autoencoder performs better than the predictor in all anomaly scenarios. By combining the resulting AUCs in different configurations, we could observe patterns that are not noticeable while looking at the general results. Cases like CAN ID 100 help to obscure these patterns as both approaches have a very similar high performance. In some IDs we can see very high AUC differences between the detectors and notice that not in every single scenario autoencoders perform best. For example ID FB7, as seen in Figure B.7, is a single counterexample to some of the conclusions we have drawn on variability influence on the results. Alternatively, for IDs 1FB, 104, 116, shown in Figure B.8 ,B.11, and B.12 respectively, we can see great performance improvement when using an autoencoder since it can handle most of the cases in which the predictor performs quite badly. The trend among all IDs is favorable to our autoencoder approach.

6 CONCLUSIONS AND FUTURE WORK

In this thesis we presented a new approach to anomaly detection in sequences of automotive data using a completely unsupervised learning approach, where no attack data was used to either train or fine-tune the detector. This approach consisted of using an autoencoder, a deep neural network architecture used to learn the representation of CAN data sequences. To test the advantages of our approach with respect to other proposals, we compared it to a state-of-the-art anomaly detection system designed for automotive data. Since the implementation of this detector is not publicly available, we proceeded to implement it following the authors published research [37, 67, 66].

Our proposal was inspired by the research on anomaly detection using autoencoders, which had been proposed in other application areas. We adapted the approach to bring it into an automotive cybersecurity context, and demonstrating it yields significant performance improvements over the current state-of-the-art.

In this section we review our process and findings, recap of our contributions, and finally, lay out a path for possible future research on this topic.

6.1 PROCESS REVIEW

6.1.1 *CAN data*

The first step during this research was to analyze CAN traffic data in order to confirm the assumptions made in previous research, and also to obtain an overview of the data at hand. We began this process by collecting real CAN bus data from an Alfa Romeo Giulia Veloce. With this data we observed that CAN traffic can be characterized with the following features, as we showed in Section 2.3:

- The majority of CAN traffic is periodical.
- Traffic that is transmitted at high frequencies usually present a greater payload variability.
- Each ID creates new symbols constantly, resulting in a symbol dictionary that grows rapidly as more data is collected.

We also confirmed that there's underlying structure to most CAN ID payloads. To automatically find some potential structures, we replicated Markovitz and Wool field identification algorithm. The results from this classification were used in the process of training our detectors and designing our threat model. However, since we do not have the real semantics behind the CAN IDs it was impossible to test the accuracy of the algorithm.

6.1.2 *Predictor replication*

We followed by replicating prediction-based ADS proposed by Taylor et al. as it represents one of the state-of-the-art systems for CAN data sequence anomaly detection. Nonetheless, replication is quite a time consuming task as the original code is not available, and because some details are not immediately available. One of the most demanding tasks was the search for the best performing symbol output combination method, which yielded a considerably high number of results characteristics to consider, we ended up averaging these results and from this we chose the best combination method.

6.1.3 *Implementation of the autoencoder-based detector*

Based on previous work on anomaly detection using LSTM neural networks, we proposed a detection scheme with an autoencoder that learns to reconstruct normal traffic data. The autoencoder-based detector can be used to identify anomalies when a test sequence is not consistent with the statistical characteristics of what is considered legitimate traffic. We considered autoencoders as a more natural fit to our problem given the conditions that real attack samples are not available, which limits the choice of a detection scheme.

Through our experiments we observed that autoencoders are a suitable modeling technique for CAN traffic data, as it can successfully model the time-series and it does not make any assumptions on the data generating mechanism for each time-series. These assumptions include whether the signal is predictable in some way, or more generally, its stationarity.

With our approach we also presented a much simpler procedure to prepare the autoencoder for anomaly detection tasks. Once the autoencoder has been trained, we can set the anomaly scoring mechanism with nothing more than normal/legitimate data. The anomaly scoring mechanism is configured by learning the statistical distribution of reconstruction errors on non-anomalous data. The autoencoder receives as input sequence data from the dataset and outputs the respective reconstructions, the system then computes the reconstruction errors and fit these into a multivariate Gaussian distribution. This distribution corresponds to a ground truth on what to consider normal error scores. To evaluate if a given sequence is anomalous, the system measures the Mahalanobis distance from the reconstruction errors of a test sequence to the legitimate data error distribution.

6.1.4 *Detection performance*

In order to compare the predictor and autoencoder approaches, we established that both detectors are evaluated using the same performance metric. We used similar test scenarios and parameters for both detectors, these are also similar to the ones used in the the state-of-the-art detector research for two reasons: one was to replicate the original experiment with our own data, the second was to have common ground from which we can make a fair comparison on the detectors performance.

To test the detectors we need to simulate anomalies. To do so, we implemented a system that can manipulate existing CAN data and create a new set of anomalous sequences. Both detectors are tested on the same anomalous data.

Overall, the autoencoder detector showed better performance than the predictor detector. However, some of the cases where the autoencoder performed best were not immediately clear, as the process of averaging results over different variables and parameters can obfuscate some information. To obtain further insight on these results, we analyzed different levels of abstraction on the results and we were able to draw several conclusions. The most relevant one is that on average our autoencoder detector performs either equally or better than the state-of-the-art predictor system in most circumstances.

The most relevant observations from our detection results are:

- The autoencoder performs better on average when dealing with anomalies on low variability fields. We see that many cases where the predictor struggles are characterized by the fact that the anomaly in question targets low variability fields.

- For some IDs the predictor yields relatively poor performance at detecting replay field anomalies. These are anomalies in which a target field is replaced with a series of bits from another point in time. On the other hand, the autoencoder-based system performed better on average on this type of data field anomaly.

- Both detectors performance on high-variability fields is fairly similar, which gives us further incentive to propose the autoencoder as a substitute to the predictor rather than a complement. Figure B.12 in the Appendix illustrates a clear example of this phenomenon, the predictor performs well overall, however, all the cases where it performs badly (AUC reaching values as low as 0.55) are characterized only by the low-variability target fields.

In general our experiments showed that anomalies in medium-variability fields are the most difficult to detect for both detectors. However, as we discussed in Section 5.11, these results were likely caused by the fact that the target fields were always quite short in length with respect to other categories. Additionally, in our entire dataset only a handful of IDs had medium-variability fields, meaning that we likely didn't have a representative sample of these fields. To explain this phenomenon, we propose three hypothesis: not having a finely-tuned field identification algorithm made the detector more likely to misclassify mid-variability fields, the model needs more training data to better characterize the data, or, there are inherent problems in our implementation of the field identification algorithm.

6.2 CONTRIBUTIONS

We present two main contributions from this work. The first main contribution is proving that autoencoders are effectively good modelers of sequences CAN traffic data, even in the scenario where the data generating mechanism is completely unknown. Our approach does not make the same assumptions on the time-series characteristics we are modeling as the predictor approach does, that is, its stationary or predictability. Our approach also assumes no prior knowledge of attack data to successfully create a baseline of what constitutes normal/legitimate sequences. Our unsupervised approach managed to perform better than the semi-supervised predictor approach.

The second main contribution from our work was demonstrating that the autoencoder approach is better suited for anomaly detection tasks on CAN than a state-of-the-art predictor approach. In Section 5 we demonstrated the overall superior detection performance of our approach, we observed that on average the autoencoder covers most of the shortfalls of the predictor, and in the remaining cases both detectors perform similarly well. From these results we can confidently propose an autoencoder detection system as a potential substitute to predictor-based detection systems.

To conclude, we investigated the limitations of our approach. On most cases we identified potential ways to remediate these issues, for instance, some cases can be improved using more training data. Additionally, an improved field identification scheme could potentially improve the performance of the detection system as we would obtain a more accurate field structure. Nonetheless, some of the limitations of our approach in certain scenarios remain unclear, however, investigating these causes is a difficult task unless we have access to the semantics behind each CAN ID. Therefore, more work is needed to study these limitations.

6.3 FUTURE WORK

An important topic of future work is evaluating the appropriateness of our assumption on the errors distribution of normal CAN data.

Even though our approach does not make any assumptions on the CAN sequences, it does make a quite strong assumption regarding reconstruction errors. During the post-processing phase, we fit the reconstruction errors of legitimate data into a multivariate Gaussian distribution. Our results indicate that even with this assumption in place the results are promising, however, further work is needed to verify that these errors can in fact be modeled as a normal distribution, or alternatively propose new methods for modeling these multivariate errors.

Having some prior knowledge of the CAN signals semantics can also provide great insight regarding how to improve the detectors. Partial semantics availability can greatly improve the quality of the field identification algorithm, and even indicate us what fields are worth monitoring. Additionally, a system could choose the best approach (autoencoder or predictor) depending on the nature of the signal to be monitored.

Future research can also be done by testing different architectures for different IDs. Because of computing resources limitations, we performed the hyperparameter search of our autoencoder architecture using a single randomly chosen ID. A valuable future work consists of testing the autoencoder architecture for each CAN ID so that the neural network can be modeled for each signal individually, thus achieving better individual representations.

