# CANLP: Intrusion Detection for Controller Area Networks using Natural Language Processing and Embedded Machine Learning

Kavya Balasubramanian

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science

University of Washington

2024

Committee:

Radha Poovendran

Payman Arabshahi

Program Authorized to Offer Degree:
Electrical and Computer Engineering

University of Washington

## Abstract

CANLP: Intrusion Detection for Controller Area Networks using Natural Language Processing and Embedded Machine Learning

Kavya Balasubramanian

Co-Chairs of the Supervisory Committee:
Radha Poovendran
Department of Computer Science & Engineering

Payman Arabshahi
Department of Electrical & Computer Engineering

The Controller Area Network (CAN) protocol is the most widely used standard in the automotive industry for in-vehicle networks. However, the CAN protocol lacks essential security features such as encryption and message authentication. Absence of such security features has been shown to make the vehicle network vulnerable to exploits by an adversary. Although multiple types of intrusion detection systems (IDS) have been developed for CAN, it can be difficult to deploy them in real-time with low latency. Many of these IDSs are unable to isolate a specific transmitting Electronic Control Unit (ECU) and CAN frame on which an attack has been mounted, which makes it challenging to design defense mechanisms.

In this thesis, we develop CANLP, a Natural Language Processing (NLP)-based intrusion detection system to determine whether each transmitted message originated from a legitimate ECU or an adversary. CANLP uses Term Frequency-Inverse Document Frequency (TF-IDF), a NLP technique to discern complex features associated with CAN data and trains machine learning models to identify *three types of attacks*- fuzzing, spoofing, and masquerade. When an attack is detected, CANLP identifies the compromised transmitter ECU and malicious CAN frame, which is important for developing resilient systems. Extensive experiments on 4 publicly available vehicle network datasets (which represent data collected from over three vehicle makes and four models)

show that CANLP performs attack classification with high F1-scores of $0.9974$. We also show that CANLP can be deployed for attack detection on resource-constrained hardware through experiments on a testbed with latency as low as $< 0.05\ ms$, hence improving the accuracy-compute tradeoff and making it perfect for real-world automotive applications.

# TABLE OF CONTENTS

Page

# LIST OF FIGURES

# LIST OF TABLES

# GLOSSARY

CAN: Controller Area Network is a high-integrity, serial communication protocol standardized by ISO 11898, designed for real-time distributed control and multiplexing in automotive and industrial systems.

NLP: Natural Language Processing (NLP) is a field of artificial intelligence that focuses on the interaction between computers and humans through natural language, enabling the processing and analysis of large volumes of text and spoken words.

TF-IDF: Term Frequency-Inverse Document Frequency (TF-IDF) is a numerical statistic used in information retrieval and text mining that reflects the importance of a word in a document relative to a collection of documents (corpus).

ADVERSARY: An attacker who attempts to exploit vulnerabilities on the CAN Bus for malicious purposes.

QUANTIZATION: the process of reducing the precision of the model's weights and activations, typically from floating-point to lower-bit representations, to optimize computational efficiency and reduce memory usage without significantly compromising accuracy.

EMBEDDED MACHINE LEARNING: the integration of machine learning algorithms directly into hardware devices, enabling data processing and decision-making capabilities on resource-constrained systems without relying on cloud computing.

T-SNE VISUALIZATION: t-distributed stochastic neighbor embedding (t-SNE) visualization is a technique for visualizing high-dimensional data in a lower-dimensional space.

# ACKNOWLEDGMENTS

For my work on CANLP, I am grateful to my co-authors Adithya Gowda Baragur, Denis Donadel, Dinuka Sahabandu, Alessandro Brighente, Bhaskar Ramasubramanian, Mauro Conti and Radha Poovendran. I am very grateful to my thesis committee members Radha Poovendran and Payman Arabshahi for their involvement, support and feedback.

I thank Zhangchen Xu and Luyao Niu for helpful discussions and feedback, as well as my peers at UW Network Security Lab for their support. This work was supported by the US National Science Foundation, the Office of Naval Research, and European Commission under the Horizon Europe Program, as part of the project LAZARUS, via grants CNS-2153136, N00014-20-1-2636, N00014-23-1-2386, and 101070303 respectively. Denis Donadel thanks Omnitech S.r.l. for supporting their research.

# DEDICATION

To my family, my favorite people, Balasubramanian, Latha and Vivek.

To my advisor, Prof Radha Poovendran and the best mentors, Bhaskar and Dinuka.

To friends, who quickly became my second family, Raja, Apoorv, Pooja, Ram, Sreeram, Tina and Sticky.

To my favorite furballs, Nolan and Jakey.

# Chapter 1

# INTRODUCTION

In-vehicle security has been a rising concern as a consequence of modern vehicles using software-driven Electronic Control Units (ECUs) and wireless connectivity. The Controller Area Network (CAN) protocol is the most popular and extensively used in-vehicle messaging standard in the automotive industry [30, 22, 12]. However, the design of the CAN protocol does not include essential security features such as encryption or message authentication, which makes the network vulnerable to attacks by an adversary [11]. An adversary has been shown to be able to exploit vulnerabilities in remote endpoints to compromise an ECU, access the in-vehicle network, and cause abnormal vehicle behavior, thus jeopardizing driver and passenger safety [13, 30, 36].

Intrusion Detection Systems (IDSs) are a commonly proposed solution to detect possible adversarial behavior on the CAN bus. IDSs for CAN are typically categorized as rule-based [52], signature-based [31], or Machine Learning (ML) based [37] depending on the methodology used. While the former two have been implemented in a wide range of settings, they have been shown to fail if an adversary takes control of an ECU before the rules of the IDS have been determined [34], or might incorrectly identify benign signals on the CAN bus as malicious [23]. ML-based IDSs [37], on the other hand, provide a scalable way to detect the presence of an adversary while handling large amounts of data transmitted on the CAN bus. ML-based IDSs also cause minimal disruption to the CAN protocol and ECUs and are hence widely utilized for in-vehicle security to detect a variety of attacks [37, 28, 7].

ML-based IDSs use contents of CAN data or derive features from a window of messages and develop models to detect attacks. Some IDSs [16] also consider periodicity of ECU transmissions to predict the next set of CAN messages and detect attacks based on significant changes observed between actual and predicted values. However, we believe that deploying these IDSs in real-time

can be challenging because: (i) Stealthy attacks are performed at the expected periodic frequency of legitimate CAN messages and timing-based IDSs are unable to detect these attacks; (ii) Attack resiliency requires identification of the individual transmitter ECU and CAN frame which mounted the attack, thus IDSs using a window of messages [28, 7, 2] cannot perform this distinction; and (iii) CAN protocol allows message transmissions at speeds of up to 1 Mbps which makes it difficult for IDSs with large ML models [56, 28, 59] to be deployed in resource-constrained hardware with low latency.

In order to overcome these challenges, we use the insight that the structure and formatting of messages on the CAN bus can be exploited to construct a human-readable format according to a set of well-defined rules in the CAN Database (DBC) file. Such transformation of the CAN messages enables interpreting this data using techniques from Natural Language Processing (NLP). The data in a CAN frame can be segmented into blocks of 8 one-byte values, 64 one-bit values, 1 sixty-four bits value, or any combination of these [24]. The Term Frequency-Inverse Document Frequency (TF-IDF) [38] encoding technique provides a way to group binary data into bits of different lengths.

We develop CANLP, an Intrusion Detection for Controller Area Networks using Natural Language Processing and Embedded Machine Learning. CANLP uses the TF-IDF [3] to extract fine-grained features from CAN data. We propose to extract character-level (1,2)-gram TF-IDF features from encoded CAN data bytes to enable capturing frame-specific fine-grained bit patterns [40]. CANLP identifies frequency patterns in CAN data with respect to each frame as well as a set of frames observed on the bus, and subsequently uses this information for real-time attack detection. CANLP analyzes individual CAN frames and identifies the type of attack as well as the transmitting ECU (considering ECUs with pre-defined unique IDs) on which the attack was mounted – thus overcoming a problem faced by ML-based IDSs considering a window of messages for evaluation [28, 7, 2].

CANLP uses the extracted features to train a Deep Neural Network (DNN) to distinguish between legitimate transmissions and adversarial behavior to perform *multi-class attack classification*. Our feature extraction technique demonstrates variability in the features among different attacks, thus allowing us to use a small DNN model which is further reduced in size using quanti-

zation techniques [18]. This enables CANLP to perform attack detection and additionally enable real-time deployment with low overhead.

Through extensive experiments on four publicly available vehicle network datasets - (i) HCRL Car Hacking: Attack and Defense Challenge (AD) [26], (ii) HCRL Car-Hacking (CH) [42], (iii) HCRL Survival Analysis Dataset for Automobile IDS (SA) [20], and (iv) Real ORNL (Oak Ridge National Laboratory) Automotive Dynamometer (ROAD) CAN Intrusion Dataset [55] - we evaluate the effectiveness of CANLP in detecting attacks. Our experimental evaluations on the CAN testbed show that CANLP detects fuzzing, spoofing, or masquerade attacks mounted on the CAN bus with a high F1-score of $0.9974$ even after model compression. We implement CANLP on a Raspberry Pi 4 Model B running at $1.8GHz$ with $1GB$ of RAM and deploy it on a CAN testbed. The latency of CANLP to classify each CAN frame is as low as $0.049ms$.

### *1.1 Summary of Research Contributions*

**The key contributions of this thesis are:**

- We develop a scheme for feature extraction of CAN messages by exploiting patterns within CAN data frame and ID correlations using the TF-IDF technique from NLP.

- We analyze most commonly used, publicly available CAN datasets and generate synthetic datasets to increase data variance and overcome skewing of ML models.

- We propose CANLP, an ML-based IDS, which gives an end-to-end framework enabling real-time detection of fuzzing, spoofing, and masquerade attacks.

