Real-time Lightweight Intrusion Detection System for In-Vehicle Network

by Said M. Rachidi

B.S. in Electrical Engineering, May 2000, University of District of Columbia
Master of Business Administration, May 2005, George Washington University

A Praxis submitted to

The Faculty of
The School of Engineering and Applied Science
of The George Washington University
in partial fulfillment of the requirements
for the degree of Doctor of Engineering

August 31, 2024

Praxis directed by

John M. Fossaceca, Ph.D.
Professorial Lecturer of Engineering Management and Systems Engineering

Shahryar Sarkani, D.Sc
Adjunct Professor of Engineering and Applied Science

The School of Engineering and Applied Science of The George Washington University certifies that Said M. Rachidi has passed the Final Examination for the degree of Doctor of Engineering as of July 26, 2024. This is the final and approved form of the Praxis.

Real-time Lightweight Intrusion Detection System for In-Vehicle Network

Said M. Rachidi

Praxis Research Committee:

Dr. John Fossaceca, Professorial Lecturer of Engineering Management and Systems Engineering, Praxis Director

Dr. Shahryar Sarkani, Adjunct Professor of Engineering and Applied Science, Praxis Co-Director

Dr. Muhammad Islam, Professorial Lecturer of Engineering Management and Systems Engineering, Committee Chair

© Copyright 2024 by Said M. Rachidi
All rights reserved

Dedication

The author wishes to dedicate this research to the memory of his beloved grandmother, Lala Zayneb, whose wisdom and love have always been a guiding light in his life. Though she is no longer with us, her spirit continues to inspire him every day.

To his father, Mohammed, whose legacy of strength and perseverance lives on in his heart. His guidance and love have shaped the author into the person he is today.

To his mother, whose unwavering belief in his potential has been a constant source of motivation. Her dream of seeing him achieve academic excellence has driven him to strive for the highest levels of education, culminating in this milestone.

To his beloved wife, Asmaa, whose love and unwavering support have been his bedrock. From the very beginning of this journey, she has stood by his side, offering endless encouragement and understanding. Her patience and sacrifice have allowed him to focus on his studies, knowing that their family was well taken care of. Asmaa's faith in his abilities has been a constant source of strength, particularly during the challenging preparation for his praxis. He is profoundly grateful for her unwavering belief in him, and this accomplishment is a testament to her love and dedication.

To his children, Zayneb, Sarah, and Ali, who are his greatest sources of inspiration. Their smiles and boundless energy give him the strength to keep pushing forward, even in the face of adversity. This achievement is dedicated to them as a symbol of the endless possibilities that lie ahead.

Acknowledgments

The author extends heartfelt gratitude to Dr. John Fossaceca for his exceptional support and guidance throughout the duration of the Doctor of Engineering program. Dr. Fossaceca's insightful advice and encouragement were invaluable, and his ability to direct the author toward the right path at critical moments greatly contributed to the success of this research.

Special thanks are also due to Dr. Shahryar Sarkani for his unwavering support during the Praxis proposal phase. Dr. Sarkani's expertise and constructive feedback were essential in guiding this research to completion.

The author would also like to extend special appreciation to Dr. Islam, the Committee Chair, for his leadership and support. His guidance and feedback were crucial in refining and finalizing this work.

Abstract of Praxis

Real-Time Lightweight Intrusion Detection System (IDS) for In-Vehicle Network

The evolution of automotive technology has seen a rapid shift towards highly connected and autonomous vehicles, which rely heavily on intricate intra-vehicle networks. This network, primarily managed by the Controller Area Network (CAN) bus, coordinates communication among numerous electronic control units (ECUs) that control various functions within the vehicle. Despite its widespread use and efficiency, the CAN bus protocol lacks fundamental security features, making it vulnerable to cyberattacks that can lead to unauthorized control and disruption of vehicle systems.

This praxis presents the development and evaluation of a real-time, lightweight Intrusion Detection System (IDS) specifically designed for in-vehicle networks. The primary focus is creating a solution that not only detects and classifies malicious activities with high accuracy but also meets the stringent real-time and resource constraints typical of automotive environments. Unlike previous studies that primarily focus on detection accuracy, this research also emphasizes detection time, which must be less than two milliseconds to be considered real-time (Avatefipour et al., 2019) , and model size, which should be under 1MB to be considered lightweight in this context (Rolfsmeier et al., 2003) to ensure the IDS can be practically deployed within the limited computational resources of a vehicle.

The research utilizes the Car-Hacking Dataset, a comprehensive compilation of normal and attack data generated from CAN bus traffic, to train and evaluate the IDS. This praxis explores and compares the performance of three machine learning models:

Random Forest (RF), eXtreme Gradient Boosting (XGBoost), and Bidirectional Long Short-Term Memory (Bi-LSTM), with a particular emphasis on balancing detection accuracy, processing speed, and model size. The XGBoost model emerged as the superior choice, offering the best balance of high accuracy, low false alarm rate, and minimal detection latency, making it ideal for real-time applications in automotive networks. The proposed IDS improves upon existing solutions by addressing both the performance and practical deployment challenges in the context of in-vehicle networks.

This research advances the field of automotive cybersecurity by providing a robust system for real-time CAN bus intrusion detection, paving the way for enhanced security measures in modern vehicles. Future work could extend this system to include more advanced detection techniques and the broader scope of inter-vehicle network communications, further strengthening vehicle security against evolving cyber threats.

Table of Contents

Dedication ..... **iv**

Acknowledgments ..... **v**

Abstract of Praxis ..... **vi**

Table of Contents ..... **viii**

List of Figures ..... **xii**

List of Tables ..... **xiii**

List of Symbols ..... **xiv**

List of Acronyms ..... **xv**

Chapter 1 —Introduction **1**

1.1 Background 1

1.2 Research Motivation 3

1.3 Problem Statement 4

1.4 Thesis Statement 4

1.5 Research Objectives 5

1.6 Research Questions and Hypotheses 5

1.6.1 Research Questions 5

1.6.2 Research Hypotheses 6

1.7 Scope of Research 6

1.8 Research Limitations 6

1.9 Organization of Praxis 7

Chapter 2 —Literature Review **8**

2.1 Introduction 8
2.2 In-Vehicle Networks 8
2.3 In-Vehicle Networks Cybersecurity Vulnerabilities 13
2.4 In-Vehicle Networks Common Cyberattacks 14
2.5 Well-Known Recent IVN Attacks 15
2.6 CAN BUS Intrusion Detection Systems for IVN 17
2.6.1 Signature-based IDS 17
2.6.2 Anomaly-based IDS 18
2.7 Summary and Conclusion 25
Chapter 3 —Methodology 27
3.1 Introduction 27
3.2 Dataset 28
3.2.1 Dataset Acquisition 28
3.2.2 Dataset Structure 29
3.3 Exploratory Data Analysis 31
3.4 Data Preprocessing 32
3.4.1 Data Cleaning and Filtering 33
3.4.2 Categorical Data Encoding 33
3.4.3 Data Standardization 33
3.4.4 Data Splitting into Training, Validation, and Testing Datasets 34
3.4.5 Data Balancing 35
3.5 Machine Learning Models 36
3.5.1 Random Forest 36

3.5.2 eXtreme Gradient Boosting 39
3.5.3 BiLSTM 40
3.6 ML Model Development 41
3.6.1 Model Hyperparameter Tuning 42
3.6.2 Model Performance Metrics 43
3.6.3 Model Performance Validation and Testing 45
Chapter 4 —Results 48
4.1 Introduction 48
4.2 Exploratory Data Analysis 49
4.2.1 Feature Overview 49
4.2.2 Feature Detailed Statistics 50
4.2.3 Data Cleaning 54
4.3 Data Preprocessing 55
4.3.1 Categorical Data Encoding 56
4.3.2 Data standardization 56
4.3.3 Data balancing 57
4.3.4 Feature selection 57
4.4 Model Performance Comparison 59
4.4.1 Hyperparameter Tuning 60
4.4.2 Random Forest Hyperparameter Tuning 60
4.4.3 Random Forest Performance Metrics 61
4.4.4 XGBoost Hyperparameter Tuning 63
4.4.5 XGBoost Performance Metrics 63

4.4.6 BiLSTM Hyperparameter Tuning 65
4.4.7 BiLSTM Performance Metrics 65
4.5 Summary and Conclusion 67
4.6 Comparison with other published research papers 68
Chapter 5 —Discussion and Conclusions 71
5.1 Discussion 71
5.1.1 Research Question 1 72
5.1.2 Research Question 2 72
5.1.3 Research Question 3 73
5.2 Conclusion 74
5.3 Contribution to Body of Knowledge 75
5.4 Recommendations for Future Research 76
References 78
Appendix A 86

List of Figures

Figure 1-1. CAN bus Architecture. 2

Figure 2-1. CAN Data Frame 11

Figure 3-1. Random Forest Structure 37

Figure 3-2. Boosting Ensemble-based Learning Model 39

Figure 3-3. Neural Network Architecture of BiLSTM 41

Figure 4-1. Non-Hexadecimal Values in Data Columns 55

Figure 4-2. Feature Importance 59

Figure A-1. Learning Curve 1 86

Figure A-2. Learning Curve 2 87

Figure A-3. Loss Curve 87

Figure A-4. Feature Correlation Matrix 88

List of Tables

Table 1-1. CAN Bus Attack Types 3

Table 3-1. Dataset Structure 29

Table 3-2. Model Hyperparameters 42

Table 4-1. Random Forest Hyperparameters 61

Table 4-2. Random Forest Performance Metrics 61

Table 4-3. XGBoost Hyperparameters 63

Table 4-4. XGBoost Performance Metrics 64

Table 4-5. BiLSTM Hyperparameters 65

Table 4-6. BiLSTM Performance Metrics 66

Table 4-7. Summary Results 68

Table 4-8. Results Comparison 69

List of Symbols

p_i Probability of class i

j Total number of classes in the dataset.

$F_m(x_i)$ Boosed model at the m-th iteration

$F_{m-1}(x_i)$ Model obtained up until the previous iteration

$f_m(x_i)$ Current decision tree being added to the model

List of Acronyms

Adam	Adaptive Moment Estimation
ANN	Artificial Neural Networks
AUC	Area Under the Curve
AVs	Autonomous Vehicles
Bi-LSTM	Bidirectional Long Short-Term Memory
CAN	Controller Area Network
CVs	Connected Vehicles
DLC	Data Length Code
DNN	Deep Neural Networks
DoS	Denial of Service
DT	Decision Tree
ECU	Electronic Control Unit
FAR	False Alarm Rate
FN	False Negative
FP	False Positive
GAN	Generative Adversarial Network
GB	Gradient Boosting
GPS	Global Positioning System
GRU	Gated Recurrent Unit
HCRL	Car Hacking and Countermeasure Research Lab
IDS	Intrusion Detection System

IF	Isolation Forest
IoV	Internet of Vehicles
IVN	Intra-Vehicle Network
KNN	K-Nearest Neighbors
ML	Machine Learning
MTH-IDS	Multi-Threshold Intrusion Detection System
NaN	Not a Number
OBD-II	On-Board Diagnostics II
OCSVM	One-Class Support Vector Machine
RF	Random Forest
RNN	Recurrent Neural Networks
ROC	Receiver Operating Characteristic
SMOTE	Synthetic Minority Oversampling Technique
SVM	Support Vector Machine
V2V	Vehicle-to-Vehicle
V2X	Vehicle-to-Everything
XGBoost	eXtreme Gradient Boosting

Chapter 1—Introduction

1.1 Background

In recent years, automotive technology has undergone a fast and disruptive transition from mechanically driven vehicles to advanced connected vehicles (CVs) and autonomous vehicles (AVs) enabled by the rapid development of the Internet of Vehicles (IoV) technology (Siegel et al., 2018). IoV is made possible by two primary networks: the intra-vehicle network (IVN), which operates within the vehicle, and the external vehicular network, which connects the car and its IVN to external systems (Nanda et al., 2019). The IVN depends on an increasing number of electronic control units ECUs (70 to 100 per vehicle). Each is responsible for performing a specific functionality, such as airbag deployment, changing gear, or activating an antilock braking system. ECUs within a vehicle are assigned unique addresses each and are connected via a Controller Area Network (CAN) bus to send and receive messages to execute assigned tasks (Bosch, 2014).

The CAN bus protocol was designed by Robert Bosch in 1986 to allow microcontrollers and devices to communicate with each other within a vehicle without the need for a central computer (Robert, 1991). The goal at that time was to minimize wiring complexity and improve vehicle efficiency, enabling ECUs to communicate with one another over a two-wire network and minimizing the need for a separate wire for each device. It was designed for robust intra-vehicle communication, but it lacks inherent security features like encryption or authentication, primarily because it was developed at a time when cybersecurity was not a significant concern in automotive design (Bozdal et

al., 2020). Even with the potential security vulnerabilities by design, the CAN bus protocol is still widely utilized on modern cars due to its simplicity, low cost, and real-time communication (Zeng et al., 2016).

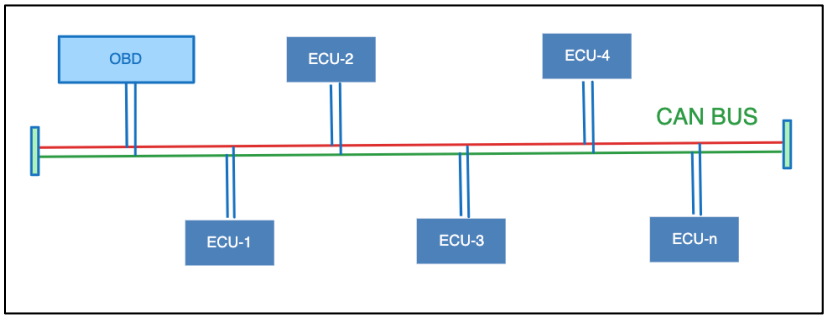


Figure 1-1. CAN bus Architecture.

As depicted in Figure 1-1 above, data within the CAN bus travels in a multi-master, message-oriented fashion.

Each node also known as an Electronic Control Unit (ECU) on the network can send a message when the bus is free. The non-destructive arbitration ensures that the message with the highest priority gets transmitted first in case of simultaneous transmission. However, this openness makes the CAN bus inherently vulnerable to attacks (Oladimeji et al., 2023). Since the CAN protocol lacks native security features, malicious actors can inject or manipulate messages, leading to unauthorized control or disruption of vehicle functions.

While the connectivity of modern vehicles brings numerous benefits, it also introduces significant cybersecurity risks. The potential attack surface of the CAN bus system is increasing over time. It includes physical access points like the On-Board Diagnostics (OBD-II) port, wireless interfaces like Bluetooth and Wi-Fi, and external network connections like cellular or GPS communications (Zeng et al., 2016). Each access point presents a potential vulnerability that could be exploited to cause

consequences ranging from minor inconveniences to critical safety risks. Table 1-1 below list the different possible CAN bus attack types.

Table 1-1. CAN Bus Attack Types

Attack Type	Definition
Message flooding	CAN bus is overwhelmed with messages, causing a denial of service
Message spoofing	Attackers impersonate a legitimate ECU to send false data
Message Injection	Attackers send unauthorized messages to control vehicle functions

1.2 Research Motivation

Over the last decade, several automotive cyber-attacks have been reported (Kukkala et al., 2022). The first major remote hack was demonstrated in 2015 by security researchers who successfully attacked a 2014 Jeep Cherokee infotainment system sending CAN messages to vehicle ECUs via 4G LTE (Miller & Valasek, 2015). More recently, a T-BONE attack on a Tesla Model 3 using a drone made headlines (Deng et al., 2022). Attackers could remotely control car acceleration, seat positions, and several other functions. These two examples and many other reported attacks would not have succeeded if the CAN bus had not had inherent security vulnerabilities. For this reason, vehicles now more than ever need a reliable intrusion detection system (IDS) that can be deployed in real-time to protect the CAN bus and connected ECUs from cyberattacks.

This praxis presents a novel approach to developing a machine learning-based real-time CAN bus Intrusion Detection System (IDS) that is both effective and efficient.

Unlike previous research that primarily emphasizes detection accuracy, this praxis also considers detection time, which must be less than two milliseconds to be deemed real-time (Avatefipour et al., 2019), and model size, which should remain under 1MB to ensure it is lightweight and suitable for deployment within a vehicle's constrained computational resources (Rolfsmeier et al., 2003). This praxis examines several configurations of different models to develop the most optimized configuration. Testing the proposed machine learning model in a simulated environment demonstrates enhanced performance with a low detection time compared to existing models.

1.3 Problem Statement

As vehicles become more software-defined and more connected, they are increasingly vulnerable to cyber-attacks and thefts (Kukkala et al., 2022), where a single cyber incident could cost an automaker up to $1.1 billion.

Current modern vehicles depend on several onboard software systems to run. They contain more than 100 million lines of code and are projected to exceed 300 million in the near future (Rajapaksha et al., 2023).Vehicles' increased automation and connectivity make them more vulnerable to numerous cyberattacks. In 2015, for the first time in history, Fiat Chrysler Automotive (FCA) recalled 1.4 million vehicles due to cybersecurity vulnerabilities (Lampe & Meng, 2023).

1.4 Thesis Statement

A real-time and lightweight machine learning-based intrusion detection system (IDS) is required to protect connected vehicles against cybersecurity attacks.

Developing a machine learning-based IDS that can detect cyberattacks against the CAN bus in real-time will shield modern cars from both physical and remote cyberattacks.