Testing different representational models is also an interesting path for future work. Autoencoders come in different presentations, some of which consist of using generative models, like Variational autoencoders [20]. The research of generative models could also be of great value as a way to effectively estimate a distribution density of the time series.

Another opportunity for future research consists on improving predictor-based detectors. Time-series forecasting has been an active area of research for many years, as presented in Section 3.5, new and interesting approaches to predict time-series have been proposed. Relatively newer models like neural networks with dilated convolutions have been shown to outperform RNNs in some forecasting tasks, and they are also significantly easier to train [45]. In cases where there is knowledge that some CAN signals are known to be predicable, an interesting approach would be to test these novel networks and evaluate their performance against LSTM predictors using automotive data.

Building an effective detection system can be done by combining different approaches. Automotive security issues cannot be solved using a single method. In the case of anomaly detection, we observe that there is not a necessarily "best method" to detect all kinds of anomalies. Effectively, frequency/timing-based detectors are more efficient in attack scenarios where the bus is constantly flooded with malicious traffic. As an additional measure we can make use of fingerprinting mechanisms to identify the attacker once the anomaly has been detected on the data stream, and if appropriate shut it off the network using mitigation approaches like Parrot 3.3.2. Firewalls can also be implemented in external facing interfaces as additional security measures. In conclusion, an effective solution requires putting together several detection schemes and security mechanisms so they can properly deal with the different manifestations of automotive cyber attacks.

Finally, a valuable path for future work is the design and implementation of actuator mechanisms to bring the vehicle into a safe state when an attack is detected. A possible solution is to trigger a "safe-mode" state, where the vehicle shuts down most non safe-critical components while instructing the driver to safely stop the vehicle, and it can also automatically contact the authorities or insurance company. Most of these are complex problems to solve, and additionally they would have to be deemed safe enough by the relevant authorities.
Our framing of the problem can be used to direct researchers to propose new or improved anomaly detection systems that work for the greater purpose of improving automotive security.

BIBLIOGRAPHY

[1] Majid S alDosari. “Unsupervised Anomaly Detection in Sequences Using Long Short Term Memory Recurrent Neural Networks.” PhD thesis. 2016 (cit. on pp. 4, 38–40, 42, 44, 54).

[2] Jason Andress. *The basics of information security: understanding the fundamentals of InfoSec in theory and practice*. Syngress, 2014 (cit. on p. 32).

[3] Jan Paul Assendorp. “Deep learning for anomaly detection in multivariate time series data.” PhD thesis. Hochschule für Angewandte Wissenschaften Hamburg, 2017 (cit. on pp. 42, 44, 54).

[4] Omid Avatefipour and Hafiz Malik. “State-of-the-art survey on in-vehicle network communication (CAN-Bus) security and vulnerabilities.” In: *arXiv preprint arXiv:1802.01725* (2018) (cit. on p. 1).

[5] Anastasia Borovykh, Sander Bohte, and Cornelis W Oosterlee. “Conditional time series forecasting with convolutional neural networks.” In: *arXiv preprint arXiv:1703.04691* (2017) (cit. on p. 43).

[6] CAN Bosch. “Specification version 2.0.” In: *Published by Robert Bosch GmbH (September 1991)* (1991) (cit. on pp. 1, 9).

[7] Sucheta Chauhan and Lovekesh Vig. “Anomaly detection in ECG time signals via deep long short-term memory networks.” In: *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE. 2015, pp. 1–7 (cit. on p. 43).

[8] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, Tadayoshi Kohno, et al. “Comprehensive Experimental Analyses of Automotive Attack Surfaces.” In: *USENIX Security Symposium*. San Francisco. 2011 (cit. on pp. 2, 3, 26).

[9] Kyong-Tak Cho and Kang Shin. “Viden: Attacker Identification on In-Vehicle Networks.” In: *arXiv preprint arXiv:1708.08414* (2017) (cit. on pp. 4, 36).

[10] Kyong-Tak Cho and Kang G Shin. “Error handling of in-vehicle networks makes them vulnerable.” In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2016, pp. 1044–1055 (cit. on p. 28).

[11] Kyong-Tak Cho and Kang G Shin. “Fingerprinting electronic control units for vehicle intrusion detection.” In: *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association. 2016, pp. 911–927 (cit. on pp. 4, 29, 35, 37).

[12] Wonsuk Choi, Hyo Jin Jo, Samuel Woo, Ji Young Chun, Jooyoung Park, and Dong Hoon Lee. "Identifying ECUs Using Inimitable Characteristics of Signals in Controller Area Networks." In: *arXiv preprint arXiv:1607.00497* (2016) (cit. on pp. 4, 36).

[13] Tsvika Dagan and Avishai Wool. "Parrot, a software-only anti-spoofing defense system for the CAN bus." In: *ESCAR EUROPE* (2016) (cit. on p. 34).

[14] Kien Do, Truyen Tran, and Svetha Venkatesh. "Energy-based anomaly detection for mixed data." In: *Knowledge and Information Systems* 57.2 (2018), pp. 413–435 (cit. on p. 43).

[15] Cheng Feng, Tingting Li, and Deeph Chana. "Multi-level anomaly detection in industrial control systems via package signatures and lstm networks." In: *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE. 2017, pp. 261–272 (cit. on p. 43).

[16] Pavel Filonov, Andrey Lavrentyev, and Artem Vorontsov. "Multivariate industrial time series with cyber-attack simulation: Fault detection using an lstm-based predictive data model." In: *arXiv preprint arXiv:1612.06676* (2016) (cit. on p. 43).

[17] Ugo Fiore, Francesco Palmieri, Aniello Castiglione, and Alfredo De Santis. "Network anomaly detection with the restricted Boltzmann machine." In: *Neurocomputing* 122 (2013), pp. 13–23 (cit. on p. 43).

[18] Ian Foster and Karl Koscher. "Exploring controller area networks." In: *login. USENIX Association* 40.6 (2015) (cit. on p. 13).

[19] Ian Foster, Andrew Prudhomme, Karl Koscher, and Stefan Savage. "Fast and vulnerable: a story of telematic failures." In: *9th {USENIX} Workshop on Offensive Technologies ({WOOT} 15)*. 2015 (cit. on pp. 14, 28).

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016 (cit. on pp. 5, 39, 42, 89).

[21] Nico Görnitz, Mikio Braun, and Marius Kloft. "Hidden markov anomaly detection." In: *International Conference on Machine Learning*. 2015, pp. 1833–1842 (cit. on p. 38).

[22] Andy Greenberg. "Hackers remotely kill a jeep on the highway - with me in it." In: *Wired* (2015) (cit. on pp. 2, 28).

[23] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory." In: *Neural computation* 9.8 (1997), pp. 1735–1780 (cit. on p. 40).

[24] Tobias Hoppe, Stefan Kiltz, and Jana Dittmann. "Security threats to automotive CAN networks—Practical examples and selected short-term countermeasures." In: *Reliability Engineering & System Safety* 96.1 (2011), pp. 11–25 (cit. on pp. 2, 27, 32).

[25] Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Tom Soderstrom. "Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding." In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM. 2018, pp. 387–395 (cit. on p. 43).

[26] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*. Vol. 112. Springer, 2013 (cit. on pp. 48, 57).

[27] Karl Henrik Johansson, Martin Törngren, and Lars Nielsen. "Vehicle applications of controller area network." In: *Handbook of networked and embedded control systems*. Springer, 2005, pp. 741–765 (cit. on pp. 9–11).

[28] Shrijit S Joshi and Vir V Phoha. "Investigating hidden Markov models capabilities in anomaly detection." In: *Proceedings of the 43rd annual Southeast regional conference-Volume 1*. ACM. 2005, pp. 98–103 (cit. on p. 38).

[29] Min-Joo Kang and Je-Won Kang. "Intrusion detection system using deep neural network for in-vehicle network security." In: *PloS one* 11.6 (2016), e0155781 (cit. on p. 37).

[30] Eamonn Keogh and Jessica Lin. "Clustering of time-series subsequences is meaningless: implications for previous and future research." In: *Knowledge and information systems* 8.2 (2005), pp. 154–177 (cit. on p. 38).

[31] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization." In: *arXiv preprint arXiv:1412.6980* (2014) (cit. on p. 62).

[32] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. "Experimental security analysis of a modern automobile." In: *2010 IEEE Symposium on Security and Privacy*. IEEE. 2010, pp. 447–462 (cit. on pp. 2, 3, 10, 25–27, 29).

[33] Hyunsung Lee, Seong Hoon Jeong, and Huy Kang Kim. "OTIDS: A Novel Intrusion Detection System for In-vehicle Network by using Remote Frame." In: (2017) (cit. on p. 35).

[34] Matan Levi, Yair Allouche, and Aryeh Kontorovich. "Advanced Analytics for Connected Cars Cyber Security." In: *arXiv preprint arXiv:1711.01939* (2017) (cit. on p. 38).

[35] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. "Intrusion detection system: A comprehensive review." In: *Journal of Network and Computer Applications* 36.1 (2013), pp. 16–24 (cit. on p. 2).

[36] Weixin Luo, Wen Liu, and Shenghua Gao. "Remembering history with convolutional LSTM for anomaly detection." In: *2017 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE. 2017, pp. 439–444 (cit. on p. 43).