- We deploy CANLP on resource-constrained hardware using a CAN Bus testbed with a transmission speed of up to 1Mbps and demonstrate the low latency, high accuracy and generalizability of CANLP.

### *1.2 Thesis Outline*

The rest of the thesis is organized as follows. Chapter 2 summarizes existing literature on CAN IDS and highlights the necessity for CANLP. Chapter 3 provides necessary background on CAN, NLP, and quantization required to follow the rest of the thesis. Chapter 4 describes the threat model we consider and Chapter 5 explains the defense capabilities. Chapter 6 presents the datasets and analysis performed on the raw data. Chapter 7 details the design of CANLP, data preprocessing steps, and our experimental setup including the CAN testbed construction. The results of our experiments and evaluation are presented in Chapter 8. Chapter 9 concludes the research and provides a brief overview of future directions.

# Chapter 2

# RELATED WORK

Many works in literature focus on intrusion detection for attacks on the CAN bus [37, 28, 7, 2, 35, 5]. The authors of [37] converted the CAN ID and data to a decimal format and used them as input features for models such as Random Tree (RT), Random Forest (RF), Stochastic Gradient Descent (SGD) with hinge loss, and Naive Bayes (NB). An alternate approach in [28] segments CAN data into sliding windows and provides them as input to a quantized convolutional neural network (CNN) deployed on a FPGA.

However, FPGA-based CAN models only evaluated two types of attacks- Denial of Service (DoS) and Fuzzing- and showed marginal improvement in performance compared to SOTA models indicated in [28]. A one-hot encoding technique for both CAN ID and Data and providing encoded data to a Self-Organizing Map (SOM) Network was proposed in [7]. However, the SOM network is computationally large and did not consider CAN data sequentially, making it impractical to deploy in real-time. In comparison, we evaluate CANLP on three different attacks- fuzzing, spoofing, and masquerade. We additionally demonstrate a way to deploy CANLP on resource-constrained hardware.

Agrawal et al. [2] combine multiple CAN frames and provide it to a CNN followed by a Long Short-Term Memory (LSTM) network. Although the use of LSTMs allows capturing temporal dependencies in the data, it incurs long processing delays and only provides an incremental increase in (classification) accuracy over SOTA models stated in [2]. The authors of [35] proposed applying a minimum-maximum transformation to decimal representations of CAN ID and Data, followed by feeding the transformed data into a LeCun Network for attack prediction. The model performance is optimized after a large number of epochs and the processing delays for predictions are not presented. In [5], the total number of packets from unique CAN ID and the size of outbound messages

from unique IDs are extracted as features and then given as input to an LSTM autoencoder based model for attack prediction. A limitation, as highlighted in [5], is that the model cannot be used for multi-class classification, which is used in CANLP.

The application of the N-gram TF-IDF feature model has found practical implementation in the domain of cybersecurity, encompassing diverse applications such as software vulnerability assessments [8, 57], and cyber threat detection [14, 50, 58, 4]. The researchers in [8] employed TF-IDF features derived from bug reports to create a tool specifically designed to identify software bugs. The authors of [57] used TF-IDF features of package manifest files to assess the security level of Android applications. In [14], they extracted TF-IDF features from process logs to construct an effective intrusion detection system for computer networks.

In this thesis, we use the insight that bytes of information transmitted on the CAN bus can be interpreted as 'words', which allows us to leverage techniques from NLP. Specifically, CANLP uses TF-IDF features extracted from data on the CAN bus to effectively distinguish normal messages from three different types of attack messages transmitted on the CAN bus. We also show that our system can be deployed on resource-constrained hardware, which can detect attacks in real-time with high F1-score and low latency.

# Chapter 3

# PRELIMINARIES

This chapter summarizes the technical background required to follow this thesis. The chapter is organized as follows. Section 3.1 provides a detailed description of CAN protocol, arbitration mechanism and details the frame format of transmissions on the CAN bus. Section 3.2 formalizes the TF-IDF NLP techniques and describes how calculations are performed to extract parameters from a document and use it as features. Section 3.3 provides an overview of the DNN classifier and associated funcations. Finally, quantization process to optimize CANLP model and make it compatible with resource-constrained hardware is highlighted in Section 3.3.

## *3.1 Controller Area Network*

The CAN protocol is one of the most widely used in-vehicle networking standards [25, 30]. The broadcast nature of CAN Bus enables the transmission and reception of messages by any ECUs on the bus to any other ECU and provides every ECU the capability to observe all ongoing transmissions. The CAN messages follow a specified frame structure which is illustrated in Fig. 3.1. It contains ID-, control-, data-, CRC- and ACK-bit(s) but does not include encryption, authentication, or timestamps. The message identifier (ID) describes the data content. The lower the binary ID, the higher is its priority. An ID consisting entirely of zeros is the highest priority message on a network because it holds the bus dominant the longest. The CAN protocol is specified such that the ECUs only transmit a message when the bus is idle [25]. The CAN protocol includes an arbitration mechanism to resolve potential conflicts when more than one ECU attempts to occupy the bus at the same time. In such a scenario, the winner at the end of the arbitration phase is the ECU that has the highest priority ID.

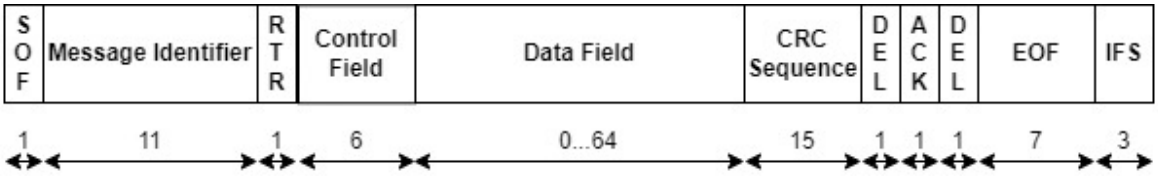


Figure 3.1: Frame Format of Standard CAN (size of each field in bits): The Start of Frame (SOF) bit marks the start of a message. Priority ID is a message identifier that sets priority of a data frame. Remote Transmission Request (RTR) defines the frame type. Data Length Code (DLC) contains the number of bytes being transmitted. A data frame can contain up to 8 bytes of ECU data. Cyclic redundancy check (CRC) contains the checksum of the preceding data for error detection. The acknowledge (ACK) bit indicates receipt of a message. The End of Frame (EOF) bit is the end of a CAN frame.

### *3.2 TF-IDF*

N-gram Term Frequency (TF) and Inverse Document Frequency (IDF) are widely used computational methods in Natural Language Processing (NLP) tasks such as text mining [38, 3, 51, 6]. An N-gram is a contiguous sequence of N terms (characters, words) in the content of a document. Typically, N-grams are extracted by moving a window of length N forward, one term at a time, along the content of document. The TF [38] of a term $\tau$ in a document is:

$$TF(\tau) = \frac{\text{No. of times } \tau \text{ appeared in the document}}{\text{No. of terms in the document}}. \tag{3.1}$$

The IDF measures informativeness of terms in a collection of documents (corpus) [38]. It assigns lower values to terms that commonly appear in the corpus as they do not contribute to distinguishing contents of documents. Conversely, higher values are assigned to less frequent terms in the corpus as they may constitute patterns inherent to document contents. The IDF of a term $\tau$ is:

$$IDF(\tau) = \log\left(\frac{\text{No. of documents in corpus + 1}}{\text{No. of documents with } \tau \text{ + 1}}\right) + 1. \tag{3.2}$$

The TF-IDF is a statistic that quantifies the importance of a term to a document in a corpus [38], and is obtained by taking the product of Eqn. (3.1) and Eqn. (3.2) as:

$$\textit{TF-IDF}(\tau) = TF(\tau) \times IDF(\tau). \tag{3.3}$$

This value of TD-IDF is termed *features*.

### *3.3 DNN*

Deep neural network (DNN) [41] classifiers take in an input $X$ and return a vector $S$ of scores for each class, where $s \in S$ (given by $s = [s_1, s_2, \ldots, s_C]$ called logits, where C is the number of classes) is a class score indicating whether $X$ belongs to class $C$. The softmax function is used to translate all these scores to probabilities, where:

$$softmax(S) = \frac{1}{\sum_{k=1}^{C} e^{s_k}} [e^{s_1}, e^{s_2}, \ldots, e^{s_C}]. \tag{3.4}$$

The categorical cross-entropy (CE) loss function is commonly used in multi-class classification tasks due to its effectiveness in measuring the difference between predicted probabilities and true class labels [19]. Mathematically, the loss function is expressed as [32]:

$$CE = -\log\left(\frac{e^{s_p}}{\sum_{k=1}^{C} e^{s_k}}\right), \tag{3.5}$$

where $s_p$ is the class score for true label/positive class for a particular input X. To learn complex patterns and relationships between features in DNN for the given input $X$, the dense layer [19] is used which is a layer of neurons (each neuron is connected to every neuron within both the preceding and subsequent layers). A DNN classifier is considered overfitted [43] when it classifies members (samples in the training set) correctly but exhibits lower accuracy when classifying nonmembers. To prevent overfitting and improve the generalization of DNN, the dropout layer randomly deactivates a fraction of neurons during training, wherein the dropout rate represents the proportion of neurons being deactivated [45].

### *3.4 Quantization*

Quantization [18] is a process by which the numerical value of a quantity or signal is represented using a smaller number of bits or discrete levels. In CANLP, we quantize weights and activations of DNN layers to achieve model compression before deployment on resource-constrained hardware. We examine two types of quantization: (i) *Float32*, in which each number is represented as a fixed decimal with 32 bits of precision, and (ii) *Dynamic range*, in which quantization levels are determined based on the range of values that the quantity can take.

# Chapter 4

# THREAT MODEL

This chapter summarizes the threat model we consider for CANLP and details assumptions about adverserial capabilities for mounting attacks on the CAN bus. This chapter also explains timing transparent and timing opaque attacks and describes different attack types. The chapter is organized as follows. Section 4.1 highlights how an adversary performs attack and Section 4.2 formalizes different attack scenarios and which ones CANLP is designed to detect.