1.5 Research Objectives

The primary objective of this research is to address the security vulnerabilities of the CAN bus by developing a working solution that can be deployed in the real world on each car to protect it from cyber-attacks. To achieve this objective, a machine learning-based predictive intrusion detection model was developed and tested. This model aims to:

1. Ensure minimal latency: Detect and respond to cyber threats without significant delay, thereby not impacting the vehicle's performance or safety-critical functions.
2. Enhance prediction accuracy: Improve the accuracy in predicting potential attacks, utilizing advanced algorithms and machine learning techniques to effectively distinguish between normal operations and anomalous, potentially malicious activities.

1.6 Research Questions and Hypotheses

This research addresses three main questions, all related to the possibility of developing a highly accurate predictive model that is be efficient and can be deployed in real-time with low detection time.

1.6.1 Research Questions

RQ1: Which features will most accurately identify and predict an attack on the in-vehicle CAN network?

RQ2: Can a lightweight real-time intrusion detection system be developed to detect cyber-attacks on in-vehicle CAN network with an accuracy of over 95%?

RQ3: Which of eXtreme gradient boosting, Random Forest, and Artificial Neural Networks can most accurately and efficiently (in terms of model size and latency) detect in-vehicle network attacks in real-time?

1.6.2 Research Hypotheses

H1: "CAN ID" feature of all features in the datasets are best for detecting in-vehicle network attacks.

H2: A real-time and lightweight IDS can be utilized to detect in-vehicle network cyber-attacks with an accuracy rate of over 95%.

H3: eXtreme gradient boosting model will most accurately and efficiently (in terms of model size and latency) detect an attack compared to Random Forest and Artificial Neural Networks models.

1.7 Scope of Research

The scope of this research centers on enhancing the cybersecurity of modern vehicles, explicitly targeting the security vulnerabilities within the intra-vehicle network. It focuses on the data traveling through the CAN bus inside the vehicle. This research does not address attacks against the inter-vehicle networks that could occur outside the CAN BUS. To carry out the research, the praxis uses the Car-Hacking Dataset for Intrusion Detection Dataset, which consists of 17,558,462 labeled records of normal and malicious CAN bus traffic.

1.8 Research Limitations

This research is not performed on actual cars. We relied on the dataset collected from CAN bus traffic to simulate the real-time communication traffic between ECUs within the vehicle network.

1.9 Organization of Praxis

This praxis is organized into five chapters. Chapter 1 is the introduction that provides the background information on the research topic. It includes the problem and thesis statements, the research motivation, objectives, questions, hypothesis, scope, and limitations. Chapter 2 presents a literature review of previous studies on this praxis's in-vehicle network intrusion detection research domain. Chapter 3 describes the research methodology, data sources, data preprocessing, feature engineering, and analytical techniques used in the praxis. Chapter 4 presents the results of the praxis research. Chapter 5 summarizes the outcome of this praxis, its key findings, and contributions to the research field. It also provides recommendations for future research.

Chapter 2—Literature Review

2.1 Introduction

This chapter provides a general overview of the literature in the field of in-vehicle network intrusion detection systems. The goal is to provide a context for the research in this praxis. This background information provides an understanding of the problem addressed in the research and insight into the challenges researchers face in this domain. Section 2.2 of this chapter covers a general overview of the in-vehicle networks (IVN). Section 2.3 discusses IVN cybersecurity vulnerabilities. Section 2.4 covers IVN most common cyberattack types. Section 2.5 presents some well-known recent IVN attacks. Section 2.6 provides an overview CAN BUS Intrusion Detection Systems for IVN. This chapter concludes with Section 2.7 providing a general summary and a conclusion highlighting the reason for this praxis to develop a real time intrusion detection for IVN.

2.2 In-Vehicle Networks

In-vehicle networks constitute the critical infrastructure for communication among the various systems within an automobile, facilitating the seamless transmission of information. Central to these networks are the Electronic Control Units (ECUs), which serve as the sophisticated control modules overseeing distinct vehicular functions. Each ECU operates as an advanced computational unit that manages and monitors a range of operations from engine regulation to the deployment of airbags, and the modulation of braking systems. These ECUs are interconnected through a complex network that allows them to communicate and collaborate, ensuring the vehicle functions in a cohesive and safe manner. The data exchanged over this network enables real-time coordination and

control, which is essential for optimizing vehicle performance and ensuring passenger safety. To enable the diverse functionalities and performance demands of each ECU within the network, the automotive industry employs a range of IVN protocols, each with its unique features and applications. The primary protocols include FlexRay, Media Oriented Systems Transport (MOST), Local Interconnect Network (LIN), and Controller Area Network (CAN) (Smith, 2016):

- **FlexRay**: is a sophisticated in-vehicle network system designed for modern automotive needs, providing high-speed data transmission up to 10 Mbps (Makowitz & Temple, 2006). It excels in supporting safety-critical functions like brake-by-wire with its deterministic communication and dual-channel redundancy, enhancing system reliability. However, FlexRay's complexity and higher implementation costs compared to simpler systems can pose challenges for integration and widespread adoption in the automotive industry (Groza & Murvay, 2019).
- **MOST (Media Oriented Systems Transport)** is a high-speed multimedia network technology used in vehicles to integrate infotainment, audio, video, and telematics. It supports streaming of high-quality multimedia content and data with minimal latency and synchronization issues. However, its specialization in infotainment systems means it's not suited for controlling critical vehicle functions, and the cost of implementation can be higher compared to basic audio and video systems, potentially limiting its use to luxury or high-end vehicles (Frenzel, 2016).

- **The Local Interconnect Network (LIN)** is a cost-effective, low-speed network protocol used in vehicles for managing simple devices like mirrors, wipers, and window controls. LIN is designed to complement higher-speed networks like CAN by handling less critical communication tasks, reducing overall system costs. However, its limited data transmission speed and capacity restrict its use to simpler, non-critical functions, making it unsuitable for more complex or safety-critical applications in vehicles (Evers & Pannell, 2022).
- **The Controller Area Network (CAN)** is a robust, standard communication protocol widely adopted in the automotive industry for connecting various electronic control units (ECUs) within a vehicle. It facilitates reliable and efficient transmission of data between components such as the engine control unit, transmission, airbags, and more, without a central computer. CAN is renowned for its high fault tolerance, real-time capabilities, and efficient handling of complex networks with minimal wiring requirements, making it integral for critical vehicle functions and diagnostics (Avatefipour et al., 2019).

  Compared to other vehicle network protocols like LIN and MOST, CAN offers a superior balance between speed, reliability, and cost-effectiveness (Yaji et al., 2020). While LIN is optimized for simpler, low-speed tasks and MOST for high-bandwidth multimedia applications, CAN provides a versatile backbone capable of managing essential and complex vehicle operations. Its widespread acceptance and use across the automotive industry have also led to a rich ecosystem of tools and support, further enhancing its value. Despite its lower data rate compared to newer technologies like FlexRay, CAN's proven reliability and extensive industry

adoption make it a fundamental element in vehicle architecture, especially for safety-critical communications (Yaji et al., 2020).

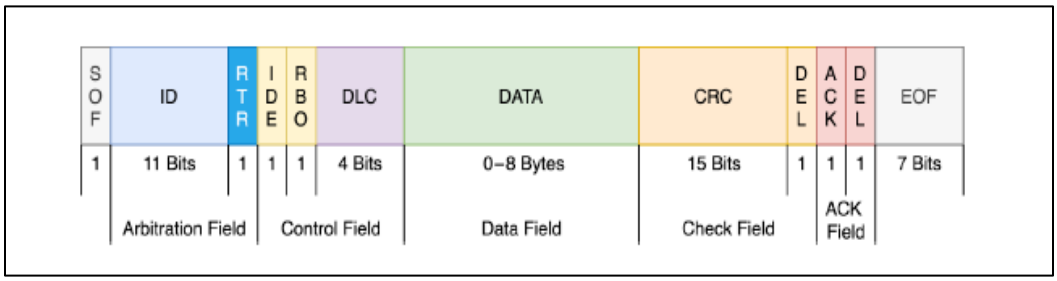


Figure 2-1. CAN Data Frame.

The CAN bus data frame is a structured unit of data that facilitates communication between various electronic control units (ECUs) within a vehicle's network. The design of the CAN data frame is ingeniously crafted to ensure efficient, reliable, and real-time data exchange crucial for automotive functionalities. Figure 2-1 shows the data frame structure, it consists of seven fields:

1. **Start of Frame (SOF):** A single dominant bit that signals the beginning of a data frame, ensuring that all nodes on the network prepare to receive data.
2. **Arbitration Field**: This includes the unique identifier (ID) for the message, which also determines the message's priority on the network. Lower ID numbers have higher priority. The field also contains the Remote Transmission Request (RTR) bit, differentiating between data frames (dominant bit) and request frames (recessive bit).

3. **Control Field**: This part specifies the format of the frame (standard or extended) and the data length code (DLC), which indicates the number of bytes in the payload (ranging from 0 to 8 bytes).
4. **Data Field**: The core of the frame, carrying the actual data payload intended for the receiving ECUs. Its size is variable, up to 8 bytes, depending on the DLC.
5. **CRC Field**: Contains a cyclic redundancy check code to allow the receiving nodes to detect any errors in the transmitted frame. This is followed by a CRC delimiter of one recessive bit.
6. **Acknowledge Field (ACK):** Comprises an ACK slot and an ACK delimiter. The transmitting node monitors this slot to verify if at least one receiver has correctly received the frame, indicated by a dominant bit in the ACK slot.
7. **End of Frame (EOF):** Marks the end of a CAN frame with seven recessive bits, ensuring a clear separation between consecutive frames.

In addition to the above seven fields, a sequence of bits, called Interframe Space, follows the EOF to provide a gap before the next frame starts, allowing for error checking and processing.

The CAN data frame's architecture is engineered to offer a robust and adaptable means of communication. It is essential for a wide array of automotive applications, ranging from straightforward sensor data reporting to intricate control commands. This structured data transmission approach is crucial for the safe and efficient operation of modern vehicles, allowing electronic control units (ECUs) to exchange information in real-time.

2.3 In-Vehicle Networks Cybersecurity Vulnerabilities

In-vehicle networks, the backbone of modern vehicle communication systems, have evolved significantly with advancements in automotive technology. However, as vehicles become more connected and sophisticated, cybersecurity vulnerabilities within these networks have emerged as a critical concern (Li et al., 2018). The integration of numerous electronic control units (ECUs), sensors, and actuators through the Controller Area Network (CAN) has expanded the in-vehicle network attack surface (Li et al., 2018). Despite its widespread use and reliability in vehicle communication, the CAN bus exhibits several security vulnerabilities that make it susceptible to cyberattacks. These weaknesses stem from inherent design choices that prioritize efficiency and simplicity over security, leading to potential risks. The following is a list of some of this design decisions that is impacting its security (Avatefipour et al., 2019):

- **Lack of Authentication**: Any ECU on the network can impersonate another, sending messages without a system to verify the sender's identity. This opens the door for unauthorized message injection.
- **Broadcast Domain Vulnerability**: The CAN operates as a broadcast domain, where messages sent are received by all nodes. This allows a compromised node to monitor all network communications, leading to potential data breaches.
- **Absence of Encryption**: To keep communication swift, CAN messages are sent unencrypted. This decision, while beneficial for speed, makes it easy for cyber attackers to intercept and analyze messages, leading to sniffing attacks.
- **ID-Based Priority Exploitation**: Messages are prioritized based on their ID values, with lower IDs taking precedence. Attackers can exploit this by flooding

the network with low-ID messages, causing a Denial-of-Service (DoS) attack and disrupting vehicle operations.

2.4 In-Vehicle Networks Common Cyberattacks

Within the realm of in-vehicle network security, three prevalent types of cyberattacks pose significant threats to the integrity and functionality of automotive systems: fuzzing, Denial-of-Service (DoS), and impersonation attacks. These attacks exploit specific vulnerabilities in-vehicle communication systems to disrupt operations, compromise the safety, and undermine user privacy.

Fuzzing Attacks

Fuzzing involves bombarding the in-vehicle network with random, malformed, or unexpected data packets to uncover security vulnerabilities. This technique tests the network's resilience against abnormal inputs, aiming to cause erratic system behavior or crashes. By identifying weaknesses, attackers can craft more sophisticated and targeted attacks, potentially gaining unauthorized access to vehicle controls and sensitive data.

Denial-of-Service (DoS) Attacks

DoS attacks target the vehicle's network by overwhelming it with a flood of messages or commands, consuming bandwidth and processing resources to the point where legitimate communication cannot occur. This can incapacitate critical vehicle functions, from navigation and infotainment systems to essential safety mechanisms, posing direct risks to vehicle operability and passenger safety. In extreme cases, DoS attacks could render a vehicle completely inoperable, causing a shutdown of all electronic systems.

Impersonation Attacks

Impersonation attacks, also known as *spoofing*, occur when attackers send forged messages that appear to originate from legitimate sources within the vehicle's network. By masquerading as a trusted electronic control unit (ECU), attackers can manipulate the behavior of the vehicle, issuing unauthorized commands or extracting confidential information. This type of attack can lead to a wide range of adverse effects, including unauthorized access, manipulation of vehicle settings, or the triggering of unintended actions that could compromise safety.

To mitigate these threats, it is essential to deploy an intrusion detection system (IDS) to identify and respond to suspicious activities in real time; protecting against fuzzing, DoS, and impersonation attacks is crucial for maintaining the reliability, safety, and privacy of modern vehicles, ensuring they remain resilient against cyber threats.

2.5 Well-Known Recent IVN Attacks

In 2015, researchers Charlie Miller and Chris Valasek executed a groundbreaking remote hack of a Jeep Cherokee, exploiting weaknesses in its Uconnect internet-connected system (Miller, 2019). They gained control over the vehicle’s critical functions such as steering and brakes by hacking into the car's Controller Area Network (CAN) bus system, which facilitates communication between various electronic control units (ECUs). Their demonstration notably included remotely slowing down the Jeep on a highway, showcasing the potential dangers of automotive cybersecurity breaches. In response, Fiat Chrysler Automobiles recalled 1.4 million vehicles to fix the identified software flaws. The gravity of this event punctuated the critical demand for fortified

cybersecurity protocols in the automotive sphere, spotlighting the imperative to shield connected vehicles from such exploitable weaknesses.

Several other notable car hacks have highlighted vulnerabilities in modern vehicles, emphasizing the growing importance of cybersecurity in automotive design. Here are a few similar incidents:

1. **Tesla Model S (2016):** Researchers demonstrated multiple times the ability to remotely hack into Tesla Model S vehicles. In 2016, a team from Keen Security Lab managed to remotely control the brakes, trunk, and other features (Car Crash Computing, 2017)
2. **Nissan Leaf (2016):** Security researchers discovered a vulnerability in the NissanConnect app that allowed them to remotely access the Nissan Leaf's climate control systems and access data about the vehicle's recent activities without authentication (Fingas, 2016).
3. **BMW ConnectedDrive (2015):** Researchers from ADAC (Allgemeiner Deutscher Automobil-Club) identified a flaw in BMW's ConnectedDrive system that could allow attackers to remotely unlock the doors of up to 2.2 million vehicles, including BMW, Mini, and Rolls Royce models (Prince, 2015).
4. **Volkswagen Key Fob Hack (2016):** A team of researchers found a way to clone Volkswagen key fobs by exploiting weaknesses in the car's keyless entry system. This vulnerability potentially affected millions of vehicles sold over the past 20 years (Greenberg, 2016).
5. **Mercedes-Benz E-Class (2017):** Researchers demonstrated a way to remotely exploit a vulnerability in the infotainment system of the Mercedes-Benz E-Class,

allowing them to control the car's steering wheel and dashboard functions (Paganini, 2020).

These incidents collectively illustrate the critical need for robust cybersecurity measures in the automotive industry. As vehicles become increasingly connected and reliant on digital technologies, the potential for cybersecurity threats grows, making it imperative for manufacturers to prioritize and enhance the security of their vehicles.

2.6 CAN BUS Intrusion Detection Systems for IVN

Intrusion Detection Systems (IDS) are classified into two primary types: signature-based and anomaly-based, distinguished by their detection methodologies. Signature-based IDS excel in accurately identifying known threats, resulting in a minimal rate of false positives. Nevertheless, their critical limitation lies in the inability to recognize new, previously unrecorded attacks. On the other hand, anomaly-based IDS are adept at detecting new forms of attacks by monitoring for deviations from established patterns of normal behavior (Müter et al., 2010).

2.6.1 Signature-based IDS

Signature-based CAN bus intrusion detection systems focus on identifying specific patterns or signatures that match known threats, attacks, or malicious activities within the CAN network. These systems rely on a known signatures derived from previously identified and analyzed threats. The effectiveness of a signature-based IDS hinges on the comprehensiveness and update frequency of its signature database. Two prominent approaches within this framework are specification-based and fingerprint-based detection, each offering unique strengths in identifying intrusions:

Specification-based IDS: Operates on predefined sets of rules and specifications related to CAN bus protocols and ECUs behavior. By monitoring network traffic against these specifications, the IDS can detect deviations indicative of intrusions, such as unexpected messages or unauthorized commands (Larson et al., 2008).