[37] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. "Long short term memory networks for anomaly detection in time series." In: *Proceedings*. Presses universitaires de Louvain. 2015, p. 89 (cit. on pp. 43, 46, 49, 50, 63, 71, 85).

[38] Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. "LSTM-based encoder-decoder for multi-sensor anomaly detection." In: *arXiv preprint arXiv:1607.00148* (2016) (cit. on pp. 5, 44, 46, 54, 56–59, 62, 63, 71).

[39] Mirco Marchetti, Dario Stabili, Alessandro Guido, and Michele Colajanni. "Evaluation of anomaly detection for in-vehicle networks through information-theoretic algorithms." In: *Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI), 2016 IEEE 2nd International Forum on*. IEEE. 2016, pp. 1–6 (cit. on p. 37).

[40] Erik Marchi, Fabio Vesperini, Florian Eyben, Stefano Squartini, and Björn Schuller. "A novel approach for automatic acoustic novelty detection using a denoising autoencoder with bidirectional LSTM neural networks." In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2015, pp. 1996–2000 (cit. on p. 44).

[41] Moti Markovitz and Avishai Wool. "Field classification, modeling and anomaly detection in unknown CAN bus networks." In: *Vehicular Communications* 9 (2017), pp. 43–52 (cit. on pp. 21–23, 69, 85, 99).

[42] Charlie Miller and Chris Valasek. "A survey of remote automotive attack surfaces." In: *black hat USA* 2014 (2014) (cit. on pp. 3, 27).

[43] Charlie Miller and Chris Valasek. "Adventures in automotive networks and control units." In: *DEF CON* 21 (2013), pp. 260–264 (cit. on pp. 2, 3, 11, 27, 35).

[44] Charlie Miller and Chris Valasek. "Remote exploitation of an unaltered passenger vehicle." In: *Black Hat USA* 2015 (2015), p. 91 (cit. on pp. 2, 3, 27, 36).

[45] John Miller and Moritz Hardt. "When Recurrent Models Don't Need To Be Recurrent." In: *arXiv preprint arXiv:1805.10369* (2018) (cit. on p. 89).

[46] Pal-Stefan Murvay and Bogdan Groza. "Source identification using signal characteristics in controller area networks." In: *IEEE Signal Processing Letters* 21.4 (2014), pp. 395–399 (cit. on p. 36).

[47] Michael Müter and Naim Asaj. "Entropy-based anomaly detection for in-vehicle networks." In: *Intelligent Vehicles Symposium (IV), 2011 IEEE*. IEEE. 2011, pp. 1110–1115 (cit. on p. 37).

[48] Michael Müter, André Groll, and Felix C Freiling. "A structured approach to anomaly detection for in-vehicle networks." In: *Information Assurance and Security (IAS), 2010 Sixth International Conference on*. IEEE. 2010, pp. 92–98 (cit. on pp. 2, 34).

[49] Anvardh Nanduri and Lance Sherry. "Anomaly detection in aircraft data using Recurrent Neural Networks (RNN)." In: *2016 Integrated Communications Navigation and Surveillance (ICNS)*. IEEE. 2016, pp. 5C2–1 (cit. on p. 43).

[50] Sandeep Nair Narayanan, Sudip Mittal, and Anupam Joshi. "OBD_SecureAlert: An Anomaly Detection System for Vehicles." In: *Smart Computing (SMARTCOMP), 2016 IEEE International Conference on*. IEEE. 2016, pp. 1–6 (cit. on p. 38).

[51] Andrea Palanca, Eric Evenchick, Federico Maggi, and Stefano Zanero. "A stealth, selective, link-layer denial-of-service attack against automotive networks." In: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer. 2017, pp. 185–206 (cit. on pp. 28, 29, 36).

[52] Lei Pan, Xi Zheng, HX Chen, T Luan, Huzefa Bootwala, and Lynn Batten. "Cyber security attacks to modern vehicular systems." In: *Journal of information security and applications* 36 (2017), pp. 90–100 (cit. on pp. 26, 27).

[53] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks." In: *International conference on machine learning*. 2013, pp. 1310–1318 (cit. on p. 40).

[54] Muriel Pellissier. "Anomaly detection technique for sequential data." Theses. Université de Grenoble, Oct. 2013 (cit. on p. 43).

[55] Andreea-Ina Radu and Flavio D Garcia. "LeiA: A lightweight authentication protocol for CAN." In: *European Symposium on Research in Computer Security*. Springer. 2016, pp. 283–300 (cit. on p. 3).

[56] Martin Ring, Jürgen Dürrwang, Florian Sommer, and Reiner Kriesten. "Survey on vehicular attacks-building a vulnerability database." In: *2015 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*. IEEE. 2015, pp. 208–212 (cit. on p. 27).

[57] Sang Uk Sagong, Xuhang Ying, Andrew Clark, Linda Bushnell, and Radha Poovendran. "Cloaking the Clock: Emulating Clock Skew in Controller Area Networks." In: *arXiv preprint arXiv:1710.02692* (2017) (cit. on pp. 4, 35).

[58] Mayu Sakurada and Takehisa Yairi. "Anomaly detection using autoencoders with nonlinear dimensionality reduction." In: *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*. ACM. 2014, p. 4 (cit. on p. 44).

[59] Noräs Salman and Marco Bresch. "Design and implementation of an intrusion detection system (IDS) for in-vehicle networks." MA thesis. Chalmers University of Technology/University of Gothenburg, 2017 (cit. on p. 34).

[60] Craig Smith. *The car hacker's handbook: a guide for the penetration tester*. No Starch Press, 2016 (cit. on pp. 2, 9, 13).

[61] Hyun Min Song, Ha Rang Kim, and Huy Kang Kim. "Intrusion detection system based on the analysis of time intervals of CAN messages for in-vehicle network." In: *Information Networking (ICOIN), 2016 International Conference on*. IEEE. 2016, pp. 63–68 (cit. on p. 35).

[62] Yale Song, Zhen Wen, Ching-Yung Lin, and Randall Davis. "One-class conditional random fields for sequential anomaly detection." In: *Twenty-Third International Joint Conference on Artificial Intelligence*. 2013 (cit. on p. 38).

[63] Matthew William Spicer. "Intrusion Detection System for Electronic Communication Buses: A New Approach." MA thesis. Virginia Tech, 2018 (cit. on p. 36).

[64] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting." In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958 (cit. on p. 57).

[65] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks." In: (2014), pp. 3104–3112 (cit. on pp. 56, 59).

[66] Adrian Taylor. "Anomaly-based detection of malicious activity in in-vehicle networks." PhD thesis. Université d'Ottawa/University of Ottawa, 2017 (cit. on pp. 5, 19, 30, 38, 70, 73, 85).

[67] Adrian Taylor, Sylvain Leblanc, and Nathalie Japkowicz. "Anomaly detection in automobile control network data with long short-term memory networks." In: *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE. 2016, pp. 130–139 (cit. on pp. 4, 5, 30, 38, 43, 45, 46, 48, 49, 51–53, 56, 70, 75, 80, 85, 86).

[68] Adrian Taylor, Nathalie Japkowicz, and Sylvain Leblanc. "Frequency-based anomaly detection for the automotive CAN bus." In: *Industrial Control Systems Security (WCICSS), 2015 World Congress on*. IEEE. 2015, pp. 45–49 (cit. on p. 35).

[69] Vrizlynn LL Thing and Jiaxi Wu. “Autonomous vehicle security: A taxonomy of attacks and defences.” In: *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE. 2016, pp. 164–170 (cit. on p. 3).

[70] Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. “WaveNet: A generative model for raw audio.” In: *SSW* 125 (2016) (cit. on p. 43).

[71] Anthony Van Herrewege, Dave Singelee, and Ingrid Verbauwhede. “CANAuth-a simple, backward compatible broadcast authentication protocol for CAN bus.” In: *ECRYPT Workshop on Lightweight Cryptography*. Vol. 2011. 2011 (cit. on p. 3).

[72] Shuangfei Zhai, Yu Cheng, Weining Lu, and Zhongfei Zhang. “Deep structured energy based models for anomaly detection.” In: *arXiv preprint arXiv:1605.07717* (2016) (cit. on p. 43).

A IDENTIFIED FIELDS

In this appendix we present the results of the field identification task in our dataset. By implementing Markovitz and Wool field identification algorithm, as discussed in Section 2.4.1, we obtain for each ID a list of disjoint fields that make up the data field structure. These results are used for two purposes, one is determining the target fields when implementing data field anomalies, and the other is to have a better understanding of the IDs payload structure when evaluating their performance.

For each ID, we list the number of fields present, their respective type (either sensor multi-value or constant), its leftmost start bit, its length, its variability type (low, medium, or high variability), and finally the number of unique symbols indicated by column **# symbols**.