## *4.1 Adversary Assumptions*

We assume that the adversary can intercept all CAN traffic and has the ability to inject messages of their own into the bus. We assume that the adversary performs attacks through physical access by introducing a new ECU to the bus or by taking control of a compromised ECU [12]. We consider different levels of compromised ECUs based on the control that the adversary exerts [55]. A weakly compromised ECU is a node that an adversary can disrupt from transmitting on the bus, while a strongly compromised ECU is a node over which the adversary has complete control, with the ability to send fabricated messages and access the node's memory. The adversary can strongly compromise ECUs by gaining control of the vehicle's OBD-II port.

The adversary is assumed to perform attack injection in a sporadic manner, where there is no observed pattern in attack frequency to reduce possibility of or avoid being detected. The adversary is also assumed to perform multiple attacks in no particular order and can target any component of the vehicle to cause abnormal behavior.

### *4.2 Attack Scenarios*

The objective of the adversary is to compromise the safety of the vehicle by injecting anomalous messages into the CAN bus. This goal is accomplished by carrying out one of the attacks described below. We define different classes of CAN attacks shown in Table 4.1 and highlight the attacks which CANLP aims to detect, consistent with attack definitions in [16] and [55]. Based on the

Table 4.1: Overview of the types of attacks that an adversary can carry out on CAN Bus and the importance of each of the attributes (ID, Data, and Timestamp) of CAN malicious packets for attack identification. Here the symbols ●, ◐, and ○ represent that the attack has full, partial, and no dependency respectively on the particular attribute. CANLP aims to detect Timing Opaque attacks using feature extraction techniques on ID and Data.

| **Attack Type** | **Attack** | **ID** | **Data** | **Timestamp** |
|---|---|---|---|---|
| Timing Transparent | Replay | ◐ | ◐ | ● |
| | DoS | ◐ | ○ | ● |
| Timing Opaque | Spoofing | ● | ◐ | ○ |
| | Fuzzing | ● | ● | ○ |
| | Masquerade | ● | ◐ | ○ |

frequency of message injection, attacks can broadly be classified into two types:

**1. Timing Transparent (TT) Attack**: A TT attack is any attack that is hypothetically detectable using a frequency-based method [55]. Primarily, Denial of Service (DoS) and replay attacks fall under this category as they are characterized by abnormally fast message timing [55] and can easily be detected using timing-based IDSs.

**2. Timing Opaque (TO) Attack [55]**: A TO attack, on the other hand, is any attack that cannot be detected using a frequency-based IDS because it does not disrupt normal timing or ID distributions. Attacks that may alter the overall state of the vehicle to cause out-of-control behavior fall under this category as defined below:

**Fuzzing Attack** [59] — For a fuzzing attack, messages with random IDs and arbitrary payloads are injected into the bus at the expected periodicity of CAN messages on the bus with the intent to alter the vehicle behavior. Fuzzing attacks can also be implemented by injecting messages at high frequencies which causes an impact similar to DoS attacks, where the bus is occupied with injected messages instead of legitimate ECU transmissions. However, our model considers fuzzing attacks where the adversary aims to create abnormal behavior rather than a straightforward DoS. Additionally, we assume that the adversary only injects messages with arbitrary IDs that do not conflict with legitimate ECU IDs on the bus to implement a fuzzing attack. The scenario where the adversary injects messages into a legitimate ECU falls under the scope of a spoofing attack, described below.

**Spoofing Attack** [59] — To implement a spoofing attack, an adversary injects messages using the target ID of a legitimate ECU (collected by observing traffic on the bus) and a manipulated data field. The adversary uses a specific target ID but modifies certain selected bits of the payload randomly. This attack is also referred to as targeting a signal and aims to cause unexpected behavior of vehicle components since the data appears as it if were transmitted by a legitimate ECU. On the other hand, a targeted ID attack which aims to manipulate specific functionality is implemented by injecting data frames that are designed to have a particular effect on the vehicle based on the selected target ID.

**Masquerade Attack** [55] — To implement masquerade attacks, the adversary first suspends messages of a weakly compromised target ECU, for instance, using bus-off attack [10, 15], and then injects spoofed messages with this target ID using a strongly compromised ECU, thus masquerading as the target ECU. Using this strategy, a targeted ID attack can be carried out without message conflict, thus allowing for a more stealthy attack than simple message injection. Using masquerade attacks, the adversary can not only inject attack messages from the compromised/impersonating ECU but also change the expected periodicity of messages, significantly degrading the in-vehicle network performance and increasing complexity for detection. Masquerade attacks have

been shown to cause critical issues such as non-abortable transmission requests, deadline violation, and significant priority inversion.

Due to the nature of TO attacks, specific methods such as a payload-based detector that uses data field, or a side-channel method monitoring the physical layer must be deployed [55]. In theory, binary search algorithms can be used to detect fuzzing attacks. However, our defense model is designed for multi-attack classification on the bus, necessitating the use of scalable models capable of distinguishing between various attack types. CANLP is primarily designed to detect TO attacks using feature extraction techniques on the CAN payload and work against adversaries implementing stealthy attacks. In this paper, we develop CANLP to detect fuzzing, spoofing, and masquerade attacks. We describe defense capabilities of CANLP for these attacks in Chapter 5.

# Chapter 5

# DEFENSE MODEL

This chapter presents capabilities of the defense and resources available in Section 5.1. The metrics used to evaluate the effectiveness of CANLP are discussed in Section 5.2.

## *5.1 Defense Capabilities*

The defense has limited real-time computational resources and uses ML to train a model or re-train publicly available models to achieve high F1-score for attack classification. We further assume that the defense is aware that the adversary performs different types of attacks on the CAN Bus and has knowledge of these attacks. However, the defense has no knowledge of the order or frequency in which these attacks are performed. We assume that the defense has access to a set of CAN packets whose labels are known for model training. Our objective is to develop an IDS that detects fuzzing, spoofing and masquerade attacks on the CAN Bus in real-time and can be deployed inside a vehicle.

## *5.2 Performance Metrics*

We assume that CANLP returns a 'Normal' label for legitimate messages and identifies the type of attack for other messages. Each type of attack is considered as a separate class and each of the chosen datasets for our experiments have different combinations of the attacks described above.

**True Positive Rate (TPR)**: The fraction of attack packets that are classified correctly [44].

**False Positive Rate (FPR)**: The fraction of legitimate packets that received an incorrect prediction, hence raising a false alarm [44].

**F1-score:** An effective IDS should have a high TPR and a low FPR (minimizing false alarms).

Defining $TNR = 1 - FPR$, the F1-score [44] combines TPR and TNR as:

$$F1 = \frac{2 \times TPR \times TNR}{TPR + TNR}. \tag{5.1}$$

**Accuracy:** The number of correct predictions by the IDS against the total number of predictions it makes [44].

**Latency:** The time taken by the hardware to perform data pre-processing, run the prediction model, and provide the output using received CAN data frame.

# Chapter 6

# DATASETS

This chapter describes all four datasets used to train and evaluate CANLP and highlights the contents of each in Section 6.1. Section 6.2 formalizes analysis of each dataset and details the observed issues and related consequences on models.

## *6.1 Dataset Description*

To evaluate CANLP, we use four datasets that are popular in literature and each of them is described below:

**Car Hacking: Attack and Defense Challenge (AD) Dataset [26]:** The AD dataset consists of data collected from a Hyundai Avante CN7 under both dynamic driving and stationary conditions, which allows our model to analyze adversarial behavior irrespective of the vehicle state. The data contains four primary features: Timestamp, CAN Arbitration_ID, DLC, and Data_Field. Two distinct labels are used to categorize the data. The primary label, 'class', indicates whether the captured frame is benign or malicious; labeled as 'Normal' or 'Attack' respectively. For the CAN frames identified as an 'Attack', a secondary label termed 'subclass' is used to represent the type of attack. The dataset classifies attacks into four predominant categories: DoS/ flooding, spoofing, replay, and fuzzing attacks.

**HCRL Car Hacking (CH) Dataset for Intrusion Detection [42]:** The CH dataset contains CAN data collected from an unspecified real vehicle and has four features similar to the AD dataset: Timestamp, CAN Arbitration_ID, DLC, and Data_Field. The dataset consists of three different types of attacks: DoS/ flooding, fuzzing, and spoofing attacks which are available as individual CSV files. The spoofing attacks are further divided into Revolutions Per Minute (RPM) gauge and gear spoofing based on the vehicle component that the attacker aims to compromise. The dataset

contains two distinct labels- 'T' or 'R', where 'T' represents injected message while 'R' represents a normal message.

**HCRL Survival Analysis Dataset for Automobile IDS (SA) [20]:** The SA dataset captures in-vehicle network traffic from three distinct car models: the Hyundai Sonata, KIA Soul, and Chevrolet Spark collected during stationary state of each vehicle. The datasets include Timestamp, CAN Arbitration_ID, DLC, and Data_Field features. Each vehicle dataset is split into 'Normal' data and four attack categories, namely: DoS/ flooding, fuzzing, malfunction, and replay attacks available as individual CSV files for each car. This dataset uses the same labelling as the CH dataset.

**Real ORNL Automotive Dynamometer (ROAD) CAN Intrusion Dataset [55]:** The ROAD dataset is a compilation of real-world vehicle attacks where each injected attack is physically tested for its impact on the vehicle. The vehicle's make and model are undisclosed. All attacks were performed on a dynamometer, simulating real driving conditions. The ambient data was collected both from the dynamometer and actual road conditions in various driving scenarios. The dataset includes fuzzing and fabrication attacks, and simulated stealthy masquerade attacks. Due to the complexity of implementing masquerade attacks, no real CAN data with these attacks has been made publicly available. The dataset also contains accelerator attacks which cause abnormal accelerator behavior out of the driver's control. The accelerator attack captures have no injected messages or disclose how the attacks were performed, but simply record the CAN data when the vehicle is in this state.