Fingerprint-based IDS: Identifies intrusions by analyzing the unique characteristics or "fingerprints" of ECUs. Each ECU has a behavioral fingerprint, including timing and frequency of messages. Any variation from this fingerprint could signal a potential security breach. Researchers have developed various techniques to capture these fingerprints from ECUs, employing methods such as clock-based (Shin & Cho, 2022) and voltage measurement-based intrusion detection (Cho & Shin, 2017). These unique fingerprints enable the differentiation between authorized and unauthorized access to the network's ECUs. Cho and Shin introduced a technique named Viden, which identifies the unique signature of ECUs through the voltage levels observed in their transmitted messages (Cho & Shin, 2017). Viden operates by learning the acknowledgment patterns and voltage profiles from the transmitting ECUs, creating a detailed profile for each one (Cho & Shin, 2017) his approach allows for the swift and precise identification of malicious nodes within the network. However, it's important to note that Viden is specifically tailored for the physical layer of network security, without extending its detection capabilities to the application layer.

2.6.2 Anomaly-based IDS

Anomaly-based CAN bus intrusion detection systems are designed to identify unusual activities that deviate from the normal operation within a vehicle's Controller Area Network (CAN). These systems are crucial for automotive cybersecurity, as they

provide a dynamic method of detecting potential intrusions by monitoring for behaviors that are out of the ordinary, rather than relying on known patterns of attacks. This approach is particularly effective against novel or sophisticated threats that have not been previously identified or cataloged. Anomaly-based detection systems analyze the network traffic in real-time, employing various methodologies to discern between normal and potentially malicious activities on the CAN bus. Anomaly-based IDS can be statistical-based, frequency-based, or machine learning-based.

Statistical-based IDS: During the initial phase of integrating anomaly detection techniques into the vehicle's Controller Area Network (CAN), the primary focus of research was on creating statistical models to understand the behavior of the CAN bus. This was done by observing the consistent frequency of CAN IDs within the messages transmitted. Electronic Control Units (ECUs) on the CAN bus are designed to send messages at regular intervals, which include information on one or more sensors within the vehicle. In 2015, researchers developed an approach to anomaly detection based on the analysis of average and standard deviation of the time intervals between packets in normal and attack conditions (Taylor et al., 2015). They assessed the effectiveness of their method by examining the Hamming distance across successive message data fields and combining this with the timing differences, although these measures proved to be unreliable for identifying anomalies in message sequences. Their evaluation was performed on data from a 2011 Ford Explorer, and they compared their method to the One-Class Support Vector Machine (OCSVM) classifier under Replay and Message Erasure attack scenarios. Their technique achieved an Area Under the Curve (AUC) of over 98% when the attack's frequency and duration increased to 1x and 0.5 seconds.

However, their method was less effective over shorter durations. Nonetheless, they discovered that the OCSVM classifier performed better under these specific conditions (Taylor et al., 2015).

Frequency-based IDS: Focuses on the frequency and patterns of CAN message transmission. Anomalies are detected when there is an unexpected increase or decrease in the frequency of certain CAN messages suggesting potential intrusion activities. The fact that most attacks on networks involve the injection of additional packets, contrasting with legitimate packets that typically follow a precise timing schedule. This discrepancy serves as the foundation for an anomaly detection system that examines both present and past packet timing patterns (Taylor et al., 2015). According to Taylor et al. (2015), calculating the timing between packets within a moving window, then comparing these average timings against historical norms enables anomalies detection.

Machine learning-based IDS: utilizes the principles of artificial intelligence to analyze network traffic, learning from vast amounts of data to identify what constitutes normal and abnormal behavior on the CAN bus. While frequency and statistical-based anomaly detection methods discussed earlier offer the advantages of being lightweight and efficient for real-time applications in CAN bus communications, they often fall short in addressing anomalies within the message data payloads, a critical area of vulnerability. They have largely been concentrated on variations in CAN IDs and the timing intervals between messages, which, although effective for detecting certain types of attacks like DoS and Fuzzy that alter message frequency, overlook more sophisticated threats (Nagarajan et al., 2023). These advanced threats target specific vehicle functions by manipulating the message content itself, leading to potentially severe consequences for

the vehicle's operational integrity and safety of the driver and passengers. Machine learning-based intrusion detection systems fill this gap by their ability to analyze and understand the complex patterns within the payload data, thus offering a more robust solution. Recently, research has increasingly recognized the superiority of machine learning in intrusion detection, integrating it into newer systems to improve their performance and detection capabilities (Nagarajan et al., 2023).

ML-based IDSs can be categorized based on the type of data they analyze: ID-based detection, which focuses on the unique identifiers of CAN messages; payload-based detection, which scrutinizes the content within the messages; and CAN frame-based detection, which considers the entire structure of CAN messages, including both IDs and payloads.

- **CAN ID-based Detection:** This approach zeroes in on the unique identifiers associated with each CAN message. In the context of the CAN network, each message's ID conveys its priority and intent, making ID-based detection a critical tool for spotting irregularities. Machine learning algorithms in this category are trained to recognize patterns or anomalies in the distribution and frequency of message IDs, such as unexpected IDs appearing on the network or significant changes in the usual traffic patterns. This method is particularly effective for identifying spoofing attacks or unauthorized message injections.
- **Payload-based Detection:** Payload-based detection scrutinizes the data content within each CAN message. Since the payload carries the actual information being transmitted, such as sensor readings or control commands, analyzing it can reveal attempts to manipulate vehicle functions or fabricate sensor data. Machine

learning models dedicated to payload analysis learn the normal range and variation of payload values, enabling them to detect payloads that deviate from expected patterns, which could indicate data tampering or injection attacks.

- **CAN Frame-based Detection**: The most comprehensive approach, CAN frame-based detection, examines the entire structure of CAN messages, including both IDs and payloads. This method leverages the full context of each message, allowing for a nuanced understanding of network traffic. Machine learning algorithms applied here are capable of identifying complex attack vectors that might manipulate both the message identifier and the payload in subtle ways. By learning from the entirety of the message structure, these systems can detect sophisticated attacks that would not be apparent if only the ID or payload were analyzed in isolation.

Regardless of the data type they analyze, ML-based intrusion detection systems (IDS) fall into two categories: traditional machine learning algorithms and neural networks.

- **Traditional ML-based IDS:** Traditional algorithms are known for their expediency and straightforward architecture, which also means they have fewer hyperparameters requiring tuning. Avatefipour et al. (2019) introduced an enhanced Anomaly Detection Model (ADM) leveraging a modified version of the One-Class Support Vector Machine (OCSVM). The model demonstrated superior performance when compared to the traditional OCSVM and Isolation Forest (IF) models. However, its effectiveness was solely assessed against Denial of Service (DoS) attacks. Song and Kim (2021) introduced a method for detecting malicious intrusions within vehicle networks utilizing the CAN bus protocol, employing

Random Forest (RF), Support Vector Machine (SVM), Multi-Layer Perceptron (MLP), and Decision Tree (DT) to differentiate between normal and compromised communications. Their approach successfully identified DoS, impersonation, and fuzzy attacks. However, the robustness of their model could be improved by training it with a more extensive dataset (Song & Kim, 2021). Casillo et al. (2019) proposed a Bayesian networks model to detect attacks on CAN-based communication system and achieved 80% precision. Tian et al. (2018) developed a Decision Tree (DT) model and used a Gradient Boosting (GB) technique to enhance performance. The combined model was based on the CAN ID and message frequency, it achieved 97.68% precision.

- **Neural Networks ML-based IDS:** Neural networks are algorithms designed to recognize patterns through a process that mirrors the way the human brain operates. They are composed of layers of interconnected nodes or "neurons," each layer designed to perform specific transformations on its inputs before passing the results to the next layer. Kang & Kang introduced a Deep Neural Networks (DNN)-based CAN bus IDS that resulted in 98% precision (Kang & Kang, 2016). The model employs deep learning methodologies to differentiate between standard network operations and malicious attacks. When benchmarked against conventional neural networks (NNs), the DNN-based intrusion detection system (IDS) exhibits superior accuracy and maintains real-time responsiveness (Kang & Kang, 2016).

Long Short-Term Memory (LSTM) networks, a type of recurrent neural network (RNN), have been increasingly applied in IDS for their ability to capture temporal

dependencies in sequential data. LSTMs are particularly effective in analyzing time-series data, making them suitable for detecting anomalies in network traffic that exhibit temporal patterns. By maintaining long-term dependencies, LSTMs can identify subtle patterns that might indicate malicious activity over time (Sun et al., 2021).

Bidirectional Long Short-Term Memory (BiLSTM) networks extend the capabilities of LSTMs by processing input data in both forward and backward directions (Roy & Cheung, 2018). This bidirectional approach enhances the model's understanding of context, leading to improved detection of complex attack patterns. BiLSTM-based IDS have shown significant improvements in detecting sophisticated intrusions, leveraging the full temporal context of the data to identify anomalies more accurately. Recent studies have demonstrated the effectiveness of LSTM and BiLSTM networks in IDS applications. For example, a BiLSTM-based IDS was developed to analyze CAN bus data for vehicle networks, achieving high accuracy and precision in detecting various types of attacks, including DoS and spoofing attacks (Ding et al., 2022). The ability of BiLSTM networks to consider both past and future contexts of each data point makes them a powerful tool in enhancing the robustness of intrusion detection systems.

In summary, neural network-based IDS, particularly those employing advanced architectures like DNN, LSTM, and BiLSTM, offer robust solutions for detecting intrusions in complex and dynamic environments. Their ability to learn from large datasets and capture intricate patterns in data positions them as a critical component in modern cybersecurity frameworks.

2.7 Summary and Conclusion

This literature review provides a comprehensive overview of intrusion detection systems (IDS) for in-vehicle networks. It highlights the significant role of machine learning in enhancing IDS performance, with analysis categorized based on CAN IDs, payloads, and frames, and contrasts traditional machine learning approaches with neural network methods in IDS design. The review underscores the potential of neural networks to achieve higher precision and real-time responsiveness in intrusion detection.

Existing research in the domain of CAN bus intrusion detection has predominantly focused on detection accuracy, often at the expense of crucial aspects such as detection time and model size, which are essential for real-time deployment. There is a notable lack of attention to the stringent real-time requirements needed for implementing an IDS within the resource-constrained environment of automotive electronics. Many studies have not adequately addressed the balance between maintaining high detection performance and ensuring that the IDS is lightweight and efficient enough for practical use in vehicles. Additionally, while several studies have explored various machine learning models for CAN bus intrusion detection, there is a gap in comprehensive real-time performance analysis. Most research tends to focus on the static performance of these models without integrating evaluations of their real-time deployment capabilities. This gap includes a lack of thorough examination of model size and detection time capabilities within the context of CAN bus environments, which are critical factors for the feasibility of IDS deployment in real-world automotive scenarios.

In conclusion, the literature review highlights the advancements and potential of machine learning and neural network approaches in improving IDS for in-vehicle

networks. However, it also identifies gaps, particularly in the areas of real-time performance, model efficiency, and practical deployment within the constrained environments of automotive systems.

Chapter 3—Methodology

3.1 Introduction

This praxis introduces an innovative method for constructing a real-time Controller Area Network (CAN) bus Intrusion Detection System (IDS) utilizing machine learning. While numerous studies have aimed to create highly precise machine learning models, the aspect of detection speed has been largely overlooked, posing challenges for practical application. This research explores different configurations across three different models to identify the most efficient setup. By testing the proposed machine learning models in a controlled, simulated environment, we anticipate demonstrating superior performance with reduced detection times in comparison to existing models. This praxis tests the performance of three ML models: Random Forest (RF), eXtreme Gradient Boosting (XGBoost), and Bidirectional Long Short-Term Memory (Bi-LSTM) which is an advanced form of Artificial Neural Networks (ANN).

This chapter presents a comprehensive overview of the research methods used in this praxis to develop each of the three machine learning models mentioned above to compare their performance and select the best candidate for real-time CAN bus IDS. Section 3.2 of this chapter presents the dataset used in this praxis. Section 3.3 provides Exploratory Data Analysis (EDA) performed before the data preprocessing activities covered in Section 3.4. A detailed review of each of the three models is provided in Section 3.5, while Section 3.6 goes in-depth to show the steps taken to build and configure each model.

3.2 Dataset

The dataset used in this praxis for evaluating our models is the Car-Hacking Dataset for Intrusion Detection, developed by the Car Hacking and Countermeasure Research Lab (HCRL). The compiled dataset is a product of CAN traffic logging from an operational vehicle, utilizing the OBD-II port for data collection. The dataset captures synthetic attack scenarios superimposed onto the vehicle's normal operation. These scenarios encompass DoS (Denial of Service) attacks, which aim to overwhelm the vehicle's network by flooding it with excessive traffic; fuzzy attacks that introduce abnormal, randomized messages into the CAN network; gear spoofing attacks that fabricate messages related to the drive gear's position; and RPM gauge spoofing, where the revolutions per minute indicators are manipulated through false data injection.

The data was gathered with precision to ensure a realistic representation of both regular vehicular communications and potential security breaches. Given its size, this comprehensive dataset reflects the diversity of attack vectors and provides an extensive ground truth for developing and benchmarking intrusion detection systems. With each record in the dataset is labeled as either normal or attack-specific, the dataset serves as an ideal training ground for machine learning models designed to identify and differentiate between legitimate operations and malicious activities within automotive systems.

3.2.1 Dataset Acquisition

The Car Hacking and Countermeasure Lab developed the Car-Hacking Dataset, available in the lab’s repository (Kim, 2021). The files are publicly available via the Dropbox hosting services. Due to the considerable size of the CSV files, conventional data processing tools such as Excel proved inadequate for efficient data exploration and

manipulation. The CSV files were imported into Python using the Pandas library to address this challenge. This approach was chosen because Pandas is designed to handle large datasets efficiently. Once loaded into Python, each file was read into a distinct DataFrame object using the `read_csv` function. Subsequently, all DataFrames were merged into a single DataFrame, thus consolidating all records into one accessible data structure. This procedure enabled a more effective data management, allowing for thorough data analysis and manipulation suitable for the dataset's scale."

3.2.2 Dataset Structure

The dataset consists of five files: one for normal run data and four attack-specific files, one for each attack type: Fuzzy, spoofing the drive gear, spoofing the RPM gauze, and DoS attack. Table 3-1 below lists the number and type of messages on each file contained in the dataset:

Table 3-1. Dataset Structure

Attack Type	# of Messages	# of Normal Messages	# of Injected Messages
DoS Attack	3,665,771	3,078,250	587,521
Fuzzy Attack	3,838,860	3,347,013	491,847
Spoofing the drive gear	4,443,142	3,845,890	597,252
Spoofing the RPM gauze	4,621,702	3,966,805	654,897
GIDS: Attack-free (normal)	988,987	988,872	

- DoS Attack: In this file, a significant number of messages are injected to simulate a Denial of Service attack, which is characterized by an overwhelming flood of traffic intended to disrupt the normal functions of the network.
- Fuzzy Attack: This type of attack involves sending random messages through the CAN network to create confusion and potential malfunctions. The dataset contains nearly half a million such injected messages.
- Spoofing the Drive Gear: This file captures a spoofing attack where the adversary sends fraudulent messages to manipulate the drive gear's reported state. Close to 600,000 messages are designed to mimic this attack scenario.
- Spoofing the RPM Gauge: Similar to the drive gear spoofing, this attack involves sending deceptive messages to alter the readings of the RPM gauge. This file includes over 650,000 messages crafted to represent such conditions.
- Attack-Free (Normal) Data: This file contains nearly one million messages, almost all of which are normal messages, representing the standard operation of the vehicle's CAN network without any interference. This data is crucial for establishing a baseline of regular network activity against which anomalous behavior can be detected.

  Each message within the dataset has the following thirteen attributes:

  - Attribute 1, **Timestamp**. This is the recorded time of each CAN message, denoted in seconds. It allows for a temporal sequence analysis of the network traffic, providing insights into the behavior and timing of normal operations and potential cyber-attacks.

- Attribute 2, **CAN ID**: Representing the identifier of each CAN message in hexadecimal format (e.g., 043f), the CAN ID is crucial for distinguishing between different types of messages transmitted over the CAN network, indicating their priority and purpose.
- Attribute 3, **DLC** (Data Length Code): This indicates the number of data bytes in the CAN message, which can range from 0 to 8 bytes. The DLC helps to understand the payload size and structure of the messages.
- Attribute 4 to Attribute 11, **DATA0, DATA1, DATA2, DATA3, DATA4, DATA5, DATA6, and DATA7**: These eight fields represent the actual data values transmitted within the CAN message. Each data byte is recorded, providing the message payload content used for various vehicle controls and monitoring functions.
- Attribute 12, **Flag**: Each message in the dataset is marked with a flag, either '**T**' or '**R**'. The '**T**' flag indicates the message has been injected, simulating an attack or malicious activity within the CAN network. In contrast, the '**R**' flag denotes a normal message and represents regular vehicle operations.
- Attribute 13, **AttackType**: Specifies the type of attack simulated in the CAN network when a message is marked with “T” flag. It categorizes the nature of the injected attack.

3.3 Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a pivotal step in the data science workflow, providing deep insights into the dataset. It involves a series of methods to visualize and understand the dataset's underlying structure and characteristics, a process that is

instrumental for the subsequent development of models. EDA facilitates a thorough examination of the features, understanding their distribution, scale, and the relationships between them. This is particularly vital as our dataset encompasses multiple attack classes, and EDA aids in identifying any imbalances between these classes that could potentially introduce bias into ML models.