0FF	type	start	length	variability	# symbols
0	CONST	0	10	N/A	NaN
1	SENSOR	10	10	HIGH_VAR	514
2	CONST	20	1	N/A	NaN
3	SENSOR	21	10	HIGH_VAR	500
4	CONST	31	1	N/A	NaN
5	MULTI-VALUE	32	5	LOW_VAR	9
6	SENSOR	37	5	MID_VAR	32
7	CONST	42	10	N/A	NaN
8	SENSOR	52	12	HIGH_VAR	4096

Table A.1: List of identified fields for ID 0FF

0DE	type	start	length	variability	# symbols
0	MULTI-VALUE	0	6	LOW_VAR	11
1	SENSOR	6	10	HIGH_VAR	1024
2	CONST	16	3	N/A	NaN
3	SENSOR	19	13	HIGH_VAR	1015
4	CONST	32	4	N/A	NaN
5	SENSOR	36	12	HIGH_VAR	4096

Table A.2: List of identified fields for ID 0DE

1FB	type	start	length	variability	# symbols
0	SENSOR	0	11	HIGH_VAR	829
1	CONST	11	2	N/A	NaN
2	SENSOR	13	2	LOW_VAR	4
3	CONST	15	37	N/A	NaN
4	SENSOR	52	12	HIGH_VAR	4072

Table A.3: List of identified fields for ID 1FB

1FB	type	start	length	variability	# symbols
0	SENSOR	0	11	HIGH_VAR	829
1	CONST	11	2	N/A	NaN
2	SENSOR	13	2	LOW_VAR	4
3	CONST	15	37	N/A	NaN
4	SENSOR	52	12	HIGH_VAR	4072

Table A.4: List of identified fields for ID 1FB

100	type	start	length	variability	# symbols
0	CONST	0	2	N/A	NaN
1	SENSOR	2	11	HIGH_VAR	818
2	CONST	13	1	N/A	NaN
3	SENSOR	14	10	HIGH_VAR	514
4	CONST	24	1	N/A	NaN
5	SENSOR	25	10	HIGH_VAR	514
6	CONST	35	17	N/A	NaN
7	SENSOR	52	12	HIGH_VAR	4096

Table A.5: List of identified fields for ID 100

0EE	type	start	length	variability	# symbols
0	CONST	0	1	N/A	NaN
1	SENSOR	1	12	HIGH_VAR	2050
2	CONST	13	1	N/A	NaN
3	SENSOR	14	12	HIGH_VAR	2050
4	CONST	26	1	N/A	NaN
5	SENSOR	27	12	HIGH_VAR	2050
6	CONST	39	1	N/A	NaN
7	SENSOR	40	24	HIGH_VAR	209526

Table A.6: List of identified fields for ID 0EE

0FE	type	start	length	variability	# symbols
0	MULTI-VALUE	0	5	LOW_VAR	7
1	SENSOR	5	58	HIGH_VAR	1903
2	CONST	36	5	N/A	NaN
3	SENSOR	41	1	LOW_VAR	2
4	CONST	42	1	N/A	NaN
5	SENSOR	43	6	MID_VAR	34
6	CONST	49	3	N/A	NaN
7	SENSOR	52	12	HIGH_VAR	4096

Table A.7: List of identified fields for ID 0FE

101	type	start	length	variability	# symbols
0	SENSOR	0	4	LOW_VAR	6
1	CONST	4	3	N/A	NaN
2	SENSOR	7	12	HIGH_VAR	2050
3	CONST	19	2	N/A	NaN
4	SENSOR	21	9	HIGH_VAR	370
5	CONST	30	1	N/A	NaN
6	MULTI-VALUE	31	11	LOW_VAR	5
7	SENSOR	32	2	LOW_VAR	4
8	CONST	42	10	N/A	NaN
9	SENSOR	52	12	HIGH_VAR	4096

Table A.8: List of identified fields for ID 101

0FB	type	start	length	variability	# symbols
0	CONST	0	10	N/A	NaN
1	SENSOR	10	7	MID_VAR	53
2	CONST	17	1	N/A	NaN
3	SENSOR	18	31	HIGH_VAR	3515
4	CONST	38	1	N/A	NaN
5	SENSOR	39	1	LOW_VAR	2
6	CONST	40	12	N/A	NaN
7	SENSOR	52	12	HIGH_VAR	4096

Table A.9: List of identified fields for ID 0FB

104	type	start	length	variability	# symbols
0	MULTI-VALUE	0	5	LOW_VAR	11
1	SENSOR	5	8	HIGH_VAR	256
2	CONST	13	1	N/A	NaN
3	MULTI-VALUE	14	5	LOW_VAR	8
4	SENSOR	15	8	MID_VAR	68
5	MULTI-VALUE	25	11	LOW_VAR	9
6	CONST	36	16	N/A	NaN
7	SENSOR	52	12	HIGH_VAR	4094

Table A.10: List of identified fields for ID 104

116	type	start	length	variability	# symbols
0	SENSOR	0	52	HIGH_VAR	538013
1	MULTI-VALUE	34	6	LOW_VAR	10
2	CONST	40	12	N/A	NaN
3	SENSOR	52	12	HIGH_VAR	4096

Table A.11: List of identified fields for ID 116

11C	type	start	length	variability	# symbols
0	SENSOR	0	35	HIGH_VAR	4392
1	CONST	24	2	N/A	NaN
2	MULTI-VALUE	26	5	LOW_VAR	11
3	SENSOR	31	8	HIGH_VAR	256
4	CONST	39	13	N/A	NaN
5	SENSOR	52	12	HIGH_VAR	4096

Table A.12: List of identified fields for ID 11C

0FC	type	start	length	variability	# symbols
0	CONST	0	1	N/A	NaN
1	SENSOR	1	13	HIGH_VAR	3329
2	CONST	14	3	N/A	NaN
3	MULTI-VALUE	17	5	LOW_VAR	11
4	SENSOR	22	5	MID_VAR	32
5	CONST	27	2	N/A	NaN
6	SENSOR	29	2	LOW_VAR	3
7	CONST	31	21	N/A	NaN
8	SENSOR	52	12	HIGH_VAR	4096

Table A.13: List of identified fields for ID 0FC

B DATA FIELD ANOMALY DETECTION RESULTS

In this appendix we present the tables and figures representing the results of the anomaly detection tests targeting data fields.

B.1 AUC RESULTS

First we present the tables for anomaly tests AUC results that show the detection performance for every ID depending on the variability of the target field and the anomaly function used.

		Low var.					Mid var.					High var.				
		Max.	Min.	Constant	Random	Replay	Max.	Min.	Constant	Random	Replay	Max.	Min.	Constant	Random	Replay
0DE	Predictor	1.0000	1.0000	0.9993	1.0000	0.8774						1	0.94558	1	1	0.93895
	Autoencoder	1.0000	1.0000	0.9997	1.0000	0.9699						1	0.93677	1	1	0.96304
0EE	Predictor											1	0.94164	1	0.99998	0.98149
	Autoencoder											1	0.99719	0.99633	1	0.98915
0FB	Predictor						0.9983	0.9915	0.9234	0.9983	0.8094	0.99972	0.99995	0.99932	0.99623	0.96323
	Autoencoder						1.0000	1.0000	0.9979	1.0000	0.9987	1	1	1	1	1
0FC	Predictor	0.9977	0.9999	0.9897	0.9996	0.7288	0.7191	0.6000	0.7014	0.9892	0.6384	0.99637	0.98941	0.99877	0.99703	0.97501
	Autoencoder	1.0000	1.0000	0.9908	1.0000	0.9927	0.9250	0.6111	0.9101	0.9998	0.9920	1	1	1	1	1
0FE	Predictor	0.9817	0.9883	0.9877	0.9867	0.7512	0.9726	0.9854	0.9915	0.9920	0.5756	1	1	0.99999	1	0.99419
	Autoencoder	1.0000	1.0000	0.8913	1.0000	0.9892	1.0000	1.0000	0.7807	1.0000	0.8678	1	1	1	1	1
0FF	Predictor	0.9967	0.9969	0.9978	0.9977	0.8119	0.6349	0.6734	0.8582	0.9756	0.8190	0.93346	0.99887	0.99781	0.99798	0.93930
	Autoencoder	1.0000	1.0000	0.9992	1.0000	0.9689	0.5389	0.8838	0.7544	0.9058	0.8888	0.98247	1	0.90867	1	1
1F7	Predictor	0.9903	0.8783	0.9887	0.9859	0.8632	0.9735	0.9826	0.9581	0.9847	0.8492	0.63784	0.99232	0.9990	0.98531	0.76889
	Autoencoder	1.0000	0.5844	1.0000	1.0000	0.9814	0.7821	0.9343	0.9412	0.9954	0.8194	0.51298	1	1	1	0.74353
1FB	Predictor											0.98857	0.74986	0.75952	0.87268	0.61862
	Autoencoder											1	1	0.96584	1	0.96484
11C	Predictor	1.0000	0.9987	0.9973	1.0000	0.5940						1	0.99782	1	1	0.95527
	Autoencoder	1.0000	0.9992	0.8717	1.0000	0.8903						1	1	1	1	0.98822
100	Predictor											0.981614	0.99124	0.99899	0.99944	0.97422
	Autoencoder											0.99745	0.99882	0.99501	0.99933	0.92946
104	Predictor	1.0000	0.5821	1.0000	1.0000	0.5150	1.0000	0.8232	0.9908	1.0000	0.9803	1	0.50618	0.9965	0.99966	0.97854
	Autoencoder	1.0000	0.9990	1.0000	1.0000	0.9859	1.0000	0.8742	0.9999	1.0000	0.9945	1	0.50064	0.99872	1	1
116	Predictor	0.7016	0.9724	0.7949	0.9924	0.6487						0.99368	1	0.99980	1	0.99650
	Autoencoder	1.0000	0.9625	0.9927	1.0000	0.9410						0.999791	0.9943875	1	1	0.99854
Avg	Predictor	0.9585	0.9271	0.9694	0.9953	0.7238	0.8831	0.8427	0.9039	0.9900	0.7787	0.96094	0.92607	0.97915	0.98736	0.92368
	Autoencoder	1.0000	0.9431	0.9682	1.0000	0.9649	0.8743	0.8839	0.8974	0.9835	0.9269	0.95772	0.95231	0.98871	0.99994	0.96473

Table B.1: AUC results grouped by variability (low and medium) and anomaly function. Blank cells indicate absence of a relevant field.