Table 6.1: Number of normal, attack and total CAN messages in the AD (Driving), AD (Stationary), CH, SA (all modified as per Sec. 6.1) and ROAD datasets

| **Dataset** | **#Normal** | **#Fuzzing** | **#Spoofing** | **#Masquerade** | **#Total** |
|---|---|---|---|---|---|
| AD (Driving) | 500078 | 45474 | 164409 | N/A | 709961 |
| AD (Stationary) | 73955 | 44405 | 156337 | N/A | 274697 |
| CH | 81306 | 491847 | 475422 | N/A | 1048575 |
| SA | 236787 | 237923 | 231824 | N/A | 706534 |
| ROAD | 218756 | N/A | N/A | 18080 | 236836 |

### *6.2 Dataset Analysis*

During our evaluation of the datasets, we observed a limitation in the spoofing attack data points for AD, CH and SA datasets. The datasets contain only $n\ (< 10)$ unique spoofing attack messages, which have been repetitively injected into the vehicular system at different intervals to perform the attack. While this represents a legitimate spoofing attack, it limits the dataset to very specific cases of spoofing which restricts any variability in the data frames on the bus. This redundancy inherently biases ML models, leading to overfitting on the recurrent data points and patterns. The identical attack data points are also present in the testing set, which should ideally contain only data that is distinct from training set data to correctly evaluate the performance of the model. We noticed that certain SOTA ML-Based IDS using these datasets without modification produce skewed experimental results with F-1 scores reaching an absolute 100% due to extreme overfitting. To confirm our observations, we ran experiments on these datasets and achieved F-1 scores of 100% for spoofing attacks for different ML models we tested upon. In a real-world scenario where attack patterns may be injected in different ways, redundancy in these datasets compromises the generalizability of IDS models and provides a misleading interpretation of the model's F1-score in predicting the attack types.

# Chapter 7

# CANLP MODEL

This chapter provides a breakdown of the different components of CANLP IDS. The CANLP framework processes CAN bus data through a series of methodical steps to detect and classify adversarial attacks. The CANLP framework processes data on the bus by first converting raw CAN frames into a human-readable format for NLP. It employs the TF-IDF method to extract features, which are then fed into a DNN model to classify frames as legitimate or malicious, identifying specific attack types. The optimized DNN model is compressed using quantization and deployed on a Raspberry-Pi which can be integrated into a vehicle's CAN Bus by simply plugging it into the OBD-II port. It can also be integrated inside the vehicular CAN system if the maker chooses a more seamless build.

## *7.1 Dataset Pre-Processing*

**Data Synthesis:** To overcome the limitations observed and expand the dataset for better evaluation of our model, we synthesized spoofing attack data through a series of steps:

1. *Target CAN ID Identification*: We identified a specific set of legitimate CAN IDs for our spoofing attack. These target IDs are representative of the ECUs that the adversary has selected to launch the attack after observing data on the bus for a period of time.

2. *Data Frame Randomization*: Once the target CAN IDs have been identified, we selected all the legitimate messages already transmitted by each of the target ECUs. From each message, three bytes of the data frame were chosen and randomized. While each modified payload might not directly cause an impact on the vehicle due to randomization, we generate synthetic attack data as a targeting signal attack (as described in Sec 3.2) or spoofing attack with the aim of preventing data transmission from legitimate ECUs to vehicle components.

We generate the synthetic data to mimic targeting signal attacks as closely as possible and ensure reduced redundancy in the dataset. This randomization technique was chosen after recursively testing the synthetically generated data on different models to ensure that it does not skew model training or performance. We inject the synthesized data points into the datasets in a random manner since our model is time agnostic and focuses on the CAN IDs and data frames.

**Data Processing for Feature Extraction:** In our pre-processing phase, we consider the four datasets separately. From each dataset, we only extract the CAN ID, the data frame -as *Data_Field*- and the associated *Attack Type*, which is the provided label. We rename each label to represent the type of attack. Within each dataset, we combine all the subsets containing individual attacks (by consolidating the CSV files) to evaluate a true multi-class attack classification model. Additionally, we traverse the dataset to remove all flooding and replay attacks since we our focus is to detect TO attacks. We also remove all duplicate frames to prevent any form of redundancy in the datasets and ensure that our model evaluation is not skewed. We define a frame as a duplicate if another frame in the dataset, belonging to the same class (both normal and attack classes), and have identical CAN ID and data frame values.

Since the number of attack messages is typically a small fraction of legitimate messages, we perform data balancing to mitigate any bias towards the majority class (here legitimate messages) during DNN training [47]. This process of data balancing is achieved through a series of steps: First, we identify distinct labels in the dataset representing different attack categories. Then, we balance the number of data points in each class by randomly removing data points from classes having higher number of data points. We split the balanced dataset into training and testing subsets with 80% - 20% split, ensuring sufficient data for training. The training subset is used to perform extraction of TF-IDF features which are then fed to the model for learning.

### *7.2 Feature Extraction*

As noted in Section 3.1, a raw CAN data frame is transmitted in a hexadecimal format and is a collection of consecutive bytes representing signal information for in-vehicle components. CAN

DBC is used to decode each raw frame data in a bitwise manner to interpret signal definition as physical parameters [24]. Therefore, we choose a character as a term while using N-gram TF-IDF for extracting features from a CAN data frame, specifically using the 'Message Identifier' and 'Data Field' bits. Our proposed character-level N-gram TF-IDF feature extraction interprets signal definition as physical parameters and identifies bit patterns corresponding to CAN data within a single frame as well as an overall set of frames observed on the bus.

Our intuition is that adversary-manipulated CAN data related to TO attacks might exhibit different bit orderings or consecutive bit patterns compared to the patterns found in normal CAN messages. We observe that such consecutive bit patterns of length N can be captured through N-gram units [40]. The total number of required TF-IDF features to represent each CAN frame using this technique and number of trainable DNN parameters are shown in Table 7.1.

Table 7.1: Number of features and number of trainable DNN parameters for 1-gram, (1,2)-gram and (1,2,3)-gram Character (Char) level TF-IDF features.

| **Granularity of Features** | **No of TF-IDF Features** | **Trainable Parameters** |
|---|---|---|
| Char 1-gram | 16 | 10755 |
| Char (1,2)-gram | 272 | 43523 |
| Char (1,2,3)-gram | 4368 | 567811 |

In CANLP, we employ the following methods to extract features from CAN ID and Data_Field for multi-class classification. The CAN ID is converted from hexadecimal to decimal format, followed by normalization from the range of 0 to 2048 to a new range of 0 to 1, and then used as a feature. For the Data_Field, we consider encoded Char (1,2)-gram TF-IDF for feature extraction followed by L2-normalization [9] for multi-class classification. Our approach involves the following steps: (i) identify the smallest unit that has meaning (the NLP equivalent of a word) within the CAN data frame; (ii) identify the fixed-length patterns that need to be extracted from the data

Table 7.2: F1-scores (F1) for four different datasets for five machine learning models -Support Vector Machines (SVM), Logistic Regression (LR), Gaussian Naive Bayes (GNB), Decision Trees (DT), Random Forest (RF), and a simple Deep Neural Network (DNN). We observe that the DNN showcases the best performance compared to other models, exhibiting higher average F1-score for three out of five datasets and close to best F1-score for the remaining datasets.

| **Model Type** | **Model** | **F1 AD (Driving)** | **F1 AD (Stationary)** | **F1 CH** | **F1 SA** | **F1 ROAD** |
|---|---|---|---|---|---|---|
| Machine Learning | SVM | 0.815 | 0.921 | 0.971 | 0.873 | 0.982 |
| | LR | 0.816 | 0.925 | 0.971 | 0.884 | 0.976 |
| | GNB | 0.737 | 0.836 | 0.865 | 0.787 | 0.934 |
| | DT | 0.967 | 0.961 | **0.998** | 0.974 | 0.991 |
| | RF | **0.968** | 0.979 | 0.995 | 0.973 | 0.995 |
| Deep Learning | **DNN** | 0.964 | **0.979** | 0.997 | **0.976** | **0.997** |

frame; (iii) form a frequency vector of all possible lengths that can be used as the set of features for a CAN frame. The frequency vector must meet the following criteria [40]:

(i) Length of data frame should not affect frequency values.

(ii) Frequency values corresponding to common patterns found in data frames (considered as noise) should be reduced, as they do not contribute to distinguishing between legitimate and attack messages.

(iii) Values corresponding to frequent patterns appearing only in a small subset of the dataset should be boosted since such patterns will have a higher probability of being attack-specific patterns.

***Remark**: Although the total number of features for Char(1,2,3)-gram are significantly higher than Char (1,2)-gram, F1-scores obtained using each method are comparable. On the other hand, the Char 1-gram yields lower F1-scores since it does not contain sufficient features to train the DNN model. We opt to use the Char (1,2)-gram in our experiments to maximize the efficiency while optimizing the allocated compute resource and latency values.*

Table 7.3: F1-scores (F1) for four different datasets for five machine learning models -Support Vector Machines (SVM), Logistic Regression (LR), Gaussian Naive Bayes (GNB), Decision Trees (DT), Random Forest (RF), and a simple Deep Neural Network (DNN) including Post Quantization (PQ) F1 scores using Float32 and Dynamic quantization techniques generated using **Char level 1-gram TF-IDF features**.

| **Model Type** | **Model** | **F1 AD (Driving)** | **F1 AD (Stationary)** | **F1 CH** | **F1 SA** | **F1 ROAD** |
|---|---|---|---|---|---|---|
| **Machine Learning** | SVM | 0.735 | 0.820 | 0.869 | 0.739 | 0.914 |
| | LR | 0.738 | 0.825 | 0.873 | 0.740 | 0.914 |
| | GNB | 0.749 | 0.829 | 0.856 | 0.747 | 0.948 |
| | DT | 0.946 | 0.957 | 0.994 | 0.947 | 0.969 |
| | RF | 0.959 | 0.972 | 0.994 | 0.956 | 0.978 |
| **Deep Learning** | DNN | 0.910 | 0.959 | 0.985 | 0.937 | 0.972 |
| | Float32 PQ | 0.910 | 0.959 | 0.985 | 0.937 | 0.972 |
| | Dynamic PQ | 0.909 | 0.959 | 0.985 | 0.937 | 0.972 |

### *7.3 Learning Models*

The features extracted from CAN frames are subsequently fed into learning models to perform classification of each frame as a legitimate transmission or an attack, while also predicting the type of attack mounted on the frame.