For our dataset, the merged files mentioned in section 3.2.1, were analyzed by calculating basic statistical metrics for each feature, such as the count, mean, standard deviation, minimum, and maximum values, along with the 25th, 50th (median), and 75th percentiles. These statistics offer valuable information regarding the scale and variance of the features, highlighting any data points that may be considered outliers. Furthermore, the dataset was inspected for missing or “Not a Number” (NAN) values. These gaps in the dataset are crucial to identify as they must be addressed, either by imputation or exclusion, to ensure the integrity of the ML models' training, validation, and testing phases. Another layer of our EDA involves plotting the distribution of features like CAN ID frequencies, message types, and payload patterns. This visual exploration assists in discerning the relative scale of features and identifying any that are peculiar to specific records. Another component of our EDA is examining the data types for each feature, which informs the preprocessing steps required to optimize the data for machine learning algorithms.

3.4 Data Preprocessing

Preprocessing is a key step in creating machine learning models. It involves transforming raw data into a format that is more suitable for building predictive models.

The Car-Hacking Dataset undergoes several preprocessing steps to ensure the quality and usefulness of the data for our models:

3.4.1 Data Cleaning and Filtering

Data cleaning involves dealing with missing or incorrect data within the dataset. For the Car-Hacking Dataset, this meant scanning for NAN (Not a Number) values, which indicate missing data, and deciding whether to impute or remove these values based on their incidence and significance. The data cleaning process was expanded upon identifying several rows with corrupted data. This discovery necessitated a careful review, after which it was determined that these rows could not be repaired and thus were removed (please discuss which files had corrupted data and number of records which were removed). This step was critical to maintaining the dataset's quality, as retaining such flawed information could have led to significant inaccuracies in the machine learning model's training and subsequent predictions. The removal of corrupted data helps ensure that the analysis is based on accurate and reliable information.

3.4.2 Categorical Data Encoding

When performing EDA on the Car-Hacking dataset, we discovered categorical features. Because machine learning models cannot work directly with categorical data, categorical features were transformed into a numerical format. One-hot Encoding technique was applied to categorical variables like CAN IDs and Flag types, converting them to binary matrix essential for model processing.

3.4.3 Data Standardization

As part of our data preparation, we implemented a normalization technique to ensure that all the features in our Car-Hacking dataset were on a comparable scale. This

was essential since differences in scale could lead to biased computations in our analysis. To achieve this, we first calculated the average value and the spread (standard deviation) of each feature within our training data. We then adjusted these features so that, on average, they centered around zero and had a uniform spread. After establishing these scaling factors from our training set, we applied the same adjustments to our test data. It was essential to maintain consistency in how both sets of data were treated to ensure that our analysis would be accurate and reliable.

With these adjustments, our dataset was transformed into an optimized format for building and testing our machine-learning models. This standardization process ensured uniformity across the dataset's different features, permitting each to contribute equitably to the analysis without any single feature dominating due to its scale.

3.4.4 Data Splitting into Training, Validation, and Testing Datasets

In this praxis, we partitioned the Car-Hacking dataset into three distinct subsets to evaluate and refine our models' performance. Initially, the dataset underwent a random split, allocating 70% of the data to the training set, which is used to educate the model on the underlying patterns and behaviors associated with normal operations and various cyberattacks. We ensured that this training set was sufficiently large to provide a vigorous learning experience for the model, encompassing various scenarios.

Subsequently, the remaining data was divided into two smaller equal sets: 15% for validation and 15% for testing. The validation set plays a critical role in the model development process; it acts as a checkpoint to evaluate the model's predictions and to fine-tune its parameters. This step is pivotal as it helps to prevent overfitting, ensuring that the model does not merely memorize the training data but also learns to generalize

effectively to new, unseen data. The final 15% of the data was reserved for the testing set. This subset is crucial as it provides an unbiased evaluation of the model's performance, simulating how it would perform in real-world situations by processing data it has never encountered during the training or validation phases.

By adopting this methodical approach to data partitioning, we emulated a realistic environment where the model's ability to predict and withstand cyberattacks can be assessed with high confidence. This rigorous splitting strategy is fundamental to developing a resilient and reliable intrusion detection system for vehicular networks.

3.4.5 Data Balancing

In developing machine learning models for the Car-Hacking dataset, we confronted the challenge of imbalanced data, a common issue in multi-class datasets that can result in a model bias towards more frequently occurring classes. Such a bias occurs because a model may learn to preferentially predict the majority class due to its prevalence in the training data, consequently neglecting the minority classes. This could severely undermine the model's ability to detect less common but potentially more dangerous cyberattacks represented by the minority classes.

To rectify this imbalance and enhance the model's predictive performance across all classes, we employed the Synthetic Minority Oversampling Technique (SMOTE). This technique is favored over simple minority oversampling, as it generates synthetic examples rather than duplicating existing ones, thereby expanding the decision space within which the model learns. Applying SMOTE to our dataset involved creating artificial data points for minority classes. We achieved this by randomly selecting a point from a minority class and identifying its k-nearest neighbors within that class. For each

chosen minority point, we then selected one of its neighbors and created a new, synthetic data point along the line segment connecting these two points in feature space. This process not only increased the representation of minority classes but also introduced more variety into the training data, which is critical for building a robust model that can generalize well to new data.

It is crucial to note that we applied SMOTE solely to the training set, thereby preserving the original structure and distribution of the validation and testing sets. This approach ensures that while the model benefits from a balanced training environment, its performance is evaluated on data sets that accurately reflect the real-world distribution and diversity of cyberattack scenarios. By carefully balancing our dataset, we created an unbiased training context for our models, improving their accuracy and reliability in detecting attacks.

3.5 Machine Learning Models

This praxis studies three different ML models: Random Forest (RF), extreme Gradient Boosting (XGBoost), and Bidirectional Long Short-Term Memory (Bi-LSTM), which is an advanced form of Artificial Neural Networks (ANN). This section will introduce each model.

3.5.1 Random Forest

The first machine learning (ML) model explored in this praxis is the Random Forest (RF). Developed by Breiman in the early 2002s, is a powerful ensemble ML approach known for effectively handling large datasets with high dimensionality (Breiman, 2001). Given the dataset in use, this quality renders RF particularly suitable for the objectives of this praxis. As an ensemble of Decision Trees (DTs), RF capitalizes on

the strengths of multiple weak learners to create a robust predictive model (Biau & Scornet, 2016).

RF models aim to diminish the variance of each prediction, thus improving the model's generalizability to new data. This is achieved by synthesizing the individual predictions of the constituent DTs through a majority voting mechanism (Biau & Scornet, 2016). The RF employs a technique known as Bootstrap Aggregating or Bagging, which involves creating multiple bootstrap samples from the training data (Breiman, 1996). Each sample is used to train a separate DT, contributing to the diversity of the ensemble and thereby reducing the variance of the predictions (Biau & Scornet, 2016). The construction of a DT within the RF uses a randomized feature selection process to decide on splits, which contributes to the de-correlation between the trees, enhancing the ensemble's overall performance (Breiman, 1996).

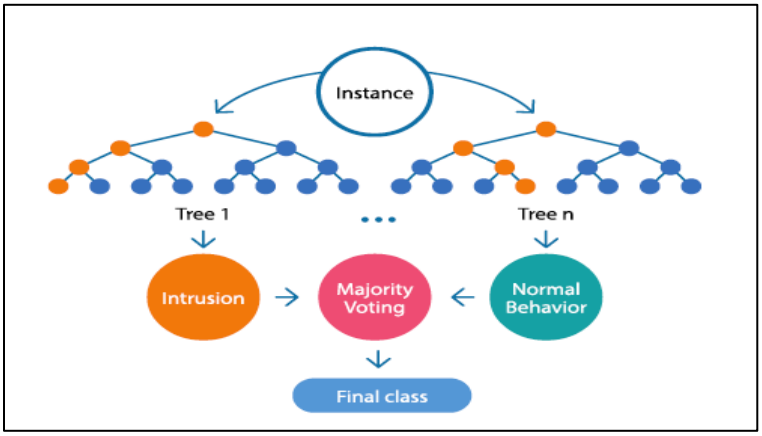


Figure 3-1. Random Forest Structure

Figure 3-1 illustrates a sample RF structure, with orange nodes depicting the decision path taken from the root to a leaf, culminating in a classification outcome. Each DT in the ensemble contributes its vote, based on such paths, to the final classification provided by the RF model.

The input dataset for the RF is randomly sampled for each DT to ensure varied learning and to mitigate overfitting (Breiman, 1996). The splitting criterion for the features at every node of a DT classifier involves the Gini impurity index and entropy, both of which aim to quantify the purity of a node (Tangirala, 2020). The Gini impurity is a measure of the likelihood of incorrect classification if a label were assigned randomly to a datapoint, while entropy measures the impurity of a split, with lower entropy indicating a more definitive classification at that node (Tangirala, 2020). A perfectly pure node in a DT, which does not require further splitting, would have a Gini impurity index of zero, indicating that all data points at the node belong to a single class. The entropy criterion similarly values nodes that cleanly divide the data into single classes. These criteria are calculated as follows (Tangirala, 2020):

$$Gini\ Impurity\ Index = 1 - \sum_{i=1}^{j} p_i^2$$

where: p_i = probability of class i and

j = total number of classes in the dataset.

$$Entropy = -\sum_{i=1}^{j} p_i^2 \log(p_i)$$

where: p_i = probability of class i and

j = total number of classes in the dataset.

In addition to the splitting criteria, RF also includes several hyperparameters that influence the model's performance, such as the number of trees in the forest, the maximum depth of the trees, and the minimum number of samples required to split a

node. The configuration of these hyperparameters will be discussed in detail in section 3.6 of this praxis when addressing model development.

3.5.2 eXtreme Gradient Boosting

eXtreme Gradient Boosting (XGBoost) is an algorithm that has gained significant popularity due to its effectiveness in predictive modeling. It operates based on the principle of gradient boosting, which involves the sequential creation of decision trees, where each tree attempts to correct the errors of its predecessor (Chen et al., 2019). The methodical approach of XGBoost in iterating through the construction of decision trees allows for efficient optimization at each step of the algorithm. Specifically, at the *i-th* step, the algorithm optimizes the current tree $f_m(x_i)$, without re-evaluating the entire model. The update equation can be represented as:

$$F_m(x_i) = F_{m-1}(x_i) + f_m(x_i)$$

Here, $F_m(x_i)$ represents the boosted model at the m-*th* iteration, $F_{m-1}(x_i)$ denotes the model obtained up until the previous iteration and $f_m(x_i)$ is the current decision tree being added to the model. This iterative approach ensures that the model improves in accuracy incrementally, with each new tree focusing on the residuals of the ensemble's previous predictions, as depicted in Figure 3-2 (Rani et al., 2022).

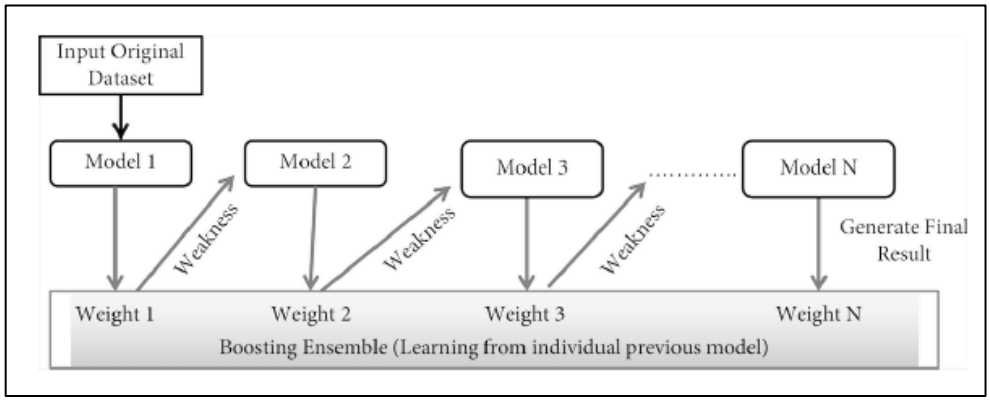


Figure 3-2. Boosting Ensemble-based Learning Model

XGBoost objective function not only aims to reduce the loss, which measures prediction accuracy but also incorporates a regularization term to prevent overfitting. Overfitting occurs when a model is too complex and starts to memorize the training data instead of learning from it, leading to poor performance on new, unseen data. The regularization term in XGBoost penalizes model complexity, keeping the model simple enough to maintain generalizability to unseen data. This is crucial because it balances the model's fit to the training data with the need to maintain simplicity for prediction accuracy on new data (Chen et al., 2019). The synergy between minimizing the predictive loss and controlling model complexity allows XGBoost to deliver superior performance based on hyperparameters configuration, which will be addressed in detail in section 3.6: Learning Rate, Gamma, Regression Alpha, Maximum Tree Depth, Number of Estimators, and Booster Type.

3.5.3 BiLSTM

Bidirectional Long Short-Term Memory (BiLSTM) network is considered a type of Artificial Neural Network (ANN). ANNs are computing systems vaguely inspired by the biological neural networks that constitute animal brains. They consist of interconnected groups of artificial neurons, where each connection can transmit a signal from one neuron to another. The receiving neuron processes the signal and then signals downstream neurons connected to it.

BiLSTM is a specific architecture of Recurrent Neural Networks (RNNs), a class of ANNs designed to recognize patterns in data sequences such as text, genomes, handwriting, or numerical time series data. Traditional RNNs have a shortcoming in capturing long-term dependencies due to issues like vanishing and exploding gradients.

LSTMs were introduced to overcome these challenges by incorporating mechanisms called gates that regulate the flow of information (Gao, 2022). As shown in Figure 3-3, a BiLSTM enhances the LSTM model by processing the data in both forward and backward directions (Roy & Cheung, 2018).This bidirectional processing allows the network to have both past and future context at any point in the sequence, making it more effective for tasks where understanding the entire context is crucial (Kang et al., 2021).

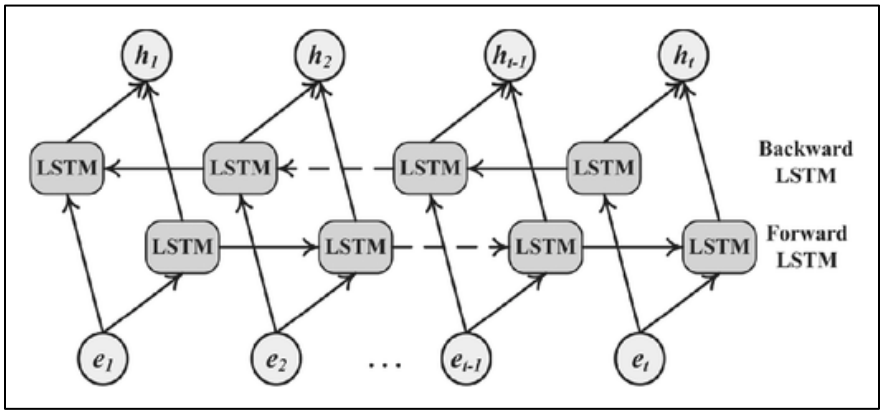


Figure 3-3. Neural Network Architecture of BiLSTM

BiLSTM is therefore a sophisticated type of Artificial Neural Network, leveraging the temporal dynamics of RNNs while addressing their limitations in handling long-range dependencies through the LSTM architecture and improving context understanding with bidirectional processing (Roy & Cheung, 2018).

3.6 ML Model Development

This section describes the procedure for building three machine learning models tailored to our research. The process begins with carefully adjusting hyperparameters, optimizing each model's performance. Following the tuning phase, we undertake a thorough batch validation, assessing the models' predictive precision using a designated portion of our dataset.

3.6.1 Model Hyperparameter Tuning

A critical phase in developing our three machine learning models for the Car-Hacking dataset was tuning the algorithms' hyperparameters. Hyperparameters are adjustable settings that govern the learning process of ML models. Tuning these parameters is pivotal to calibrating the models for optimal performance. We utilized the GridSearchCV function from Python's Scikit-learn library to explore various hyperparameter combinations systematically. GridSearchCV conducts an exhaustive search over a specified parameter grid using cross-validation, evaluating each combination to determine the most effective set of hyperparameters. Table 3-2 lists the different hyperparameters we tuned for each model.

Table 3-2. Model Hyperparameters

Model	Hyperparameter	Value Range
Random Forest	n_estimators	10, 100, 200
	max_depth	10, 20, None
	min_samples_split	2, 10
	min_sample_leaf	1, 4
	criterion	gini, entropy
XGBoost	n_estimators	50, 100, 150, 200, 250
	max_depth	3, 5, 7
	learning_rate	0.01, 0.1, 0.2
	Min_child_weight	1, 3, 5
	reg_alpha	0, 0.1, 0.5, 1
	reg_lambda	0.5, 1, 1.5, 2
BiLSTM	num_lstm_units	50, 100, 150, 200, 250
	batch_size	32, 64, 128, 256
	dropout_rate	0.1, 0.2, 0.3, 0.5
	learning_rate	0.001, 0.01, 0.1
	optimizer	Adam, rmsprop, sgd

For the Random Forest model, we tested a range of values for hyperparameters, such as the number of trees in the forest and the maximum depth of the trees. In the case

of the XGBoost model, critical hyperparameters like the learning rate, tree complexity, and regularization penalties were examined. For the BiLSTM model, we assessed configurations, including the number of LSTM units and the dropout rate. Table 3-1 lists the hyperparameter values investigated for each model.