B.2 AUC RESULTS PER ID

In this Section we illustrate the anomaly detection test results for all the 12 IDs through bar graphics. The vertical axis represents the AUC ranging form 0.5 to 1.

The parameters used for anomaly detection where:

- **Anomaly function**. Four different anomaly functions are considered: maximum (max_value), minimum value (min_value), constant (constant_value), random (random_value), and replay (replay_field).
- **Duration**. Three different field categories are tested with 4 values [0.2, 0.5, 1, 1.5], in seconds.
- **Field categories**: low variability (LOW_VAR), medium variability (MID_VAR), and high variability (HIGH_VAR)

Next, we describe the meaning of the labels in the x axis. The label string is composed by concatenating
Anomaly function : *duration* : *field category*

For example, the following string: `replay_field:0.2_s:LOW_VAR` represents a test using a replay anomaly function, with a duration of 0.2 seconds, and the variability of the target field.

These figures allow for a better analysis of the common factors that differentiate the performance of the autoencoder and predictor detectors.

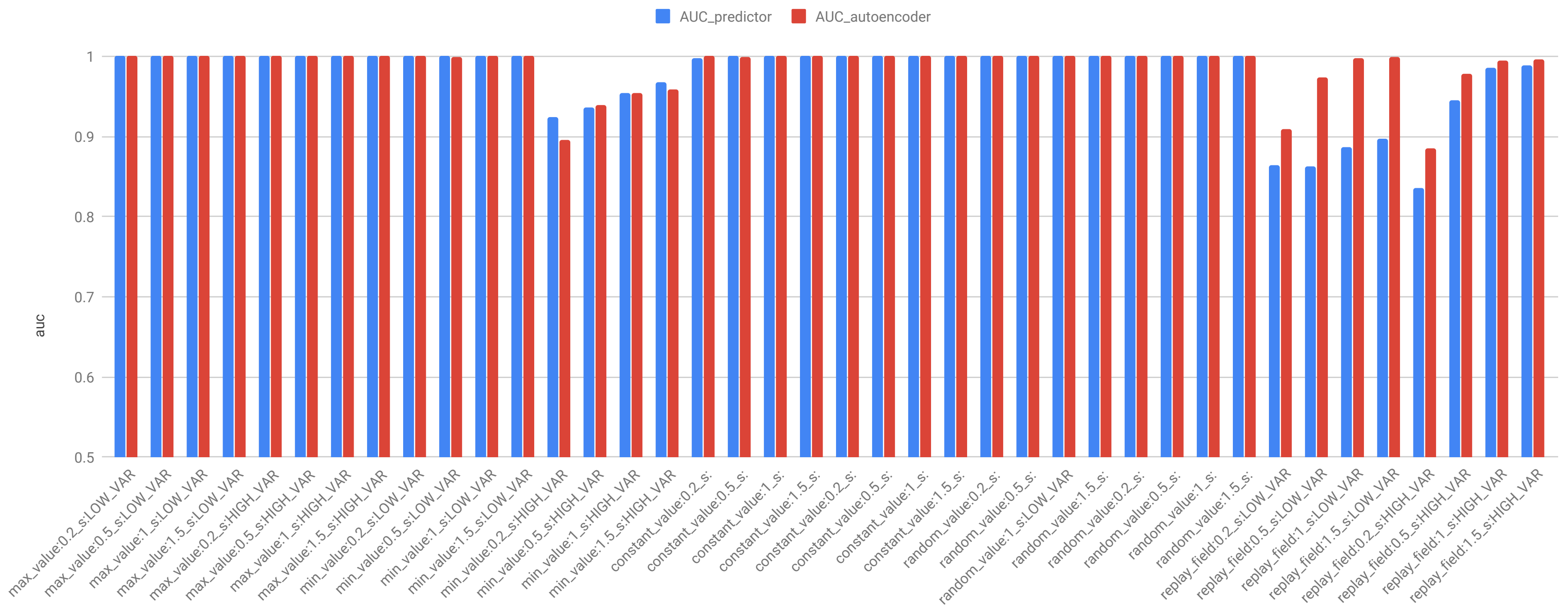


Figure B.1: AUC results for ID oDE.

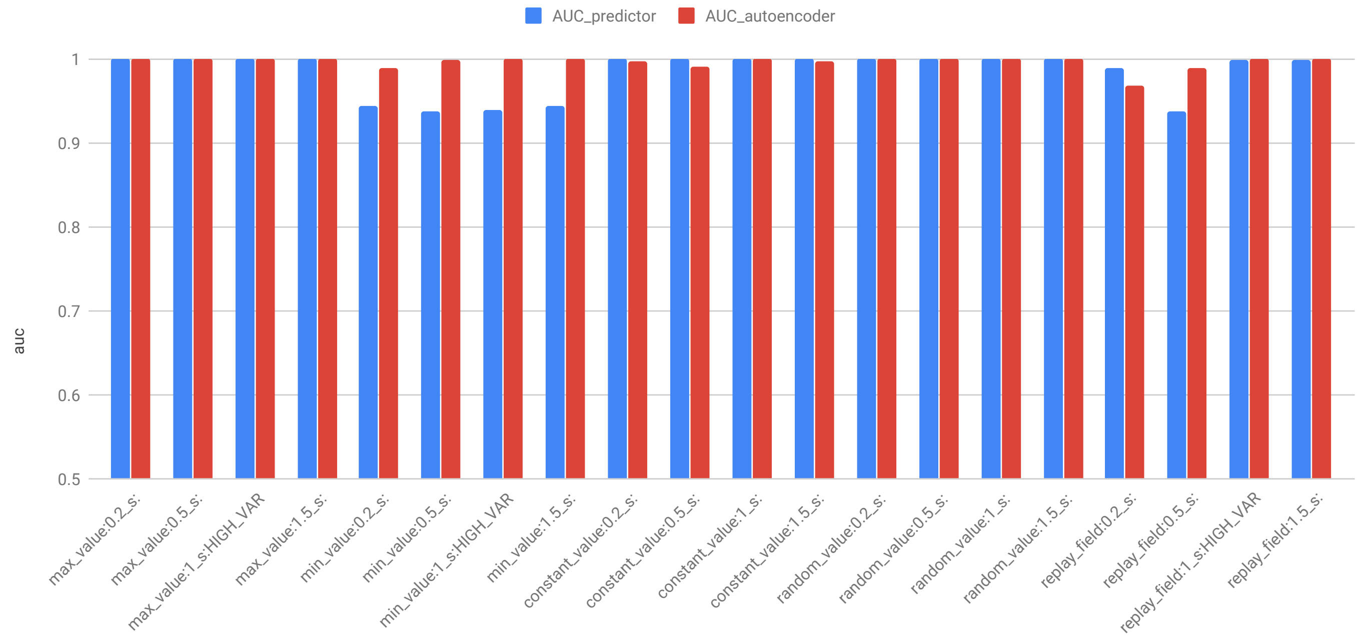


Figure B.2: AUC results for ID oEE.