We train five machine learning (ML) models for classification, including Support Vector Machine (SVM), Logistic Regression (LR), Gaussian Naive Bayes (GNB), Decision Tree (DT), Random Forest (RF) [21]. We also train a DNN to effectively extract features at different levels of abstraction which allows them to learn more complex patterns when compared to traditional ML algorithms [33] and provide techniques for easy deployment on resource-constrained hardware [48]. Our DNN model uses three dense layers with Rectified Linear Unit (ReLu) activation [1] and

Table 7.4: F1-scores (F1) for four different datasets for five machine learning models -Support Vector Machines (SVM), Logistic Regression (LR), Gaussian Naive Bayes (GNB), Decision Trees (DT), Random Forest (RF), and a simple Deep Neural Network (DNN) including post quantization F1 scores using Float32 and Dynamic quantization techniques generated using **Char level (1,2,3)-gram TF-IDF limited to (256,256) features for 2- and 3-grams respectively**.

| **Model Type** | **Model** | **F1 AD (Driving)** | **F1 AD (Stationary)** | **F1 CH** | **F1 SA** | **F1 ROAD** |
|---|---|---|---|---|---|---|
| **Machine Learning** | SVM | 0.867 | 0.957 | 0.982 | 0.912 | 0.992 |
| | LR | 0.868 | 0.957 | 0.982 | 0.923 | 0.981 |
| | GNB | 0.735 | 0.805 | 0.864 | 0.783 | 0.956 |
| | DT | 0.998 | 0.974 | 0.996 | 0.976 | 0.992 |
| | RF | 0.969 | 0.980 | 0.996 | 0.976 | 0.996 |
| **Deep Learning** | DNN | 0.966 | 0.978 | 0.998 | 0.978 | 0.997 |
| | Float32 PQ | 0.966 | 0.978 | 0.998 | 0.978 | 0.997 |
| | Dynamic PQ | 0.966 | 0.978 | 0.998 | 0.978 | 0.997 |

one dropout layer. The first two dense layers consist of 128 and 64 units, while the final layer is adjusted to match the number of attacks in dataset. A dropout layer with a rate of 0.15 is incorporated to prevent overfitting and to improve learning efficiency during training. The final dense layer employs softmax activation using Eq. (3.4) to predict the type of attack.

To train the model, Adam optimizer [29] with a constant learning rate of 0.001 is employed with the categorical cross-entropy loss function from Eq. (3.5). The model is trained for 30 epochs with a batch size of 128. The training data is further split into training and validation in an 80:20 ratio for fine-tuning the DNN. The model is implemented using Keras [27], leveraging standard Keras functions for defining architecture, training, and evaluating the DNN.

### *7.4 Deployment on Hardware*

To evaluate and validate our experiments in real-time, we use post-training quantization techniques to further compress our learning model before deployment on the CAN testbed. In order to achieve an optimal balance between F1-score and latency, the model is compressed by quantizing the model's weights using float32 and dynamic quantization employing Tensorflow Lite [48].

To deploy the complete feature extraction process and quantized DNN model on hardware for real-time inference, the following approach is used: (i) CAN ID is normalized from the range of 0 to 2048 into 0 to 1. (ii) For Data_Field, the feature extraction process involves utilizing IDF values obtained from the training data. By calculating TF using Eqn. (3.1) for incoming data and using those IDF values, TF-IDF values are derived using Eqn. (3.3). These calculated TD-IDF features are then normalized using L2-normalization. (iii) Finally, both normalized CAN ID and normalized TD-IDF values are fed as inputs to the quantized DNN model. We use Tensorflow Lite C API [49] to integrate the quantized model on hardware.

# Chapter 8

# EXPERIMENTS AND RESULTS

This chapter presents results of the experiments evaluating CANLP on the datasets and models described in Chapter 6. This chapter is organized as follows. Section 8.1 formalizes the selection of optimal DNN weights followed by evaluation of the DNN after compression using float32 and dynamic quantization techniques in Section 8.2. Furthermore, Section 8.3 highlights the results of evaluation on the constructed CAN testbed. Section 8.4 provides a comparison of results with SOTA models and a comprehensive analysis and highlights how CANLP outperforms other IDS based on the chosen metrics. Finally, Section 8.5 details how the quality of features obtained post TF-IDF processing contribute to the best accuracy-latency tradeoff achieved by CANLP. The psuedocode for each of the modules in the CANLP system are presented in Appendix A.

We evaluated CANLP on four datasets using five different machine learning (ML) algorithms- SVM, LR, GNB, DT, and RF. We also used the DNN described in Section 7.3. We use character-level (1,2)-gram features to perform multi-class classification. Figure 8.3 illustrates the classification accuracies against varying window sizes for TF-IDF features across five models (SVM, LR, GNB, DT, RF) at different attack packet thresholds. The results clearly indicate that a window size of 1 yields the highest accuracy across all models, making it the optimal choice for further analysis and model training. Table 7.2 presents F1-scores achieved by the different algorithms on the four datasets. The DNN achieves best performance compared to other models- it has the highest F1-score for three datasets and close to best F1-score for the remaining datasets. Hence, we choose DNNs for deploying CANLP on hardware.

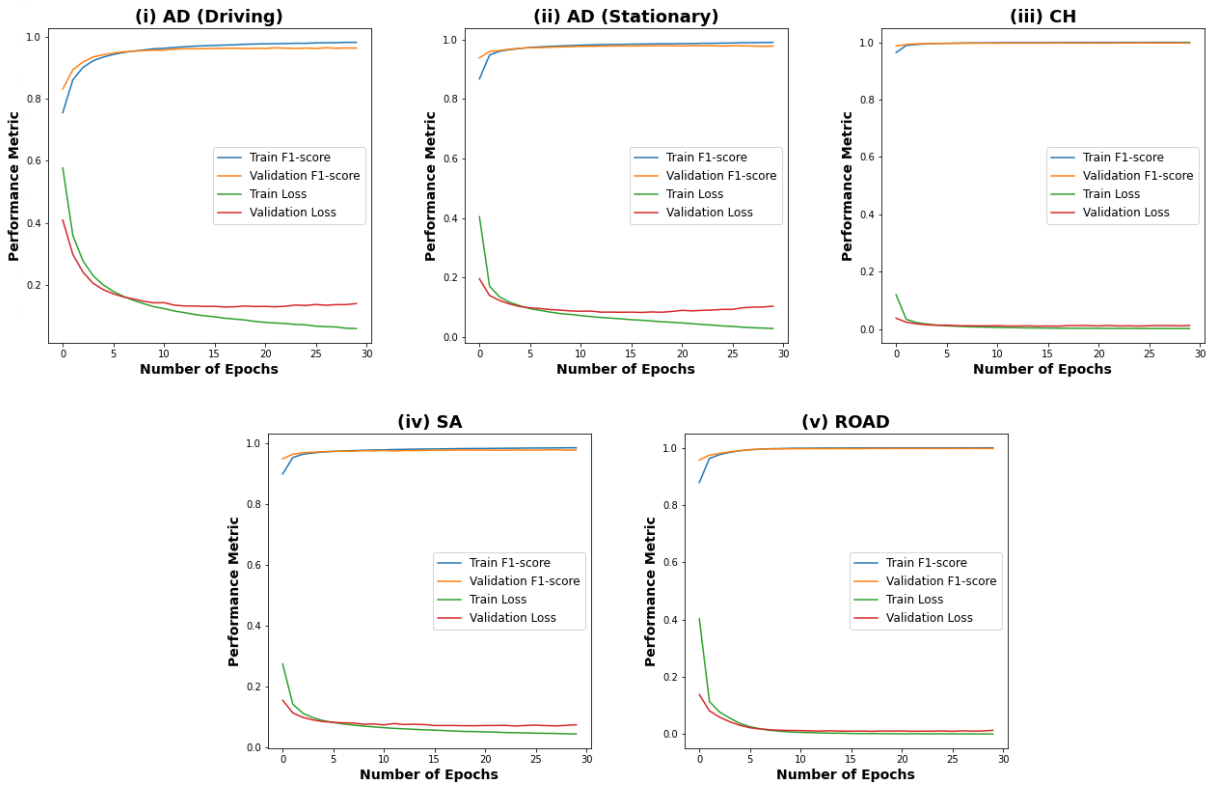


Figure 8.1: This figure plots four performance metrics- (a) training F1-score (blue), (b) validation F1-score (orange), (c) training loss (green), and (d) validation loss (red)- vs. the number of epochs for the **DNN** when trained on (i) AD (Driving), (ii) AD (Stationary), (iii) CH, (iv) SA and (v) ROAD datasets. The proximity between the training and validation F1-score curves indicates that the DNN has very minimal overfitting. Additionally, the observed reduction in both training and validation losses over the course of the epochs indicates that the selected learning rate is appropriate for the model training process. In order to determine the optimal model for deployment of CANLP, the model weights corresponding to the epoch with the highest validation F1-score are selected.