The hyperparameters' ranges selected for the GridSearchCV process were determined after an initial exploration to assess model sensitivity. The goal was to identify the values to which the model's performance was most responsive. This methodical approach allowed us to narrow down the hyperparameter space to those ranges where the model demonstrated significant performance variations in response to hyperparameter adjustments.

3.6.2 Model Performance Metrics

To evaluate and select the optimal model from the three machine learning models we developed in the previous section, a list of performance metrics is collected for each model. These metrics are accuracy, false alarm rate (FAR), precision, recall, F1 score, the Area Under the Curve (AUC) of the Receiver Operating Characteristic (ROC), and classification time.

- **Accuracy**: This metric reflects the model's overall correctness by comparing the count of true positive and true negative predictions to all predictions. A high accuracy indicates that the model can reliably identify both positive and negative classes.

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

  Where: TP is the number of true positives,

  TN is the number of true negatives,

FP is the number of false positives, and

FN is the number of false negatives.

- **Precision**: Precision scores the model's exactness by calculating the fraction of true positives among all instances labeled as positives. High precision suggests that it will likely be correct when the model predicts an instance as positive.

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

Where: TP is the number of true positives, and

FP is the number of false positives.

- **Recall**: Also known as sensitivity, recall measures the model's ability to detect all actual positives.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Where: TP is the number of true positives, and

FN is the number false negatives.

- **F1 Score**: The F1 score harmonizes precision and recall into a single metric, providing a comprehensive view of the model's performance when a balance between precision and recall is desired.

$$\text{F1 Score} = 2 \text{ X } \frac{\text{Precison X Recall}}{\text{Precison} + \text{Recall}}$$

- **AUC-ROC**: The Area Under the Receiver Operating Characteristic curve gauges the model's proficiency in distinguishing between the classes across all thresholds. It reflects the likelihood that the model will rank a randomly chosen positive instance higher than a negative one. [validate equation]

$$\mathrm{AUC} - \mathrm{ROC} = \int_0^1 \mathrm{TPR}\,(\mathrm{FPR})\mathrm{d}(\mathrm{FPR})$$

Where: TPR is the True Positive Rate also known as Recall.

- **Detection Time**: This performance metric measures the average time required for the model to classify a single instance in a real-time context. It is critical for our praxis, given the nature of real-time classification requirements.
- **Detection Time per Instance**: Measures the time taken for the model to process and classify a single instance. This metric is vital for applications requiring real-time or near-real-time responses, as it directly impacts the system's overall latency and efficiency. Optimizing this time ensures the model can meet stringent performance requirements in practical deployment scenarios.
- **Model Size**: Refers to the memory footprint of the trained model, typically measured in kilobytes or megabytes. This metric is crucial as it impacts the storage requirements, loading time, and the feasibility of deploying the model on resource-constrained devices. Reducing model size without significantly compromising performance is often a key objective in model optimization.

3.6.3 Model Performance Validation and Testing

Following the establishment of evaluation metrics, the next phase involved validating the performance of the Random Forest, XGBoost, and BiLSTM models against our Car-Hacking dataset. Performance validation is the process of verifying the effectiveness of a model using a dataset independent from the data it was trained on. The performance of each model was systematically scrutinized by applying the validation set and observing the outcomes against the established metrics.

After the validation phase, the models were tested using an independent test set. This step was crucial in assessing how well the models could handle new, unseen data and identifying any potential overfitting issues. The test set results provided a final measure of each model's performance, allowing for a comprehensive evaluation beyond the initial validation metrics.

The accuracy metric was analyzed to determine the models' overall rate of correct predictions, including both the true positive and true negative instances. A high accuracy indicates that the model can reliably identify both positive and negative classes. The precision metric provided insight into the reliability of positive classifications by calculating the fraction of true positives among all instances labeled as positives. High precision suggests that it will likely be correct when the model predicts an instance as positive. The recall metric, also known as sensitivity, was employed to confirm the models' ability to detect all actual positives and its capacity to identify all potential security breaches. The F1 score was calculated for each model to assess the balance between precision and recall, providing a comprehensive view of the model's performance when a balance between precision and recall is desired, which is particularly important in scenarios where false positives and false negatives have significant consequences, as in our case. Additionally, we assessed the models' ability to distinguish between benign and malicious instances under varying thresholds using the AUC-ROC metric, which gauges the model's proficiency in distinguishing between the classes across all thresholds. It reflects the likelihood that the model will rank a randomly chosen positive instance higher than a negative one.

The classification time was measured to ascertain each model's efficiency in real-time threat detection. The detection time per instance metric, which measures the time taken for the model to process and classify a single instance, was also considered. This is vital for applications requiring real-time responses, as it directly impacts the system's overall latency and efficiency. Optimizing this time ensures the model can meet stringent performance requirements in practical deployment scenarios in vehicular networks where the timeliness of threat detection must be less than 2 milliseconds (Avatefipour et al., 2019). Finally, the model size was evaluated, referring to the memory footprint of the trained model, typically measured in kilobytes or megabytes. This metric is crucial as it impacts the storage requirements, loading time, and the feasibility of deploying the model on resource-constrained devices. Reducing model size without significantly compromising performance is often a key objective in model optimization.

The results from this validation process provided an understanding of each model's performance strengths and weaknesses, allowing for informed decisions on selecting the best possible model considering all collected metrics.

Chapter 4—Results

4.1 Introduction

The primary objective of this praxis is to develop an efficient real-time Intrusion Detection System (IDS) tailored specifically for the Controller Area Network (CAN) bus environment. Leveraging advanced machine learning (ML) techniques, this system aims to accurately classify and detect cyber intrusions within the CAN bus network in real-time. The investigation evaluates the performance of three distinct ML models: Random Forest (RF), eXtreme Gradient Boosting (XGBoost), and Bidirectional Long Short-Term Memory (Bi-LSTM), which represents a sophisticated variant of Artificial Neural Networks (ANN).

To identify the most appropriate ML model for this critical task, a comprehensive analysis of various performance metrics is conducted, allowing for thorough comparison across the models. This chapter is structured to closely align with the sections outlined in Chapter 3 of the methodology, ensuring a coherent presentation of results derived from each procedural step.

Following this introductory section, Section 4.2 discusses the outcomes from the Exploratory Data Analysis (EDA) phase. Section 4.3 covers the details of diverse configurations and data preprocessing procedures undertaken to fine-tune the performance of the ML models under examination. Section 4.4 examines the results obtained from the testing of the proposed IDS system. This comprehensive evaluation aims to showcase the efficacy of the selected ML models for real-time CAN bus network intrusion detection.

4.2 Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a crucial step in understanding the characteristics and structure of a dataset. In this analysis, we first computed general statistics for each feature in the dataset then provided a detailed explanation for each feature's statistics.

4.2.1 Feature Overview

The dataset contains the following thirteen attributes:

1. `Timestamp`
2. `canID`
3. `DLC`
4. `Data0`
5. `Data1`
6. `Data2`
7. `Data3`
8. `Data4`
9. `Data5`
10. `Data6`
11. `Data7`
12. 'Flag'
13. 'AttackType'

For each of these features, the following statistics were calculated:

1. **Count**: The number of non-null entries in the column.

2. **Unique Values**: The number of unique values present in the column.

3. **Missing Values**: The number of null or missing entries in the column.

4. **Most Frequent Value**: The value that appears most frequently in the column.

5. **Most Frequent Value Count**: The number of times the most frequent value appears in the column.

4.2.2 Feature Detailed Statistics

1. Feature 1: ‘Timestamp’

- Count: 16,569,475
- Unique Values: 13,896,271
- Missing Values: 0
- Most Frequent Value: 1478200727.41947
- Most Frequent Value Count: 4

The `Timestamp` feature has a high number of unique values, indicating a wide range of timestamps in the data, which is expected in time-series data. The are only 4 instances of frequent value required to investigate and address. Technically, there shouldn’t be any duplicate values of the Timestamp feature. On the other hand, there are no missing values.

2. Feature 2: ‘canID’

- Count: 16,569,475
- Unique Values: 2,048
- Missing Values: 0
- Most Frequent Value: 0316
- Most Frequent Value Count: 1,431,308

The `canID` feature represents identifiers for CAN messages. It has 2,048 unique values, suggesting a variety of CAN IDs. The most frequent CAN ID appears over 1.4 million times. This is expected since all messages are generated from one of the unique electronic component unit (ECU) in the car, the same component generates several messages during operation. The ‘canID’ values are in hexadecimal (hex) format. Each ‘canID’ is a 4-digit hex number, which allows for a broad range of possible unique identifiers up to 65,536. The ‘canID’ does not have any missing values.

3. Feature 3: ‘DLC’

- Count: 16,569,475
- Unique Values: 4
- Missing Values: 0
- Most Frequent Value: 8
- Most Frequent Value Count: 16,368,810

The DLC (Data Length Code) feature in our dataset specifies the length of the data field in CAN (Controller Area Network) messages. This feature is crucial for understanding the structure and size of each CAN message. This feature has only 4 unique values, with a significant majority of entries having a value of 8, indicating that most CAN messages are of the same length. The DLC feature is directly linked to the Data0 to Data7 fields in each CAN message. The value of DLC indicates how many of the data fields (Data0 to Data7) are populated with values in a particular message.

Understanding this relationship is essential for correctly interpreting the data within each message. The presence and distribution of DLC values, along with their corresponding Data fields, provide valuable insights into the message structure and

potential anomalies. For instance, messages with a DLC value of 8 have all data fields populated, while those with lower DLC values have fewer populated fields, with the remaining fields typically set to default or null values. This linkage is vital for analyzing and validating CAN message integrity in the dataset. In our dataset the 'DLC' does not have any missing value.

4. Feature 4 to Feature 11: `Data0`, `Data1`, `Data2`, `Data3`, `Data4`, `Data5`, `Data6`, and `Data7`

- Count: 16,569,475 (slightly less for some Data columns due to missing values)
- Unique Values: Around 256 for each Data column
- Missing Values: Some columns have up to 200,665 missing values
- Most Frequent Value: 00 for most Data columns
- Most Frequent Value Count: Ranges from around 5 million to 9 million

The `Data0` to `Data7` features represent the data bytes in the CAN messages. The Data0 to Data7 features are essential for understanding the data payload of CAN messages. Each of these columns has 256 unique values, suggesting a full byte range (0-255). The most frequent value for each of these columns is 00. As observed, some columns have up to 200,665 missing values. This is an important observation that links back to the DLC feature discussed earlier. The missing values in the Data0 to Data7 columns are not random but are directly related to the DLC values, it specifies the number of data bytes present in a CAN message. They indicate that the CAN message had fewer data bytes as specified by the DLC. This relationship is expected and indicates that missing values are not an issue but a reflection of the varying lengths of CAN

message data fields. Understanding this linkage helps in correctly interpreting the structure and content of CAN messages in the dataset.

5. Feature 12: `Flag`

- Count: 16,368,810
- Unique Values: 2
- Missing Values: 200,665
- Most Frequent Value: R
- Most Frequent Value Count: 14,037,293

The Flag feature in our dataset is crucial for distinguishing between normal traffic (R) and attack messages (T) in the CAN (Controller Area Network) data. With 16,368,810 non-null entries, it has only 2 unique values: R, which indicates normal traffic and is the most frequent with 14,037,293 occurrences, and T, which indicates an attack. There are 200,665 missing values in the Flag feature that need to be addressed during data preprocessing, as each row must have a flag for accurate classification. The prevalence of R is expected, reflecting the typical scenario in CAN datasets where normal traffic predominates. This imbalance, however, must be addressed to prevent our models from becoming biased towards the majority class. Addressing the missing values and the data imbalance in the Flag feature is critical for maintaining dataset integrity and ensuring robust, unbiased model training.

6. Feature 13: `AttackType`

- Count: 16,569,475
- Unique Values: 4
- Missing Values: 0

- Most Frequent Value: normal
- Most Frequent Value Count: 14,237,958

The `AttackType` feature in our dataset categorizes each CAN message as either normal or belonging to one of several attack types: `SPOOFING`, `DOS`, and `FUZZING`. It contains 16,569,475 non-null entries with 4 unique values, ensuring complete classification of each message. The most frequent value is `normal`, appearing 14,237,958 times, followed by `SPOOFING` with 1,252,149 occurrences, `DOS` with 587,521 occurrences, and `FUZZING` with 491,847 occurrences. This distribution indicates a significant imbalance, with the majority of messages labeled as normal traffic, which is expected in such datasets. To prevent model bias, this imbalance must be addressed during data preprocessing.

4.2.3 Data Cleaning

The data cleaning process, part of the Exploratory Data Analysis, involved several steps to ensure the integrity and consistency of the dataset, which contained hexadecimal values in multiple columns. Initially, non-hexadecimal values were identified in the columns labeled `Data0` to `Data7`. This was done using a lambda function to check if each character in the values belonged to the hexadecimal character set. The unique non-hex values were printed and counted for inspection as illustrated in Figure 4-1. The cleansing process addressed non-hexadecimal values, including 'R' and `NaN`, by replacing 'R' with '0' and filling `NaN` values with '0'. This process was applied to all specified columns, and the number

of cleansed columns was recorded. Finally, all hexadecimal values were converted to decimal format, ensuring that all values were treated as strings before conversion.

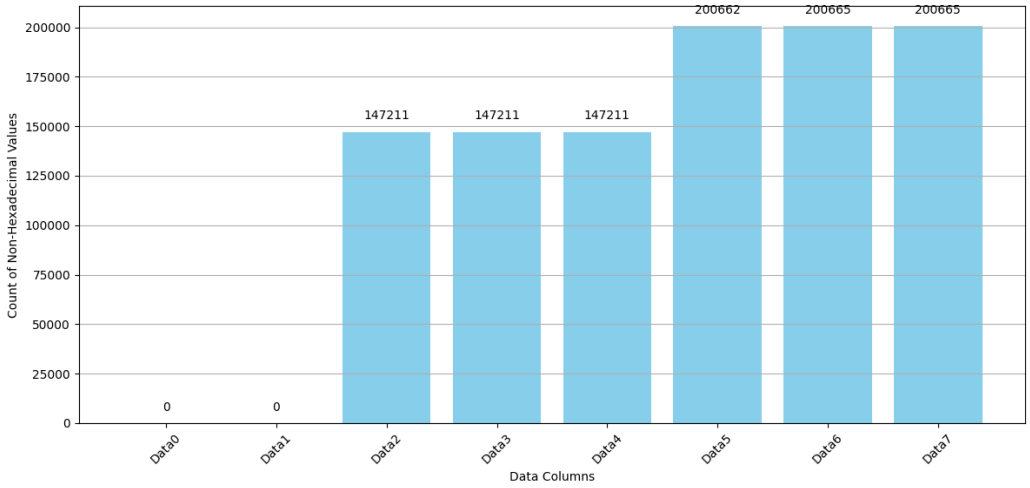


Figure 4-1. Non-Hexadecimal Values in Data Columns

In addition to handling non-hexadecimal values, the dataset was also checked for duplicate records. A query was performed against the dataset to identify any duplicate entries. Any records found to be duplicates were deleted to ensure that each data entry was unique and to prevent any potential bias or redundancy in the subsequent analysis.

The comprehensive cleaning process resulted in a dataset that was transformed from hexadecimal to a consistent decimal format across specified columns. All non-hexadecimal values were successfully replaced, and all duplicate records were removed. The data was prepared for subsequent data preprocessing, ensuring a high level of data quality and reliability.

4.3 Data Preprocessing

In the data preprocessing phase, several key activities were undertaken to prepare the CAN bus dataset for analysis. These included encoding categorical data to numerical

values for compatibility with machine learning models and standardizing numerical features to ensure uniformity and improve model performance. Additionally, efforts were made to balance the dataset to address class imbalances, and feature selection was performed to retain the most relevant data attributes. These steps are crucial for enhancing the quality and effectiveness of the subsequent modeling process and are detailed in the following sections.

4.3.1 Categorical Data Encoding

This activity involves converting categorical labels into a numerical format that can be used by machine learning algorithms. This is crucial because most machine learning models require numerical input and cannot process categorical data directly. In our case, the target variable AttackType is encoded using LabelEncoder. This transforms the categorical labels (normal, DOS, FUZZING, SPOOFING) into numerical labels that can be used by the machine learning models. The encoding ensures that each unique AttackType is assigned a unique integer value.

4.3.2 Data standardization

Data standardization is the process of scaling the features so that they have a mean of zero and a standard deviation of one. This ensures that all features contribute equally to the model, preventing features with larger scales from dominating the model training process. In this case, the StandardScaler is used to standardize the features. The scaler is fitted to the training data and then applied to transform the training, validation, and test sets. This ensures that the data used in training and evaluating the model is standardized, which typically leads to better model performance and stability.

4.3.3 Data balancing

Data balancing involves addressing class imbalances in the dataset to ensure that the model is not biased towards the majority class. SMOTE (Synthetic Minority Over-sampling Technique) is used to create synthetic samples for the minority classes to balance the class distribution. Importantly, SMOTE is applied only to the training data, and not to the validation or test data. This technique generates synthetic samples for the minority classes by interpolating between existing minority class samples in the training set, ensuring that the model learns from a balanced dataset without introducing synthetic data into the evaluation phases.

Applying SMOTE only on the training data is crucial because it maintains the integrity of the validation and test sets, allowing for an unbiased evaluation of the model's performance. If SMOTE were applied to the validation or test data, it could artificially inflate performance metrics, as the model would be evaluated on synthetic data rather than real-world data distributions.