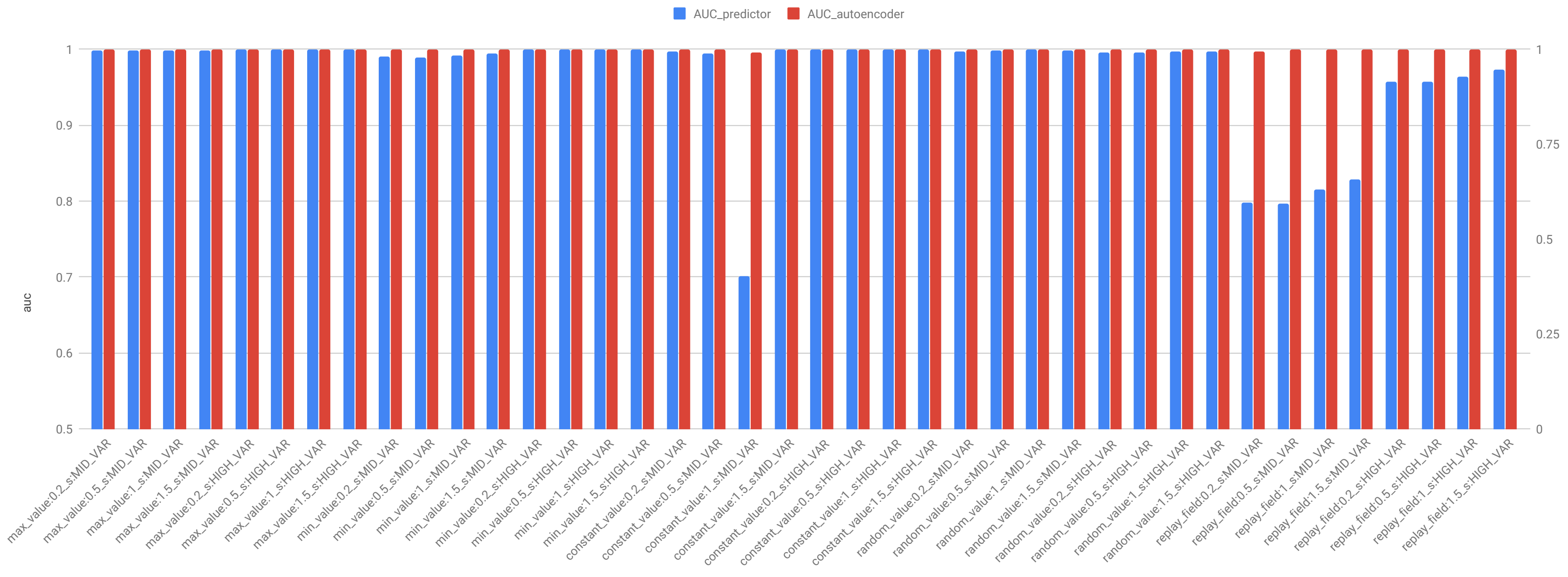


Figure B.3: AUC results for ID oFB.

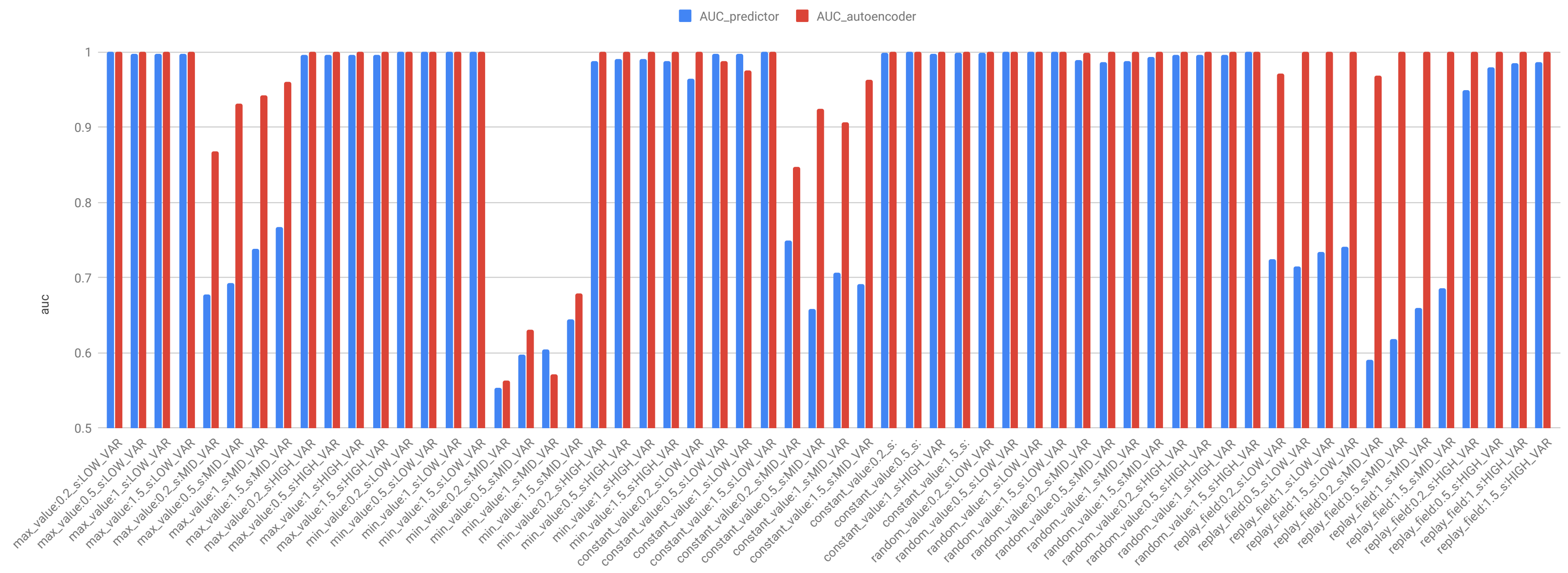


Figure B.4: AUC results for ID oFC.

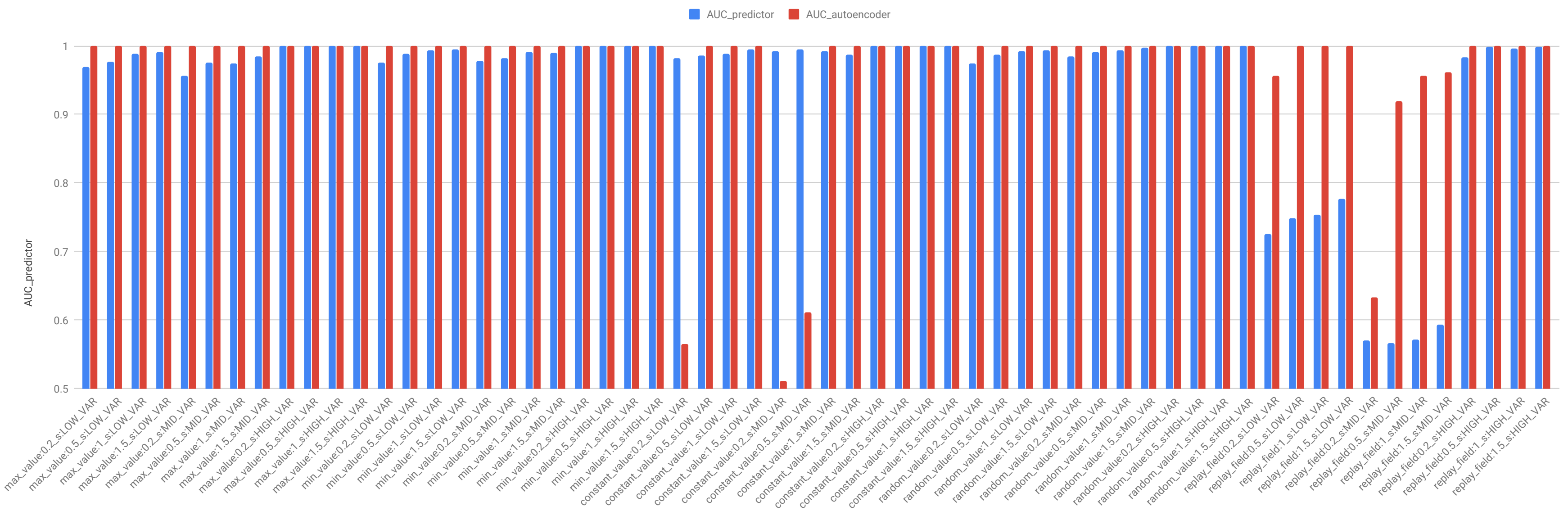


Figure B.5: AUC results for ID oFE.

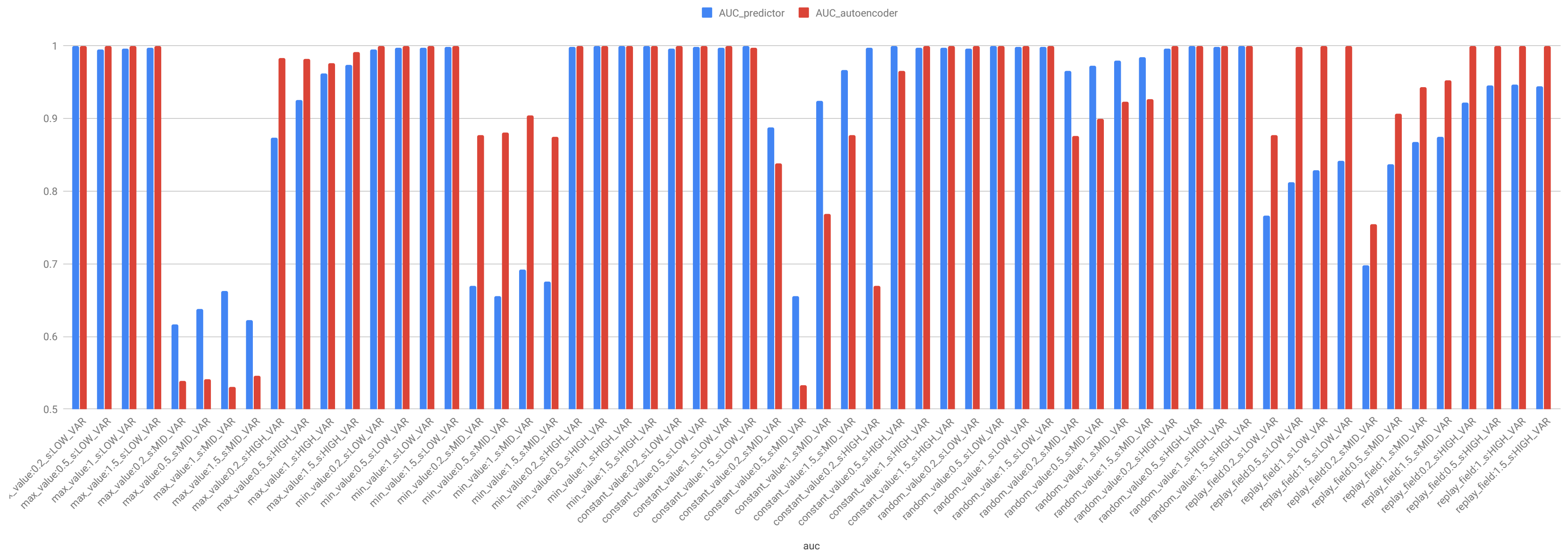


Figure B.6: AUC results for ID oFF.