Table 8.1: Classification F1-scores before (**F1 Pre-Quant.**) and after **(F1 Post-Quant.)** quantization, model size after quantization (**Post-Quant Model Size**), and model reduction percentage after quantization (**Post-Quant Model Reduction**) for both float32 and dynamic quantization for the AD (Driving), AD (Stationary), CH, SA and ROAD datasets. Since quantization has a negligible impact on F1-score, deploying a quantized model on hardware enables achieving a high F1-score in real-time. We use dynamic quantization for CANLP due to its improved PQMR value.

| **Dataset** | **F1** | **F1 (Post-Quant.)** | | **Post-Quant Model Size** | | **Post-Quant Model Reduction** | |
|---|---|---|---|---|---|---|---|
| | | **Float32** | **Dynamic** | **Float32** | **Dynamic** | **Float32** | **Dynamic** |
| AD (D) | 0.964 | 0.964 | 0.964 | | | | |
| AD (S) | 0.979 | 0.979 | 0.979 | | | | |
| CH | 0.997 | 0.997 | 0.997 | 171.75KB | 45.58KB | 68.23% | 91.57% |
| SA | 0.976 | 0.976 | 0.976 | | | | |
| ROAD | 0.997 | 0.997 | 0.997 | | | | |

### *8.1 Selection of Best Model Weights for DNN*

We train the DNN model for 30 epochs across all the datasets. Fig. 8.1 plots training (blue) and validation (orange) F1-scores and training (green) and validation (red) loss function values. The proximity between training and validation F1-score curves shows that the DNN has very minimal overfitting [46]. Furthermore, the reduction in values of both training and validation losses over 30 epochs indicates that the model has been trained effectively. In order to determine the optimal model for deployment of CANLP, the model weights corresponding to the epoch with the highest validation F1-score are chosen ($\approx 5$ epochs in most cases).

### *8.2 Evaluation of DNN after Quantization*

To assess effectiveness of CANLP, F1-scores of quantized model are compared with the original model for the same test dataset. The quantization process results in a reduction of the original

Table 8.2: This table shows the F1-Score and mean and standard deviation (Std) of latency for CANLP and other SOTA hardware-deployed models when tested on the CH dataset. Here, NR refers to Not Reported. The F1-score and latency of the models which perform binary classification (MTH-IDS [56] and MA-QCNN [28]) have been averaged and summed up respectively across all the attacks for fair comparison against the multi-class classification models presented in ACGAN [59] and CANLP (Our Work). While the F1-scores cannot be directly compared due to the incorporation of synthesized data into our dataset, the achieved F1-score remains comparable to those achieved by other SOTA models. Our findings highlight that CANLP outperforms SOTA models in terms of latency and maintains a comparable F1-score across all attacks. Consequently, these results underscore CANLP as the optimal choice for real-time deployment.

| **Method** | **Platform** | **CPU** | **F1** | **Latency** | |
|---|---|---|---|---|---|
| | | | | **Mean** | **Std** |
| MTH-IDS | RPi 3 Model B | 1.2 GHz Broadcom BCM2837 | 0.999 | 1.722 ms | NR |
| MA-QCNN | Zynq FPGA | 1.3 GHz Arm Cortex-A53 MPCore | 0.996 | 1.290 ms | NR |
| ACGAN | RPi 4 Model B | 1.8 GHz ARM Cortex-A72 | 0.992 | 0.538 ms | 0.030 ms |
| **CANLP** | **RPi 3 Model B** | **1.2 GHz Broadcom BCM2837** | **0.997** | **0.213 ms** | **0.030 ms** |
| **CANLP** | **RPi 4 Model B** | **1.8 GHz ARM Cortex-A72** | **0.997** | **0.049 ms** | **0.011 ms** |

model size from 540.57KB to 171.75KB when float32 quantization is employed, and a further reduction to 45.58KB when dynamic quantization is utilized. We define a metric called *Post-Quant Model Reduction (PQMR)* to quantify the reduction in the size of the model as:

$$\text{PQMR} = \frac{\text{OMS} - \text{PQMS}}{\text{OMS}} \times 100\%, \tag{8.1}$$

where Post-Quant Model Size (PQMS) and Original Model Size (OMS) are the model size after and before quantization.

Table 8.1 shows F1-score before (F1 Pre-Quant.) and after (F1 Post-Quant.) quantization, PQMS, and PQMR for float32 and dynamic quantization for all datasets. We observe close to zero

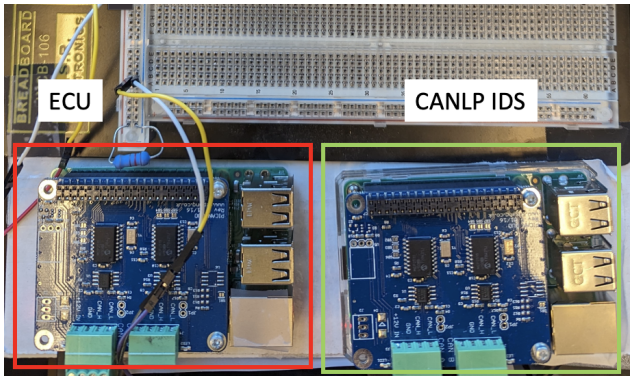


Figure 8.2: A picture of our CAN bus testbed. It has a Raspberry Pi 3 Model B equipped with a PiCAN2 Duo hat (Red) to send real-world traffic to the bus. A Raspberry Pi 4 Model B with PiCAN2 Duo hat (Green) reads data sent through CAN Bus and uses CANLP to make attack predictions.

reduction in F1-scores for both float32 and dynamic quantization, indicating that the compressed model retains its predictive power despite significant reduction in size. We choose dynamic quantization for hardware deployment of CANLP due to its better PQMR value.

### *8.3 CAN Testbed Evaluation*

To prove feasibility of deploying CANLP in a real scenario, we implemented it on a CAN testbed, shown in Fig. 8.2.

To simulate bus traffic from different ECUs, we used a Raspberry Pi 3 Model B [39] equipped with a PiCAN2 Duo hat [17]. We program it to replay one dataset at a time, thus generating traffic in the CAN bus. As per specifications [25], our bus is terminated with $120\Omega$ resistors and can support different bandwidths up to 1Mbps. The IDS is implemented and deployed on a Raspberry Pi equipped with PiCAN2 Duo hat [17] as an interface with the bus since it is a low-cost (starting from $35) and popular microcomputer with sufficient resources required for our IDS. For real world implementation, this component can simply be connected to the OBD-II interface inside an automotive vehicle and function as an IDS with minimal modification to internal systems. For fair

comparison with SOTA, we tested on two distinct versions individually: a Raspberry Pi 4 Model B to compare with [59], and a Raspberry Pi 3 Model B for comparison with [56]. The Raspberry Pi 4 (Pi 3) Model B has a CPU running at 1.8GHz (1.2GHz) and 1GB (1GB) of RAM.

When the quantized model is deployed on the Raspberry Pi 4 Model B on the testbed, we observe that the latency of attack prediction has a mean of 0.0492ms with a standard deviation of 0.0011ms for a sample of 27000 packets sent consecutively on the 1Mbps bus. Moreover, when the quantized model is deployed on the Raspberry Pi 3 Model B in the same setup, latency is observed to have a mean of 0.2133ms and a standard deviation of 0.0030ms.

### *8.4 Comparison with SOTA Models*

A comprehensive analysis of CANLP and for other SOTA hardware-deployed models is presented in Table 8.2 when tested on the CH dataset for the performance metrics: F1-score and Latency. Although direct comparison of F1-scores is not possible due to the addition of synthesized data into our dataset, the attained F1-score still aligns closely with those accomplished by other state-of-the-art models. The results of our study emphasize that CANLP demonstrates superior performance compared to the SOTA models in latency, while also maintaining a comparable F1-score across all attack scenarios. Hence, these findings strongly advocate CANLP as the optimal choice for real-time deployment.

### *8.5 Quality of Features using t-SNE*

t-Distributed Stochastic Neighbor Embedding (t-SNE) [54] is a technique for dimensionality reduction that is particularly well suited for the visualization of high-dimensional datasets. The technique can be implemented via Barnes-Hut approximations [53], allowing it to be applied on large real-world datasets. Fig. 8.4 shows the 2-D t-SNE representations of the $(1, 2)$-gram TF-IDF features extracted from the training data across the datasets. The scarcity of individual data points illustrating Flooding and Spoofing messages in the plots is because of their overlapping nature, which makes them appear as a few distinct data points.