Data balancing is critical for machine learning models because it ensures that the model does not become biased towards the majority class. In an imbalanced dataset, the model might learn to always predict the majority class, leading to poor performance on the minority class. By balancing the dataset, the model learns to recognize and accurately predict both majority and minority classes, resulting in a more robust and unbiased model.

4.3.4 Feature selection

Feature selection is the process of selecting a subset of relevant features for model training. This step involves removing features that may not contribute to the model's

predictive power or that may introduce noise. In our context, we performed a feature importance investigation using a Random Forest model to evaluate the significance of each feature. Based on our findings, we selected all features except for `Timestamp` and `DLC` due to their very low importance.

The feature importance investigation involved fitting a Random Forest model on the dataset and using its built-in feature importance scores to rank each feature. The Random Forest model evaluates the importance of a feature by looking at how much the feature decreases the impurity in the decision trees. Features that contribute more to reducing impurity are considered more important. This method provides a clear indication of which features are most influential in predicting the target variable.

Upon examining the feature importance scores, we found that `Timestamp` and `DLC` had negligible contributions to the model's predictive power. Therefore, these columns were dropped to streamline the model and improve its performance. The remaining columns (`canID`, `Data0` to `Data7`) were retained as they showed significant importance as shown in Figure 4-2 below.

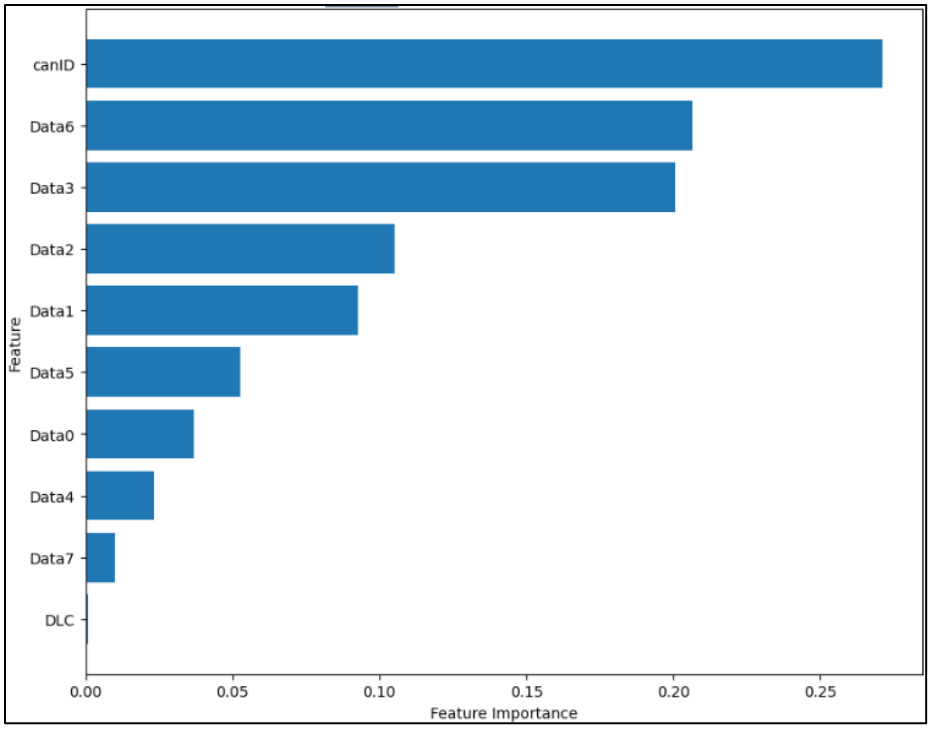


Figure 4-2. Feature Importance

This feature selection process was consistently applied across all the models: Random Forest, XGBoost, and BiLSTM. By ensuring that we train all models on the same relevant features, we maintain consistency in our approach and ensure that each model is optimized using the most significant data. This careful selection enhances the accuracy and robustness of each model, whether it be a traditional machine learning model like Random Forest and XGBoost or a more complex deep learning model like BiLSTM.

4.4 Model Performance Comparison

Following the exploratory data analysis and data preprocessing phases, this section delves into the process of hyperparameter tuning, performance evaluation, and comparison of the models. The primary goal of this praxis is not only to achieve high-

performance metrics but also to ensure minimal detection latency, crucial for real-time deployment within a vehicle network.

In testing our models, we emphasize both accuracy and efficiency. We aim to develop a model that excels in precision, recall, F1 score, and AUC-ROC, while also maintaining low latency to meet the stringent requirements of real-time detection. Testing was carried out on a machine with an Intel Xeon CPU (4-Core, 2.20 GHz) and 8 Gigabytes (GB) of memory.

The subsequent sections will present detailed reports on the performance of each model, followed by a comparative analysis. This comparison will highlight the strengths and weaknesses of each model, providing insights into their suitability for real-time deployment in automotive environments.

4.4.1 Hyperparameter Tuning

Hyperparameter tuning is a crucial step in the preparation and optimization of machine learning models. Each model has various hyperparameters that can take on different values, significantly impacting its performance. In Chapter 3, we detailed this process, particularly the use of Python's GridSearchCV() function to systematically explore a range of values for each hyperparameter. By rigorously testing different configurations, we identify the optimal values that enhance the performance of each model.

4.4.2 Random Forest Hyperparameter Tuning

Table 4-1 below lists the hyperparameters and the specific values tested for the RF model. The final values were selected after a preliminary analysis of a broader range of values to identify the model's sensitivity.

Table 4-1. Random Forest Hyperparameters

Model	Hyperparameter	Value Range	Best Value Tested
Random Forest	n_estimators	10, 100, 200	200
	max_depth	10, 20, None	20
	min_samples_split	2, 10	10
	min_sample_leaf	1, 4	1
	criterion	gini, entropy	gini

4.4.3 Random Forest Performance Metrics

Table 4-2 below presents the performance metrics for the Random Forest (RF) model across different types of car hacking attacks and normal traffic. The key metrics evaluated include Accuracy, Precision, Recall, F1 Score, Latency, Latency per Instance, and Model Size.

Table 4-2. Random Forest Performance Metrics

Attack Type	Accuracy	Precision	Recall	F1 Score	Latency	Latency/ Instance	Model Size
Flooding	99.98	100	100	100	10273 s	3.1 ms	1.71 MB
Spoofing	99.98	100	100	100			
Fuzzing	99.98	99.98	99.96	99.97			
Normal	99.98	99.99	99.99	99.99			

Flooding Attacks: The RF model achieved an accuracy of 99.98%, with perfect precision, recall, and F1 score of 100%. These results indicate that the model was highly

effective at correctly identifying flooding attacks without any misclassifications with no false positives or false negatives.

Spoofing Attacks: The model also performed exceptionally well in detecting spoofing attacks, achieving an accuracy of 99.98%, and perfect precision, recall, and F1 score of 100%. This high performance underscores the model's capability to correctly identify spoofing attacks without generating false positives or negatives.

Fuzzing Attacks: For fuzzing attacks, the RF model maintained a high accuracy of 99.98%, with precision at 99.98%, recall at 99.96%, and an F1 score of 99.97%. These metrics highlight the model's strong performance in detecting fuzzing attacks, with a low false positive rate of 0.02%.

Normal Traffic: In terms of normal traffic, the RF model achieved an accuracy of 99.98%, with precision, recall, and F1 score all at 99.99%. This indicates the model's reliability in distinguishing normal traffic from attack scenarios, ensuring a very low false positive rate of 0.01%.

The latency metric, collected during the multi-classification process, reflects the total time taken to make predictions across all attack types. The total latency was 10,273 seconds, resulting in an average latency per instance of 3.1 milliseconds. While this latency demonstrates the model's efficiency in making predictions, it does not meet the stringent requirement of being under 2 milliseconds (Avatefipour et al., 2019) for real-time deployment within vehicle networks. Additionally, the model size of 1.71 MB is relatively large for integration into vehicle systems, where models typically need to be under 1 MB to avoid overloading the vehicle's computational resources (Rolfsmeier et

al., 2003). This suggests that further optimization is necessary to reduce both the model size and latency while maintaining performance.

4.4.4 XGBoost Hyperparameter Tuning

Table 4-3 below lists the hyperparameters and the specific values tested for the XGBoost model. The final values were selected after a preliminary analysis of a broader range of values to identify the model's sensitivity.

Table 4-3. XGBoost Hyperparameters

Model	Hyperparameter	Value Range	Value Selected
XGBoost	n_estimators	50, 100, 150, 200, 250	200
	max_depth	3, 5, 7	3
	learning_rate	0.01, 0.1, 0.2	0.1
	min_child_weight	1, 3, 5	1
	reg_alpha	0, 0.1, 0.5, 1	0
	reg_lambda	0.5, 1, 1.5, 2	0.5

4.4.5 XGBoost Performance Metrics

Table 4-4 below presents the performance metrics for the XGBoost model across different types of car hacking attacks and normal traffic. The key metrics evaluated include Accuracy, Precision, Recall, F1 Score, Latency, Latency per Instance, and Model Size.

Table 4-4. XGBoost Performance Metrics

Attack Type	Accuracy	Precision	Recall	F1 Score	Latency	Latency/ Instance	Model Size
Flooding	99.99	99.99	100	100	516 s	0.156 ms	528 kb
Spoofing	99.99	99.99	100	100			
Fuzzing	99.99	99.99	99.99	99.99			
Normal	99.99	99.99	99.99	99.99			

Flooding Attacks: The XGBoost model achieved an accuracy of 99.99%, with precision of 99.99%, recall of 100%, and an F1 score of 100%. These results indicate that the model was highly effective at correctly identifying flooding attacks without any misclassifications.

Spoofing Attacks: The model performed exceptionally well in detecting spoofing attacks, achieving an accuracy of 99.99%, precision of 99.99%, recall of 100%, and an F1 score of 100%. This high performance underscores the model's capability to correctly identify spoofing attacks without generating false positives or negatives.

Fuzzing Attacks: The XGBoost model maintained a high accuracy of 99.99%, with precision of 99.99%, recall of 99.99%, and an F1 score of 99.99%. These metrics highlight the model's strong performance in detecting fuzzing attacks with a low false positive rate of 0.01%.

Normal Traffic: In terms of normal traffic, the XGBoost model achieved an accuracy of 99.99%, with precision of 99.99%, recall of 99.99%, and an F1 score of 99.99%. This indicates the model's reliability in distinguishing normal traffic from attack scenarios, ensuring a very low false positive rate.

The latency metric, collected during the multi-classification process, reflects the total time taken to make predictions across all attack types. The total latency was 516 seconds, resulting in an average latency per instance of 0.156 milliseconds. This demonstrates the model's efficiency in making real-time predictions, meeting the stringent requirement of being under 2 milliseconds for real-time deployment within vehicle networks. Additionally, the model size of 528 KB is considered lightweight and suitable for integration into vehicle systems, where models need to be under 1 MB to avoid overloading the vehicle's computational resources.

4.4.6 BiLSTM Hyperparameter Tuning

Table 4-5 below lists the hyperparameters and the specific values tested for the BiLSTM model. The final values were selected after a preliminary analysis of a broader range of values to identify the model's sensitivity.

Table 4-5. BiLSTM Hyperparameters

Model	Hyperparameter	Value Range	Value Selected
BiLSTM	num_lstm_units	50, 100, 150, 200, 250	100
	batch_size	32, 64, 128, 256	64
	dropout_rate	0.1, 0.2, 0.3, 0.5	0.1
	learning_rate	0.001, 0.01, 0.1	0.1
	optimizer	Adam, rmsprop, sgd	Adam

4.4.7 BiLSTM Performance Metrics

Table 4-6 below presents the performance metrics for the Bidirectional Long Short-Term Memory (BiLSTM) model across different types of car hacking attacks and normal traffic. The key metrics evaluated include Accuracy, Precision, Recall, F1 Score, Latency, Latency per Instance, and Model Size. Here is a detailed explanation of the results:

Table 4-6. BiLSTM Performance Metrics

Attack Type	Accuracy	Precision	Recall	F1 Score	Latency	Latency/ Instance	Model Size
Flooding	99.73	100	94.93	97.40	291 s	0.047 ms	4.03 MB
Spoofing	99.73	96.76	100.00	98.35			
Fuzzing	99.73	99.99	97.71	98.84			
Normal	99.73	99.99	100.00	99.99			

Flooding Attacks: The BiLSTM model achieved an accuracy of 99.73%, perfect precision of 100%, recall of 94.93%, and an F1 score of 97.40%. These results indicate that while the model was highly accurate and precise, it had 5.07% false negatives. The F1 score is 97.40%, reflecting a balance between precision and recall.

Spoofing Attacks: The BiLSTM model achieved an accuracy of 99.73%, precision of 96.76%, recall of 100%, and an F1 score of 98.35% for detecting spoofing attacks. These results indicate that the model was highly accurate and had no false negatives. The precision of 96.76% shows that there were some false positives, but the F1 score of 98.35% reflects a good balance between precision and recall.

Fuzzing Attacks: The BiLSTM model achieved an accuracy of 99.73%, precision of 99.99%, recall of 97.71%, and an F1 score of 98.84% for detecting fuzzing attacks. These results indicate that the model was highly accurate and precise, with a 2.29% false negative rate. The F1 score of 98.84% reflects a strong balance between precision and recall, demonstrating the model's effectiveness in detecting fuzzing attacks.

Normal Traffic: In terms of normal traffic, the BiLSTM model achieved an accuracy of 99.73%, with perfect precision and recall of 99.99% and 100%, respectively, resulting in an F1 score of 99.99%. This indicates the model's reliability in distinguishing

normal traffic from attack scenarios, ensuring very few false positives and no false negatives.

The latency metric, collected during the multi-classification process, reflects the total time taken to make predictions across all attack types. The total latency was 291 seconds, resulting in an average latency per instance of 0.047 milliseconds. This demonstrates the model's efficiency in making real-time predictions, well within the stringent requirement of being under 2 milliseconds for real-time deployment within vehicle networks. However, the model size of 4.03 MB is relatively large for integration into vehicle systems, where models typically need to be under 1 MB to avoid overloading the vehicle's computational resources.

4.5 Summary and Conclusion

The results of the three models' evaluation discussed above are summarized in Table 4-7 below. Based on the evaluation of the three models, XGBoost stands out as the most suitable model for our research. It combines high-performance metrics with both a lightweight model size and low latency, making it ideal for real-time deployment within vehicle networks. While the Bi-LSTM-AM model also demonstrated good performance, its larger size disqualifies it from consideration for this application. The Random Forest model, despite its strong performance, has a latency that exceeds the requirements for real-time applications and a larger model size. Therefore, XGBoost is chosen for its balance of accuracy, efficiency, and suitability for deployment in-vehicle environments, ensuring robust and real-time detection of car hacking attacks.

Table 4-7. Summary Results

Model	Accuracy	Precision	Recall	F1 Score	Detection Time	Latency / Instance	Real-time?	Light?
Bi-LSTM	99.73	99.19	98.16	98.64	291 s	0.047 ms	Yes	No
XGBoost	99.99	99.99	99.99	99.99	516 s	0.16 ms	Yes	Yes
Random Forest	99.98	99.99	99.98	99.99	10273 s	3.1 ms	No	No

4.6 Comparison with other published research papers

In comparison with other research models listed in Table 4-8 below, it is evident that while many models achieve high performance metrics, they often fall short in terms of latency and model size, making them less suitable for real-time deployment in vehicle networks. Their results highlight differences in performance and suitability for real-time and lightweight applications. Alalwany and Mahgoub (2022) explored several models including Voting Classifier, Stacking, and Bagging, achieving around 98.45-98.47% accuracy, but none were real-time or lightweight. Gundu and Maleki (2022) examined Random Forest (97% accuracy) and XGBoost (96% accuracy), with only the latter being lightweight. However, both lacked real-time capabilities. Mehedi et al. (2021) presented a Deep Transfer Learning model with 98.10% accuracy, but it was neither real-time nor lightweight. Moulahi et al. (2021) developed Random Forest, Decision Tree, and SVM with detection times of 1.257 ms, suitable for real-time but not lightweight. Yang et al. (2022) introduced MTH-IDS, which, despite its high accuracy (99.99%) and 0.6 ms detection time, was not lightweight. Alshammari et al. (2018) focused on KNN (97.4% accuracy) and SVM (96.5% accuracy), with KNN being real-time but not lightweight, and SVM being lightweight but with high detection time (22.56 ms).