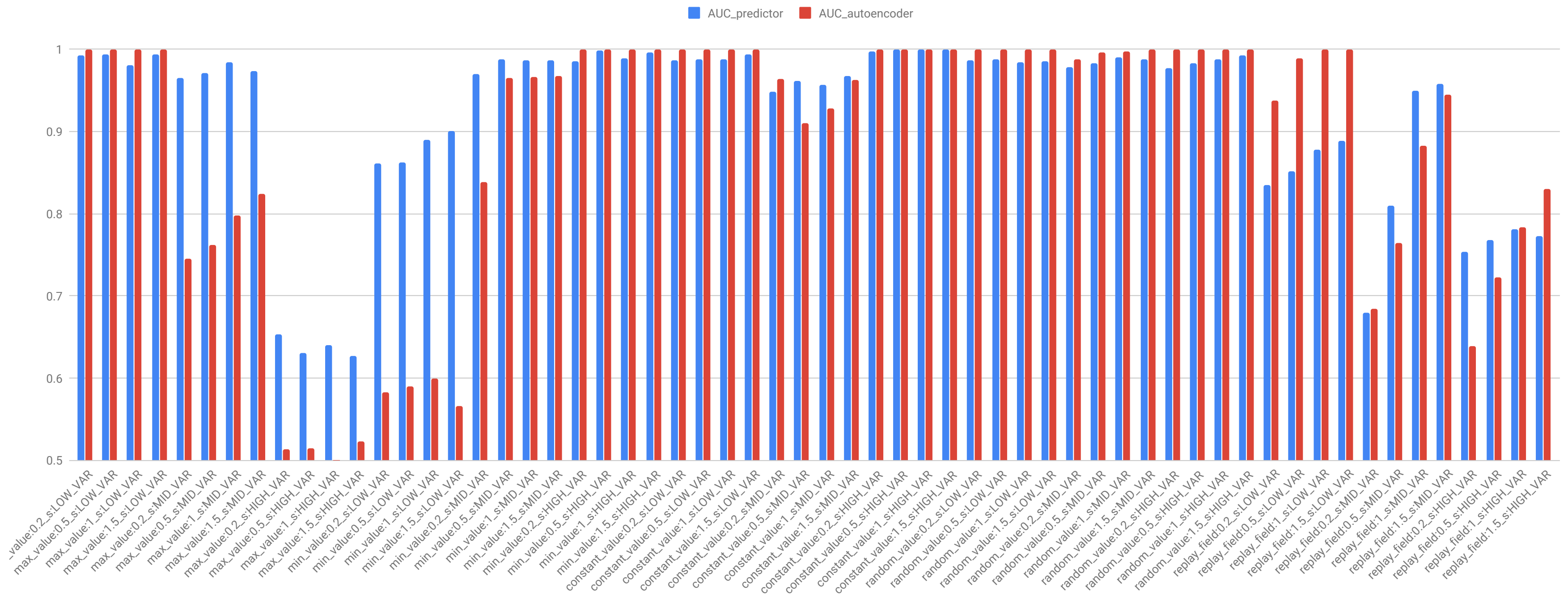


Figure B.7: AUC results for ID 1F7.

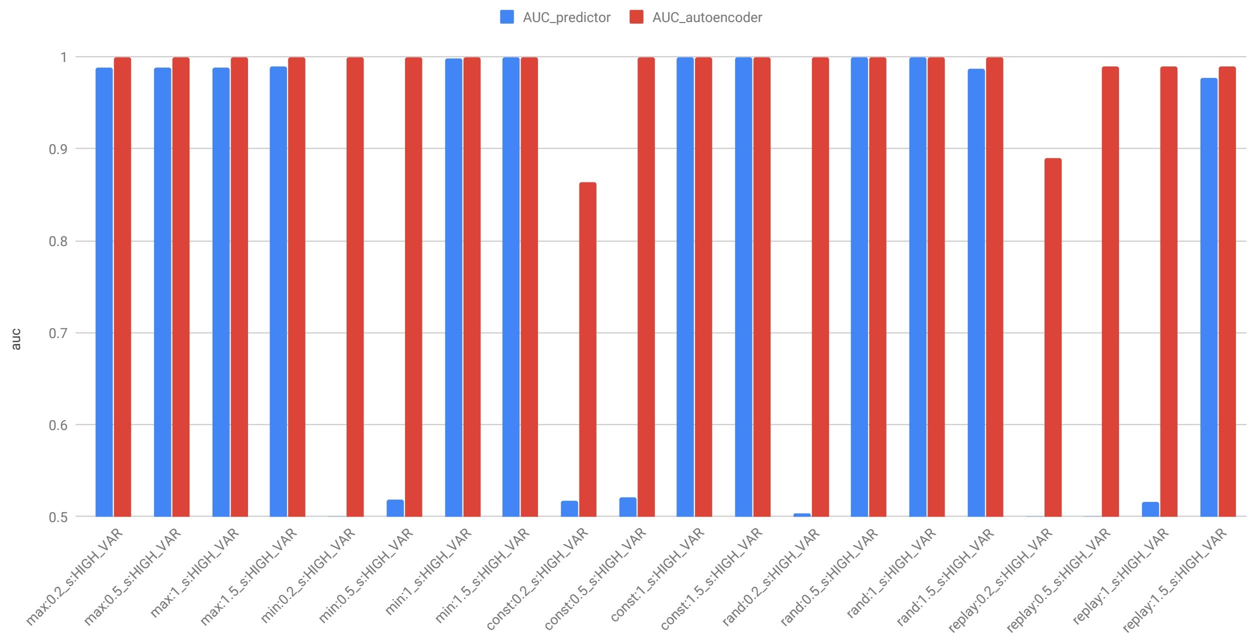


Figure B.8: AUC results for ID 1FB.

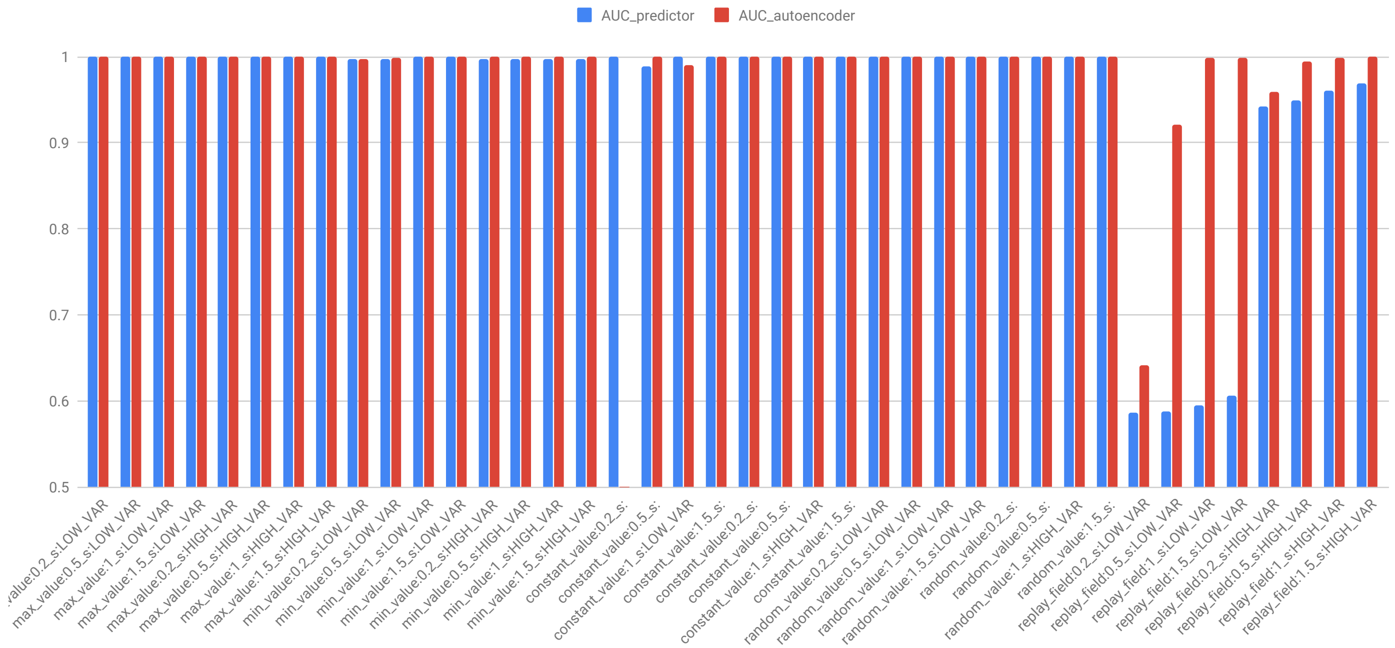


Figure B.9: AUC results for ID 11C.