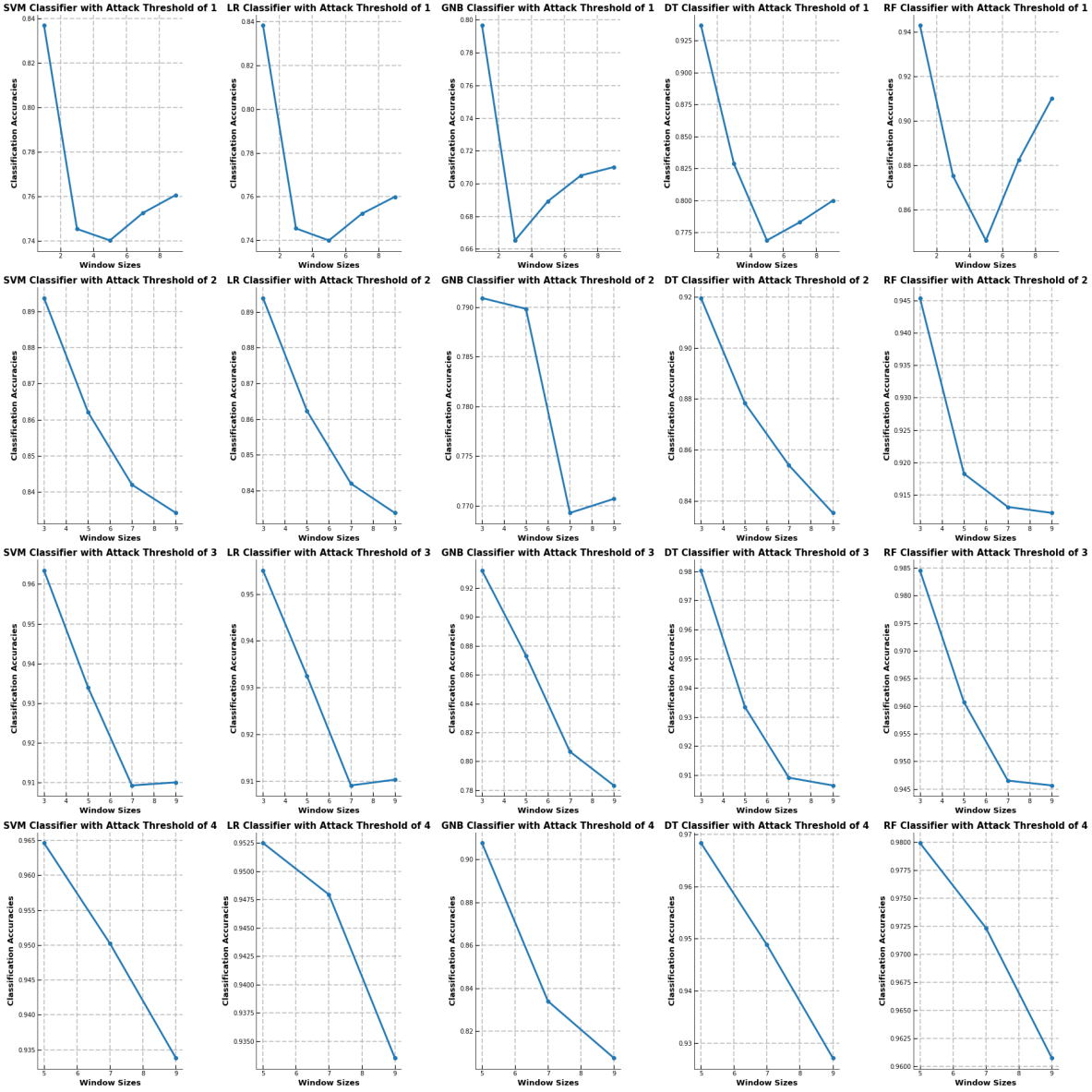


Figure 8.3: 2D plots of Classification Accuracies v/s Window Size for 1-gram TF-IDF features for varying attack packet thresholds generated across five models: SVM, LR, GNB, DT and RF. The plots indicate that across all models, window size of 1 has the best accuracy, which is hence chosen as the optimal input for CANLP.

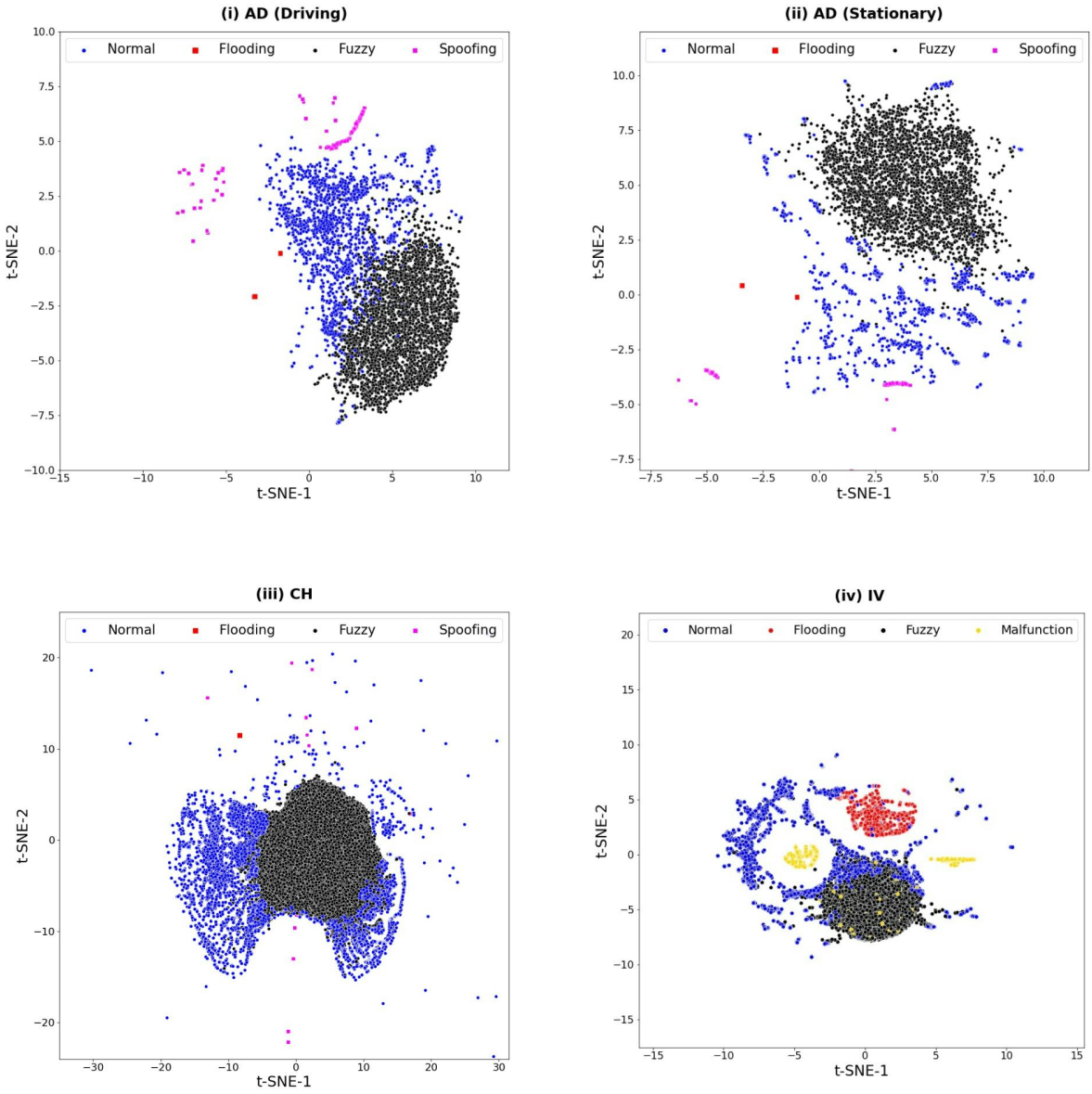


Figure 8.4: 2D t-SNE Representations of (1,2) gram character level TF-IDF features for (i) AD (Driving), (ii) AD (Stationary), (iii) CH and (iv) SA datasets. The visualization in each plot indicates a clear separability among features corresponding to the following classes: Normal message, Flooding message, Fuzzy message, Spoofing message, and Malfunction message. The limited number of data points representing Flooding and Spoofing messages in the plots is due to their overlapping nature, causing them to appear as few distinct data points. The distinct separation of data points of different classes in the plots validates the effectiveness of the proposed character-level (1,2)-gram TF-IDF features in achieving high F1-score.

The results obtained from the t-SNE analysis reveal that the extracted features using the TF-IDF technique demonstrate high level of separation between legitimate and adversarial frames (while represented in the feature space). This distinct separation validates the effectiveness of the proposed character-level (1,2)-gram TF-IDF features in achieving high classification F1-score.

# Chapter 9

# CONCLUSION AND FUTURE DIRECTIONS

In this thesis, we designed and developed CANLP, an intrusion detection system for Controller Area Networks (CAN). To the best of our knowledge, CANLP is the first framework to use natural language processing to extract features from individual CAN frames and identify specific transmitting ECUs that an attack has been mounted on. Using the term frequency-inverse document frequency encoding technique to identify complex features associated with CAN data, we trained machine learning models and a deep neural network to detect and classify fuzzing, spoofing, and masquerade attacks. Compressing the models using quantization techniques enables us to deploy it on resource-constrained hardware, indicating the feasibility of our system in real-world vehicles and improving accuracy-compute tradeoff. Our experiments evaluating CANLP on four publicly available vehicle datasets yielded significantly high F1-scores while maintaining low detection latency ($< 0.05\ ms$), indicating CANLP's effectiveness as an intrusion detection system for timing opaque attacks. We demonstrate that CANLP is suitable for different makes and models of vehicles based on evaluation on our CAN testbed and the ease of implementation with minimal modification to in-built software or hardware.

## *9.1 Future Direction*

Despite being a legacy protocol with various known security issues, CAN still represents the de-facto standard for communications within vehicles, ships, and industrial control systems despite its intrinsic vulnerability to attacks by malicious actors. In this thesis, we designed an IDS to identify attacks by training machine learning classifiers on bus traffic and its properties. Actions to take after detection are, on the other hand, less investigated, and prevention mechanisms usually include protocol modification (e.g., adding authentication). To progress the security of CAN, we

plan to work on a deterministic Intrusion Detection and Prevention system (IPS) based on physical ECU activations. In this work, we employ new classification of attacks based on the access level to the bus needed by an attacker, distinguishing between Frame Injection Attacks (FIA) (i.e., using frame-level access) and Single-Bit Attacks (SBA) (i.e., employing bit-level access). With this IPS, we can detect and prevent classical attacks on the CAN bus while detecting advanced attacks that have been investigated less in literature.

# BIBLIOGRAPHY

[1] Abien Fred Agarap. Deep learning using rectified linear units (relu), 2019.

[2] Kushagra Agrawal, Tejasvi Alladi, Ayush Agrawal, Vinay Chamola, and Abderrahim Benslimane. Novelads: A novel anomaly detection system for intra-vehicular networks. *IEEE Transactions on Intelligent Transportation Systems*, 23(11):22596–22606, 2022.

[3] Akiko Aizawa. An information-theoretic perspective of tf–idf measures. *Information Processing & Management*, 39(1):45–65, 2003.

[4] Muhammad Ali, Stavros Shiaeles, Gueltoum Bendiab, and Bogdan Ghita. Malgra: Machine learning and n-gram malware feature extraction and detection system. *Electronics*, 9(11), 2020.