Table 4-8. Results Comparison

Author and Year	Model	Accuracy	Precision	Recall	F1	Latency/ Time	Detection /Instance	Real time?	Light? <1MB
Alalwany and Mahgoub (2022)	Voting classifier	98.45	98.04	98.87	98.46	N/A	N/A	N/A	No
	Stacking	98.47	98.04	98.91	98.47	N/A	N/A	N/A	No
	Bagging	98.40	97.93	98.89	98.41	N/A	N/A	N/A	No
Gundu and Maleki (2022)	RF	97.00	97.00	97.00	97.0	N/A	N/A		
	XGBoost	96.00	96.00	96.00	96.0	N/A	N/A	N/A	Yes
Mehedi et al. (2021)	Deep Transfer Learning	98.10	98.14	98.04	97.83	N/A	N/A	N/A	No
Moulahi et al. (2021)	RF	98.52	98.52	98.12	-	14.93 s	1.257ms	Yes	N/A
	DT	98.19	98.19	98.17	-	14.94 s	1.257ms	Yes	N/A
	SVM	97.28	97.28	96.56	-	14.92 s	1.257ms	Yes	N/A
Yang et al. (2022)	MTH-IDS	99.99	93.74	-	96.30	365 s	0.6 ms	Yes	No
Alshammari et al. (2018)	KNN	97.4	96.3	-	93.40	912 s	1.50 ms	Yes	No
	SVM	96.5	95.7	-	93.30	13766 s	22.56 ms	No	Yes
Wang et al. (2023)	TLSIDS: DenseNet + GAN_D	-	99.90	99.97	99.93	CPU: 11ms/29ins	0.38 ms	Yes	No
						GPU: 8ms/29 ins	0.31 ms	Yes	No
Khan et al. (2024)	DNN	-	94.93	94.98	94.97	3 s	0.25 ms	Yes	No
Ullah et al. (2022)	LSTM	99.36	99.25	99.49	99.37	72 s	0.07 ms	Yes	No
	GRU	98.34	98.19	98.74	98.46	67 s	0.07 ms	Yes	No
	Hybrid DL	99.51	99.51	99.60	99.52	51 s	0.05 ms	Yes	No
Praxis	Bi-LSTM	99.73	99.19	98.16	98.64	291 s	0.047ms	Yes	No
	XGBoost	**99.99**	**100**	**99.99**	**99.99**	**516 s**	**0.16 ms**	**Yes**	**Yes**
	Random Forest	99.98	99.99	99.98	99.99	10273 s	3.1 ms	No	No

Wang et al. (2023) presented TLSIDS using DenseNet + GAN_D, showing high precision (99.90%) and recall (99.97%) with 0.38 ms detection on CPU and 0.31 ms on GPU, real-time but not lightweight. Khan et al. (2024) developed a DNN model with

94.93% precision and 0.25 ms detection time, real-time but not lightweight. Ullah et al. (2022) demonstrated LSTM, GRU, and Hybrid DL models with impressive performance and detection times (0.05-0.07 ms), all real-time but not lightweight.

Our Praxis models, BiLSTM, XGBoost, and Random Forest, show exceptional performance. The BiLSTM model achieved 99.73% accuracy and 0.047 ms detection time, suitable for real-time but not lightweight. The Random Forest model performed well with 99.98% accuracy but had a high detection time (3.1 ms), making it unsuitable for real-time applications and not lightweight. The XGBoost model we tested stands out as it not only achieves superior accuracy, precision, recall, and F1 score but also meets the stringent requirements for real-time deployment with a latency per instance of 0.16 milliseconds and a lightweight model size of 528 KB. This combination of attributes makes our XGBoost model a practical solution for detecting car hacking attacks in real-time within vehicle networks.

Chapter 5—Discussion and Conclusions

5.1 Discussion

The primary objective of this praxis is to develop a machine learning-based Intrusion Detection System (IDS) specifically tailored to detect and classify cyber intrusions within the network traffic of Controller Area Network (CAN) bus systems. To achieve this goal, three key research questions were posed, providing a roadmap to guide the study. The first research question focuses on identifying the essential features necessary for real-time classification of CAN bus network traffic. This involves determining which aspects of the network data are most indicative of normal versus intrusive behavior, ensuring that the IDS can operate efficiently and accurately in real-time scenarios. The second question addresses the feasibility of employing machine learning models for real-time classification of CAN bus network traffic. This includes evaluating the performance of various models under real-time constraints and ensuring that they can process and classify data swiftly enough to be practical in a live vehicle environment. The third research question seeks to identify the most effective machine learning model among those investigated for detecting and classifying intrusions in CAN bus traffic. This involves comparing models based on their accuracy, speed, and size to assess suitability for real-time deployment in automotive networks.

These research questions provide a structured approach to tackling the challenges of intrusion detection within the dynamic and complex environment of CAN bus networks. By systematically addressing these questions, the praxis aims to develop a robust, efficient, and practical IDS for automotive cybersecurity applications.

5.1.1 Research Question 1

RQ 1: Which features will most accurately identify and predict an attack on the in-vehicle CAN network?

Hypothesis 1 responds to research question 1 and claims that the CAN ID feature of all other features is a most relevant feature to detect in-vehicle network attacks.

RH 1: “CAN ID” feature of all features in the datasets is best for detecting in-vehicle network attacks

Based on our testing, we validated Hypothesis 1, confirming that the "CAN ID" feature is indeed the most relevant for detecting in-vehicle network attacks. As discussed in our earlier analysis of the model, the use of Shapley values for feature selection highlighted the "CAN ID" as the most significant feature in predicting attacks. This validation underscores the importance of the "CAN ID" in effectively identifying and mitigating cyber threats within the CAN bus network, thereby supporting our initial hypothesis.

5.1.2 Research Question 2

RQ 2: Can a lightweight real-time intrusion detection system be developed to detect cyber-attacks on in-vehicle CAN network with an accuracy of over 95%?

Hypothesis 2 answers research question 2 claiming a real-time lightweight intrusion detection system can be developed to detect cyber-attacks on in-vehicle CAN network with an accuracy of over 95%:

RH 2: A real-time and lightweight IDS can be utilized to detect in-vehicle network cyber-attacks with an accuracy rate of over 95%.

Based on our testing, we validated Hypothesis 2, confirming that a real-time and lightweight Intrusion Detection System (IDS) can be developed to detect cyber-attacks on the in-vehicle CAN network. We successfully developed and evaluated three models: Random Forest (RF), eXtreme Gradient Boosting (XGBoost), and Bidirectional Long Short-Term Memory (Bi-LSTM). Each model was rigorously tested for performance and real-time capabilities, demonstrating their feasibility and effectiveness in detecting cyber threats within the resource-constrained environment of automotive electronics with accuracy above 95%.

5.1.3 Research Question 3

RQ 3: Which of the eXtreme gradient boosting, Random Forest, and Artificial Neural Networks can most accurately and efficiently (in terms of model size and latency) detect in-vehicle network attacks in real-time?

Hypothesis 3 of this praxis responds to the research question 3 claiming extreme gradient boosting will most accurately and efficiently (in terms of model size and latency) detect in-vehicle network attacks in real-time:

RH 3: eXtreme gradient boosting model will most accurately and efficiently (in terms of model size and latency) detect an attack compared to Random Forest and Artificial Neural Networks models.

Based on our testing, we validated Hypothesis 3, confirming that the eXtreme Gradient Boosting (XGBoost) model most accurately and efficiently detects in-vehicle network attacks in real-time compared to Random Forest (RF) and Bidirectional Long Short-Term Memory (Bi-LSTM), an artificial neural networks model. The Random Forest (RF) model, while accurate, exhibited detection times and model sizes that were

not suitable for real-time deployment within the resource-constrained environment of automotive electronics. On the other hand, the Bi-LSTM model demonstrated much better detection times; however, its significantly larger model size made it impractical for real-time automotive deployment. This comparison reaffirms that the XGBoost model strikes the best balance between accuracy, detection time, and model size, making it the most suitable choice for real-time intrusion detection in CAN bus networks.

5.2 Conclusion

Through rigorous investigation and analysis, this praxis has successfully validated all three hypotheses proposed in response to the respective research questions. Hypothesis 1 posited that the CAN ID feature is the most relevant for detecting in-vehicle network attacks, a claim substantiated by our research findings. Utilizing Shapley values for feature selection confirmed that the CAN ID feature is indeed pivotal in accurately identifying and classifying malicious traffic within CAN bus networks. Hypothesis 2 asserted the feasibility of developing a real-time lightweight intrusion detection system for detecting cyber-attacks on in-vehicle CAN networks with a high accuracy greater than 95%. Our findings affirm the validity of this hypothesis, demonstrating that ML-based models can effectively detect and classify intrusions within CAN bus networks in real-time. The praxis proposed three model all of which were able to detect cyber-attacks on in-vehicle CAN networks. Lastly, Hypothesis 3 proposed that an eXtreme gradient boosting model would outperform Random Forest and Artificial Neural Networks in accurately and efficiently (in terms of model size and latency) detecting in-vehicle network attacks in real-time. Our research substantiates this claim, as the eXtreme

gradient boosting model exhibited superior accuracy and was more efficient in terms of model size and latency compared to the other models examined.

In conclusion, this praxis has effectively addressed and answered the three research questions posed. It has identified the CAN ID feature as crucial for detecting in-vehicle network attacks, demonstrated the feasibility of real-time intrusion detection in CAN bus networks using ML-based models, and determined that eXtreme gradient boosting is the most effective model for real-time intrusion detection within this context. These findings contribute significantly to the advancement of intrusion detection capabilities within CAN bus networks, paving the way for enhanced cybersecurity measures in automotive systems.

5.3 Contribution to Body of Knowledge

As discussed in Chapter 2: Literature Review, existing research within the CAN bus intrusion detection domain has predominantly focused on detection accuracy, often neglecting crucial aspects such as detection time and model size required for real-time deployment. Additionally, much of the existing research has overlooked the stringent real-time requirements for implementing an Intrusion Detection System (IDS) within the resource-constrained environment of a car. This praxis introduces a lightweight IDS designed to classify CAN bus network cyber-attacks while maintaining high-performance metrics. Unlike prior studies, this research rigorously evaluates the IDS's model size and performance in real-time scenarios, ensuring it is lightweight and deployable within the resource-constrained car electronics environment. By addressing both performance and real-time deployment challenges, this praxis provides a robust and efficient solution tailored for automotive security applications.

This praxis includes a comprehensive real-time performance analysis of three widely used machine learning models for CAN bus intrusion detection: Random Forest (RF), eXtreme Gradient Boosting (XGBoost), and Bidirectional Long Short-Term Memory (Bi-LSTM). Unlike previous studies that primarily focus on the static performance of these models, this research integrates model performance evaluation with real-time deployment requirements. By examining the size and detection time capabilities of these ML models within the context of CAN bus environments, this praxis addresses a gap in the current literature, offering valuable insights for both practitioners and researchers in the field. This dual focus on performance and real-time applicability enhances the practical relevance of the findings, guiding future implementations of intrusion detection systems in automotive networks.

5.4 Recommendations for Future Research

Moving forward, there are several avenues for future research that can further enhance the capabilities of our CAN bus Intrusion Detection System (IDS):

- Deploy the model on embedded electronic control units (ECU) within actual vehicles to evaluate its performance in real-world automotive environments. This will provide valuable insights into the practical applicability and efficiency of the IDS in safeguarding vehicular networks against cyber intrusions.
- Explore the IDS application to different vehicle communication datasets to assess its performance across various communication protocols mentioned earlier (LIN, FlexRay, MOST). Expanding the scope of analysis to encompass a broader range of communication protocols will help gain a more comprehensive understanding

of the IDS's effectiveness in mitigating cyber threats within diverse vehicular communication networks.

- Focus on enhancing the IDS to detect and mitigate emerging cyberattack vectors targeting vehicular networks. By staying abreast of evolving cybersecurity challenges and incorporating cutting-edge detection techniques, the IDS can remain effective in safeguarding the integrity and security of CAN bus networks against evolving cyber threats.
- Extend the IDS to encompass inter-vehicle networks and external communication channels, significantly enhancing security beyond individual vehicles to include the entire networked ecosystem. This expansion should focus on cybersecurity within vehicle-to-vehicle (V2V) and vehicle-to-everything (V2X) contexts, addressing the unique threats posed by interconnected automotive systems. By incorporating advanced detection techniques, the IDS can effectively manage the complexities of data exchange in dynamic, multi-vehicle environments, providing comprehensive protection against potential cyber-attacks across the broader spectrum of vehicle communications.

References

Alalwany, E., & Mahgoub, I. (2022). Classification of normal and malicious traffic based on an ensemble of machine learning for a vehicle can-network. *Sensors*, *22*(23), 9195. https://doi.org/10.3390/s22239195

Alshammari, A., Zohdy, M. A., Debnath, D., & Corser, G. (2018). Classification approach for intrusion detection in vehicle systems. *Wireless Engineering and Technology*, *9*(4), 79-94. https://doi.org/10.4236/wet.2018.94007

Avatefipour, O., Al-Sumaiti, A. S., El-Sherbeeny, A. M., Awwad, E. M., Elmeligy, M. A., Mohamed, M. A., & Malik, H. (2019). An intelligent secured framework for cyberattack detection in electric vehicles' CAN bus using machine learning. *Institute of Electrical and Electronics Engineers (IEEE)*, *7*, 127580-127592. https://doi.org/10.1109/ACCESS.2019.2937576

Biau, G., & Scornet, E. (2016). A random forest guided tour. *Test*, *25*, 197-227. https://doi.org/10.1007/s11749-016-0481-7

Bosch, R. (2014). *Bosch automotive electrics and automotive electronics: systems and components, networking and hybrid drive*. Wiesbaden Germany: Springer Vieweg. https://doi.org/10.1007/978-3-658-01784-2

Bozdal, M., Samie, M., Aslam, S., & Jennions, I. (2020). Evaluation of can bus security challenges. *Sensors (Basel, Switzerland)*, *20*(8), 2364. https://doi.org/10.3390/s20082364

Breiman, L. (1996). Bagging predictors. *Machine learning*, *24*, 123-140. https://doi.org/10.1007/BF00058655

Breiman, L. (2001). Random forests. *Machine learning*, *45*, 5-32. https://doi.org/10.1023/A:1010933404324

Car Crash Computing. (2017). Computer Shopper (London), (347), 12.

Casillo, M., Coppola, S., De Santo, M., Pascale, F., & Santonicola, E. (2019). Embedded intrusion detection system for detecting attacks over CAN-BUS. Paper presented at the 2019 4th International Conference on System Reliability and Safety (ICSRS), https://doi.org/10.1109/ICSRS48664.2019.8987605

Chen, M., Liu, Q., Chen, S., Liu, Y., Zhang, C.-H., & Liu, R. (2019). XGBoost-based algorithm interpretation and application on post-fault transient stability status prediction of power system. *Institute of Electrical and Electronics Engineers*, *7*, 13149-13158. https://doi.org/10.1109/ACCESS.2019.2893448

Cho, K.-T., & Shin, K. G. (2017). Viden: Attacker identification on in-vehicle networks. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, https://doi.org/10.1145/3133956.3134001

Deng, Z., Xun, Y., Liu, J., & Zhao, Y. (2022). A lightweight sender identification scheme based on vehicle physical layer characteristics. Paper presented at the ICC 2022-IEEE International Conference on Communications, https://doi.org/10.1109/ICC45855.2022.9838710

Ding, D., Zhu, L., Xie, J., & Lin, J. (2022). In-vehicle network intrusion detection system based on Bi-LSTM. Paper presented at the 2022 7th International Conference on Intelligent Computing and Signal Processing (ICSP), https://doi.org/10.1109/ICSP54964.2022.9778620

Evers, R., & Pannell, D. R. (2022). Local interconnect network (LIN) messaging between LIN partitions separated by a backbone communication network. https://patents.google.com/patent/US11463555B2/en

Fingas, J. (2016). Nissan disables its leaf remote control app (update). *Engadget*.

Frenzel, L., E. (2016). Chapter Thirty-Four - Media Oriented Systems Transport (MOST). *Elsevier Inc*, 141-142. https://doi.org/10.1016/B978-0-12-800629-0.00034-6

Gao, J. (2022). Network intrusion detection method combining CNN and BiLSTM in cloud computing environment. *Computational Iintelligence and Neuroscience*, *2022*(1), 7272479. https://doi.org/10.1155/2022/7272479

Groza, B., & Murvay, P. (2019). FlexRay. *Sensors (Basel, Switzerland)*.

Gundu, R., & Maleki, M. (2022). Securing CAN bus in connected and autonomous vehicles using supervised machine learning approaches. 2022 IEEE International Conference on Electro Information Technology (eIT), https://doi.org/10.1109/eIT53891.2022.9813985

Kang, G., Xiao, Y., Liu, J., Cao, Y., Cao, B., Zhang, X., & Ding, L. (2021). Tatt-BiLSTM: Web service classification with topical attention-based BiLSTM. *Concurrency and Computation: Practice and Experience*, *33*(16), e6287. https://doi.org/10.1002/cpe.6287

Kang, M.-J., & Kang, J.-W. (2016). Intrusion detection system using deep neural network for in-vehicle network security. *PloS one*, *11*(6), e0155781. https://doi.org/10.1371/journal.pone.0155781

Khan, M. H., Javed, A. R., Iqbal, Z., Asim, M., & Awad, A. I. (2024). DivaCAN: Detecting in-vehicle intrusion attacks on a controller area network using ensemble learning. *Computers & Security*, *139*, 103712. https://doi.org/10.1016/j.cose.2024.103712

Kim, H. K. (2021). Car-hacking dataset. Retrieved from https://ocslab.hksecurity.net/Datasets/car-hacking-dataset.