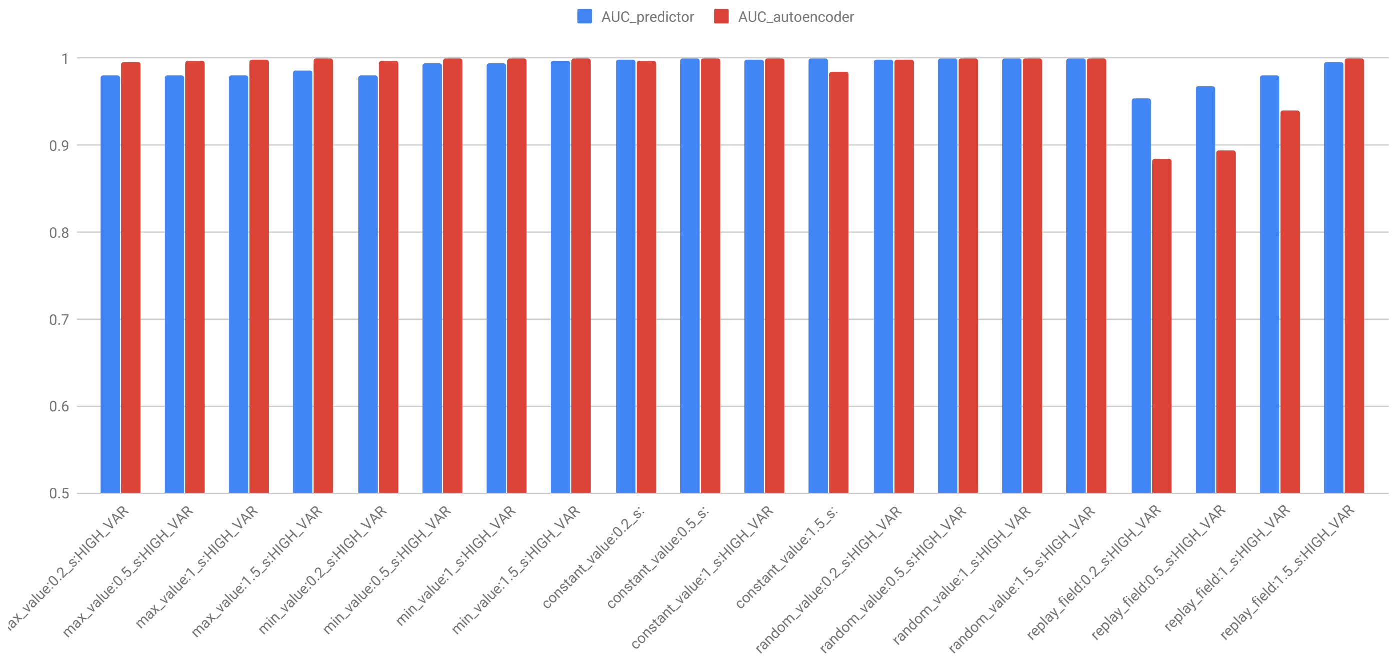


Figure B.10: AUC results for ID 100.

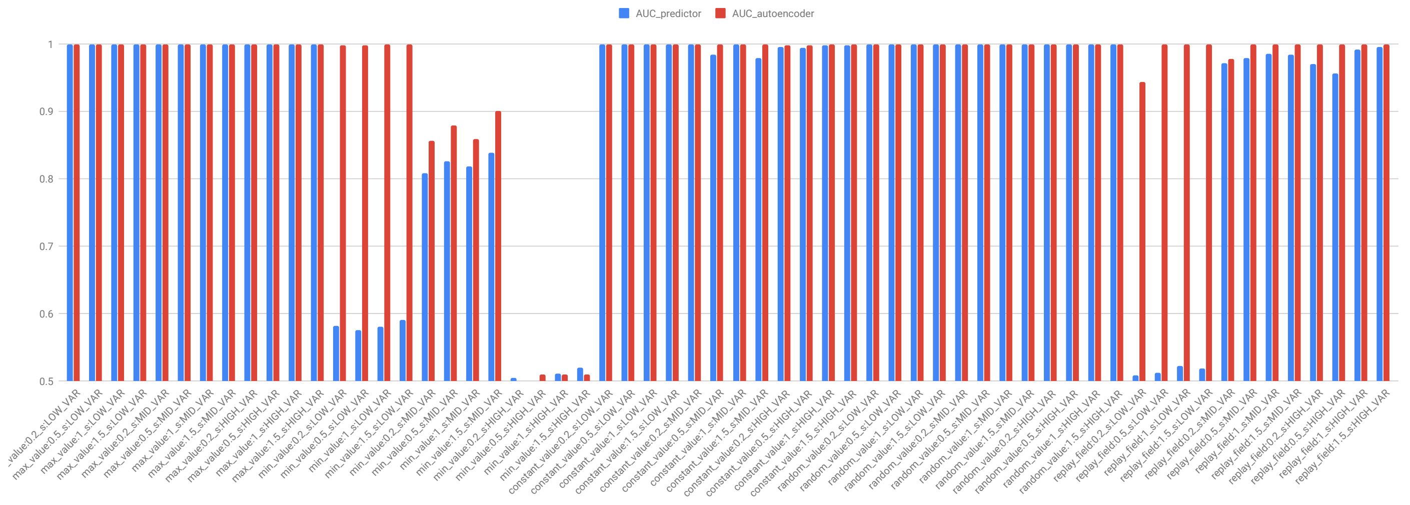


Figure B.11: AUC results for ID 104.

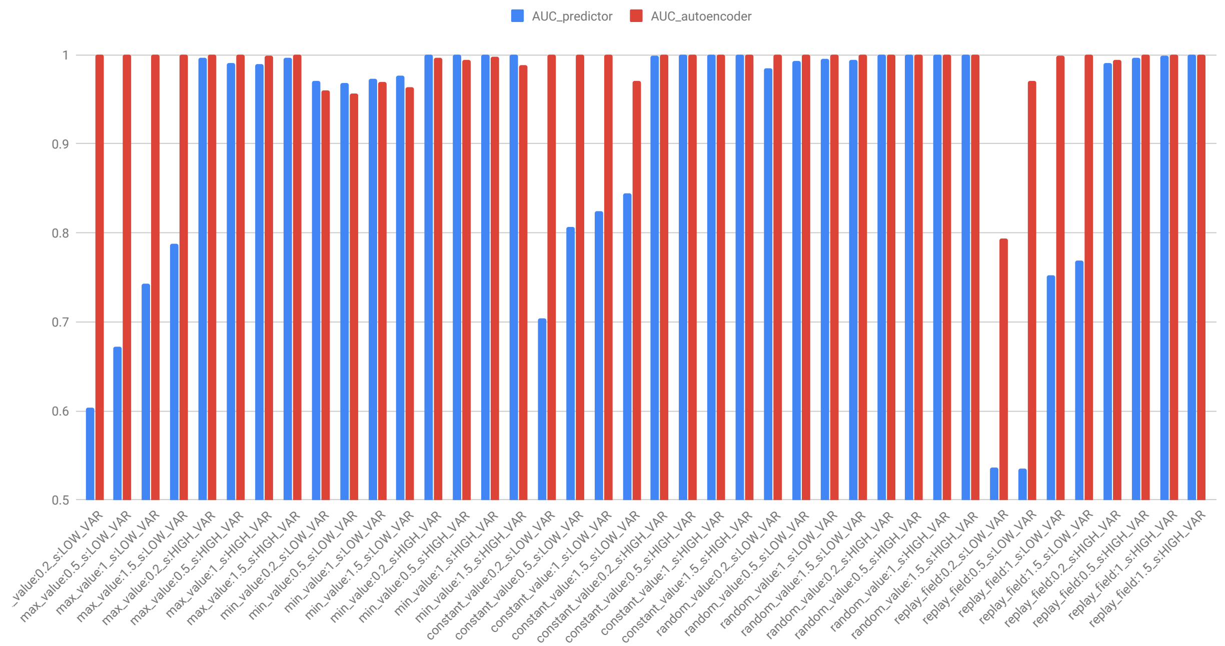


Figure B.12: AUC results for ID 116.