Kukkala, V. K., Thiruloga, S. V., & Pasricha, S. (2022). Roadmap for cybersecurity in autonomous vehicles. *IEEE Consumer Electronics Magazine*, *11*(6), 13-23. https://doi.org/10.1109/MCE.2022.3154346

Lampe, B., & Meng, W. (2023). Intrusion detection in the automotive domain: A comprehensive review. *IEEE Communications Surveys & Tutorials*. https://doi.org/10.1109/COMST.2023.3309864

Larson, U. E., Nilsson, D. K., & Jonsson, E. (2008). An approach to specification-based attack detection for in-vehicle networks. 2008 IEEE Intelligent Vehicles Symposium, https://doi.org/10.1109/IVS.2008.4621263

Li, X., Yu, Y., Sun, G., & Chen, K. (2018). Connected vehicles' security from the perspective of the in-vehicle network. *IEEE Network*, *32*(3), 58-63. https://doi.org/10.1109/MNET.2018.1700319

Makowitz, R., & Temple, C. (2006). Flexray-a communication network for automotive control systems. Paper presented at the 2006 IEEE International Workshop on Factory Communication Systems, https://doi.org/10.1109/WFCS.2006.1704153

Mehedi, S. T., Anwar, A., Rahman, Z., & Ahmed, K. (2021). Deep transfer learning based intrusion detection system for electric vehicular networks. *Sensors*, *21*(14), 4736. https://doi.org/10.3390/s21144736

Miller, C. (2019). Lessons learned from hacking a car. *IEEE Design & Test*, *36*(6), 7-9. https://doi.org/10.1109/MDAT.2018.2863106

Miller, C., & Valasek, C. (2015). Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, *2015*(S 91), 1-91.

Moulahi, T., Zidi, S., Alabdulatif, A., & Atiquzzaman, M. (2021). Comparative performance evaluation of intrusion detection based on machine learning in in-vehicle controller area network bus. *Ieee Access*, *9*, 99595-99605. https://doi.org/10.1109/ACCESS.2021.3095962

Müter, M., Groll, A., & Freiling, F. C. (2010). A structured approach to anomaly detection for in-vehicle networks. Paper presented at the 2010 Sixth International Conference on Information Assurance and Security, https://doi.org/10.1109/ISIAS.2010.5604050

Nagarajan, J., Mansourian, P., Shahid, M. A., Jaekel, A., Saini, I., Zhang, N., & Kneppers, M. (2023). Machine Learning based intrusion detection systems for connected autonomous vehicles: A survey. *Peer-to-Peer Networking and Applications*, *16*(5), 2153-2185. https://doi.org/10.1007/s12083-023-01508-7

Nanda, A., Puthal, D., Rodrigues, J. J., & Kozlov, S. A. (2019). Internet of autonomous vehicles communications security: overview, issues, and directions. *IEEE Wireless Communications*, *26*(4), 60-65. https://doi.org/10.1109/MWC.2019.1800503

Oladimeji, D., Rasheed, A., Varol, C., Baza, M., Alshahrani, H., & Baz, A. (2023). CANAttack: Assessing Vulnerabilities within Controller Area Network. *Sensors (Basel, Switzerland)*, *23*(19), 8223. https://doi.org/10.3390/s23198223

Prince, B. (2015). BMW patches security flaw that let hackers open doors. Retrieved from https://www.securityweek.com/bmw-patches-security-flaw-let-hackers-open-doors/.

Rajapaksha, S., Kalutarage, H., Al-Kadri, M. O., Petrovski, A., Madzudzo, G., & Cheah, M. (2023). Ai-based intrusion detection systems for in-vehicle networks: A survey. *ACM Computing Surveys*, *55*(11), 1-40. https://doi.org/10.1145/3570954

Rani, D., Gill, N. S., Gulia, P., & Chatterjee, J. M. (2022). An Ensemble-Based Multiclass Classifier for Intrusion Detection Using Internet of Things. *Computational intelligence and neuroscience*, *2022*(1), 1668676. https://doi.org/10.1155/2022/1668676

Robert, B. G. (1991). Document 50, C. A. N Specification, Postfach. *50*.

Rolfsmeier, A., Richert, J., & Leinfellner, R. (2003). A new calibration system for ECU development. *SAE transactions*, 107-116. http://www.jstor.org/stable/44699658

Roy, B., & Cheung, H. (2018). A deep learning approach for intrusion detection in internet of things using bi-directional long short-term memory recurrent neural network. Paper presented at the 2018 28th International Telecommunication Networks and Applications Conference (ITNAC), https://doi.org/10.1109/ATNAC.2018.8615294

Shin, K. G., & Cho, K.-T. (2022). Fingerprinting electronic control units for vehicle intrusion detection. https://patents.google.com/patent/US11044260B2/en

Siegel, J. E., Erb, D. C., & Sarma, S. E. (2018). A survey of the connected vehicle landscape—Architectures, enabling technologies, applications, and development areas. *IEEE Transactions on Intelligent Transportation Systems*, *19*(8), 2391-2406. https://doi.org/10.1109/TITS.2017.2749459

Smith, C. (2016). *The car hacker's handbook: a guide for the penetration tester (1st ed.)*. San Francisco, California: No Starch Press.

Song, H. M., & Kim, H. K. (2021). Self-supervised anomaly detection for in-vehicle network using noised pseudo normal data. *IEEE Transactions on Vehicular Technology*, *70*(2), 1098-1108. https://doi.org/10.1109/TVT.2021.3051026

Sun, H., Chen, M., Weng, J., Liu, Z., & Geng, G. (2021). Anomaly detection for in-vehicle network using CNN-LSTM with attention mechanism. *IEEE Transactions on Vehicular Technology*, *70*(10), 10880-10893. https://doi.org/10.1109/TVT.2021.3106940

Tangirala, S. (2020). Evaluating the impact of GINI index and information gain on classification using decision tree classifier algorithm. *International Journal of Advanced Computer Science and Applications*, *11*(2), 612-619. https://dx.doi.org/10.14569/IJACSA.2020.0110277

Taylor, A., Japkowicz, N., & Leblanc, S. (2015). Frequency-based anomaly detection for the automotive CAN bus. Paper presented at the 2015 World Congress on Industrial Control Systems Security (WCICSS), https://doi.org/10.1109/WCICSS.2015.7420322

Ullah, S., Khan, M. A., Ahmad, J., Jamal, S. S., e Huma, Z., Hassan, M. T., Pitropakis, N., Arshad, & Buchanan, W. J. (2022). HDL-IDS: a hybrid deep learning

architecture for intrusion detection in the Internet of Vehicles. *Sensors (Basel, Switzerland)*, *22*(4), 1340. https://doi.org/10.3390/s22041340

Wang, Y., Lai, Y., Chen, Y., Wei, J., & Zhang, Z. (2023). Transfer learning-based self-learning intrusion detection system for in-vehicle networks. *Neural Computing and Applications*, *35*(14), 10257-10273. https://doi.org/10.1007/s00521-023-08233-5

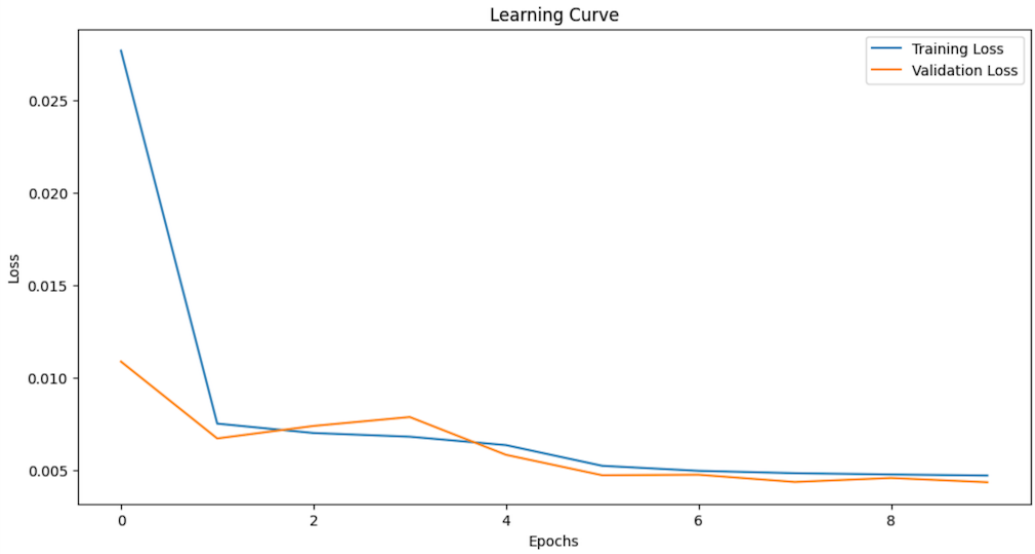
Yaji, S., Kalappa, N., & Student, M. T. (2020). Comparison of CAN, LIN, FLEX RAY and MOST in-vehicle bus protocols.

Yang, L., Moubayed, A., & Shami, A. (2022). MTH-IDS: A multitiered hybrid intrusion detection system for internet of vehicles. *Institute of Electrical and Electronics Engineers (IEEE)*, *9*(1), 616 - 632. https://doi.org/10.1109/JIOT.2021.3084796

Zeng, W., Khalid, M. A., & Chowdhury, S. (2016). In-vehicle networks outlook: Achievements and challenges. *IEEE Communications Surveys & Tutorials*, *18*(3), 1552-1571. https://doi.org/10.1109/COMST.2016.2521642

Appendix A

Figure A-1. Learning Curve 1



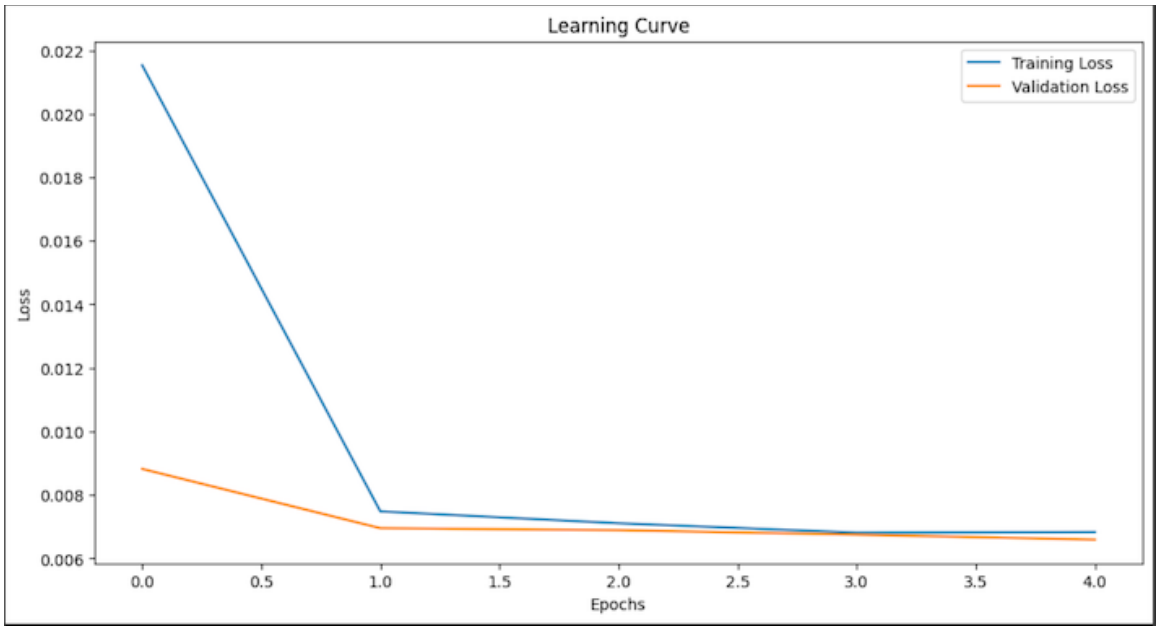
Interpreting the training loss:

Learning curve of a good fit model has a moderately high training loss at the beginning which gradually decreases upon adding training examples and flattens gradually, indicating addition of more training examples doesn't improve the model performance on training data.

Interpreting the validation loss

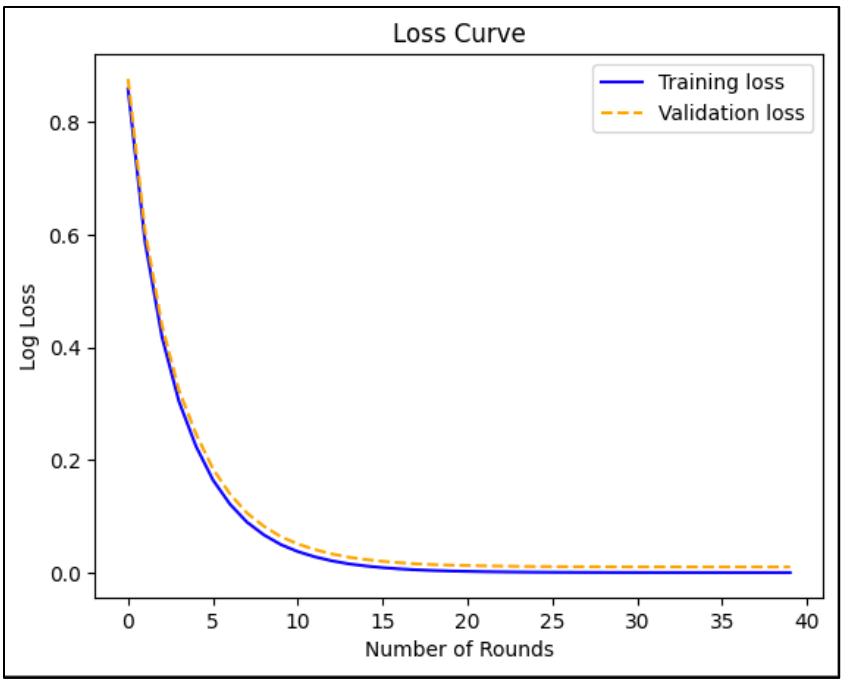
Learning curve of a good fit model has a high validation loss at the beginning which gradually decreases upon adding training examples and flattens gradually, indicating addition of more training examples doesn't improve the model performance on unseen data.

Figure A-2. Learning Curve 2



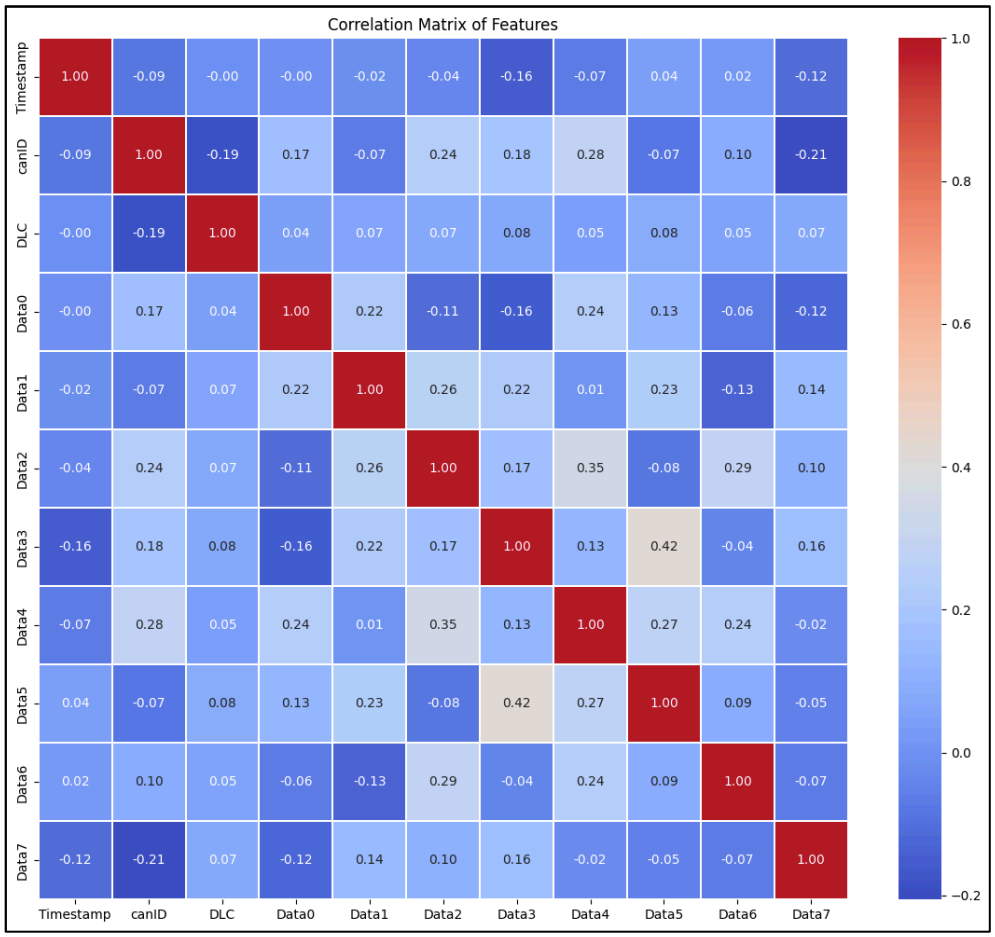
The learning curve shows no sign of overfitting, and the small fluctuations in the validation loss are typical and do not indicate overfitting.

Figure A-3. Loss Curve



Both the training and validation losses decrease significantly and converge to low values without a large gap between them. This indicates that the model has learned the patterns in the data well and generalizes effectively to the validation set, which means there are no signs of overfitting.

Figure A-4. Feature Correlation Matrix



The feature correlation matrix displayed above illustrates the correlation coefficients between pairs of features in a dataset. The values range from -1 to 1, where 1 indicates a perfect positive correlation, -1 indicates a perfect negative correlation, and 0 indicates no correlation. Observing the matrix, it is evident that the majority of

correlation values are close to 0, signifying that there is no strong linear relationship between any two features. The highest correlation value, around 0.42, is between 'Data3' and 'Data5', which is still relatively low, indicating a weak positive relationship. Similarly, the negative correlations observed, such as -0.21 between 'canID' and 'Data7', are also weak. Thus, based on this correlation matrix, we can conclude that there is no significant correlation between any two features in this dataset.