[5] Javed Ashraf, Asim D. Bakhshi, Nour Moustafa, Hasnat Khurshid, Abdullah Javed, and Amin Beheshti. Novel deep learning-enabled lstm autoencoder architecture for discovering anomalous events from intelligent transportation systems. *IEEE Transactions on Intelligent Transportation Systems*, 22(7):4507–4518, 2021.

[6] Prafulla Bafna, Dhanya Pramod, and Anagha Vaidya. Document clustering: Tf-idf approach. In *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, pages 61–66. IEEE, 2016.

[7] Vita Santa Barletta, Danilo Caivano, Antonella Nannavecchia, and Michele Scalera. A kohonen som architecture for intrusion detection on in-vehicle communication networks. *Applied Sciences*, 10(15), 2020.

[8] Diksha Behl, Sahil Handa, and Anuja Arora. A bug mining tool to identify and analyze

security bugs using naive bayes and tf-idf. In *2014 International Conference on Reliability Optimization and Information Technology*, pages 294–299, 2014.

[9] Sebahattin Bektaş and Yasemin Şişman. The comparison of l1 and l2-norm minimization methods. *International Journal of the Physical Sciences*, 5(11):1721–1727, 2010.

[10] Gedare Bloom. Weepingcan: A stealthy can bus-off attack. In *Workshop on Automotive and Autonomous Vehicle Security*, 2021.

[11] Mehmet Bozdal, Mohammad Samie, Sohaib Aslam, and Ian Jennions. Evaluation of can bus security challenges. *Sensors*, 20(8), 2020.

[12] Mehmet Bozdal, Mohammad Samie, and Ian Jennions. A survey on can bus protocol: Attacks, challenges, and potential solutions. In *International Conference on Computing, Electronics & Communications Engineering*, pages 201–205. IEEE, 2018.

[13] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *20th USENIX security symposium (USENIX Security 11)*, 2011.

[14] Rung-Ching Chen and Su-Ping Chen. Intrusion detection using a hybrid support vector machine based on entropy and tf-idf. *International Journal of Innovative Computing, Information, and Control (IJICIC)*, 4(2):413–424, 2008.

[15] Kyong-Tak Cho and Kang G. Shin. Error handling of in-vehicle networks makes them vulnerable. In *Proc. ACM SIGSAC Conf. on Computer and Communications Security*, page 1044–1055, 2016.

[16] Kyong-Tak Cho and Kang G Shin. Fingerprinting electronic control units for vehicle intrusion detection. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 911–927, 2016.

[17] Copperhill Technologies. PiCAN 2 - CAN Bus Interface for Raspberry Pi. copperhilltech.com/pican2-duo-can-bus-board-for-raspberry-pi/, 2023. Accessed: Aug. 23, 2023.

[18] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*, pages 291–326. Chapman and Hall/CRC, 2022.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, 2016.

[20] Mee Lan Han, Byung Il Kwak, and Huy Kang Kim. Anomaly intrusion detection method for vehicular networks based on survival analysis. *Vehicular communications*, 14:52–63, 2018.

[21] Peter Harrington. *Machine Learning in Action*. Simon and Schuster, New York, NY, USA, 2012.

[22] Md Delwar Hossain, Hiroyuki Inoue, Hideya Ochiai, Doudou Fall, and Youki Kadobayashi. Long short-term memory-based intrusion detection system for in-vehicle controller area network bus. In *IEEE Annual Computers, Software, and Applications Conference*, pages 10–17, 2020.

[23] John D Howard and Thomas A Longstaff. A common language for computer security incidents. Technical report, Sandia National Lab. (SNL-NM), 1998.

[24] Influx Technology. Understanding CAN DBC. www.influxtechnology.com/post/understanding-can-dbc, 2021. Accessed: Aug. 23, 2023.

[25] International Standard Organization (ISO). CAN Standard ISO 11898-1:2015. www.iso.org/standard/63648.html, 2015. Accessed: Aug. 23, 2023.

[26] Hyunjae Kang, Byung Il Kwak, Young Hun Lee, Haneol Lee, Hwejae Lee, and Huy Kang Kim. Car hacking and defense competition on in-vehicle network. In *Workshop on Automotive and Autonomous Vehicle Security*, 2021.

[27] Nikhil Ketkar. *Introduction to Keras*, pages 97–111. Apress, Berkeley, CA, 2017.

[28] Shashwat Khandelwal and Shanker Shreejith. A lightweight multi-attack can intrusion detection system on hybrid fpgas. In *International Conference on Field-Programmable Logic and Applications*, 2022.

[29] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[30] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. Experimental security analysis of a modern automobile. In *2010 IEEE symposium on security and privacy*, pages 447–462. IEEE, 2010.

[31] Christopher Kruegel and Thomas Toth. Using decision trees to improve signature-based intrusion detection. In *International workshop on recent advances in intrusion detection*, pages 173–191. Springer, 2003.

[32] Weiyang Liu, Yandong Wen, Zhiding Yu, and Meng Yang. Large-margin softmax loss for convolutional neural networks. *arXiv preprint arXiv:1612.02295*, 2016.

[33] Xiao-Yang Liu, Yiming Fang, Liuqing Yang, Zechu Li, and Anwar Walid. High-performance tensor decompositions for compressing and accelerating deep neural networks. In Yipeng Liu, editor, *Tensors for Data Processing*, pages 293–340. Academic Press, 2022.

[34] Siti-Farhana Lokman, Abu Talib Othman, and Muhammad-Husaini Abu-Bakar. Intrusion detection system for automotive controller area network (can) bus system: a review. *EURASIP Journal on Wireless Communications and Networking*, 2019:1–17, 2019.

[35] Sk. Tanzir Mehedi, Adnan Anwar, Ziaur Rahman, and Kawsar Ahmed. Deep transfer learning based intrusion detection system for electric vehicular networks. *Sensors*, 21(14):4736, jul 2021.

[36] Charlie Miller and Chris Valasek. Adventures in automotive networks and control units. *Def Con*, 21(260-264):15–31, 2013.

[37] Omar Minawi, Jason Whelan, Abdulaziz Almehmadi, and Khalil El-Khatib. Machine learning-based intrusion detection system for controller area networks. In *Proc. ACM Symposium on Design and Analysis of Intelligent Vehicular Networks and Applications*, page 41–47, 2020.

[38] Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 29–48. Citeseer, 2003.

[39] Raspberry Pi. Raspberry Pi - Computing for Everybody. `www.raspberrypi.com/`, 2023. Accessed: Aug. 23, 2023.

[40] Dinuka Sahabandu, Sukarno Mertoguno, and Radha Poovendran. A natural language processing approach for instruction set architecture identification. *IEEE Trans. Information Forensics and Security*, 2023.

[41] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, Jan 2015.

[42] Eunbi Seo, Hyun Min Song, and Huy Kang Kim. Gids: Gan based intrusion detection system for in-vehicle network. In *Annual Conference on Privacy, Security and Trust (PST)*, pages 1–6. IEEE, 2018.

[43] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models, 2017.

[44] Wissam Siblini, Jordan Fréry, Liyun He-Guelton, Frédéric Oblé, and Yi-Qing Wang. Master your metrics with calibration. In *International Symposium on Intelligent Data Analysis*, pages 457–469. Springer, 2020.

[45] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[46] Standford. Useful Plots to Diagnose Deep Neural Network. `cs231n.github.io/neural-networks-3/`. Accessed: Aug. 30, 2023.

[47] Seba Susan and Amitesh Kumar. The balancing trick: Optimized sampling of imbalanced datasets—a brief survey of the recent state of the art. *Engineering Reports*, 3(4):e12298, 2021.

[48] Tensorflow. Tensorflow Lite. `https://www.tensorflow.org/lite`, 2023. Accessed: Aug. 30, 2023.

[49] Tensorflow. Tensorflow Lite C API. `www.tensorflow.org/lite/api_docs/c`, 2023. Accessed: Aug. 30, 2023.

[50] Trung Kien Tran and Hiroshi Sato. Nlp-based approaches for malware classification from api sequences. In *2017 21st Asia Pacific Symposium on Intelligent and Evolutionary Systems (IES)*, pages 101–105, 2017.

[51] Bruno Trstenjak, Sasa Mikac, and Dzenana Donko. Knn with tf-idf based framework for text categorization. *Procedia Engineering*, 69:1356–1364, 2014.

[52] Chin-Yang Tseng, Poornima Balasubramanyam, Calvin Ko, Rattapon Limprasittiporn, Jeff Rowe, and Karl Levitt. A specification-based intrusion detection system for aodv. In *Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks*, pages 125–134, 2003.

[53] Laurens Van Der Maaten. Barnes-hut-sne. *arXiv preprint arXiv:1301.3342*, 2013.

[54] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[55] Miki E Verma, Michael D Iannacone, Robert A Bridges, Samuel C Hollifield, Pablo Moriano, Bill Kay, and Frank L Combs. Addressing the lack of comparability & testing in can intrusion detection research: A comprehensive guide to can ids data & introduction of the road dataset. *arXiv preprint arXiv:2012.14600*, 2020.

[56] Li Yang, Abdallah Moubayed, and Abdallah Shami. MTH-IDS: A multitiered hybrid intrusion detection system for internet of vehicles. *IEEE Internet of Things Journal*, 9(1):616–632, jan 2022.

[57] Hongli Yuan, Yongchuan Tang, Wenjuan Sun, and Li Liu. A detection method for android application security based on tf-idf and machine learning. *PloS one*, 15(9):e0238694, 2020.

[58] Hanqi Zhang, Xi Xiao, Francesco Mercaldo, Shiguang Ni, Fabio Martinelli, and Arun Kumar Sangaiah. Classification of ransomware families with machine learning based onn-gram of opcodes. *Future Generation Computer Systems*, 90:211–221, 2019.

[59] Qingling Zhao, Mingqiang Chen, Zonghua Gu, Siyu Luan, Haibo Zeng, and Samarjit Chakrabory. Can bus intrusion detection based on auxiliary classifier gan and out-of-distribution detection. *ACM Trans. Embed. Comput. Syst.*, 21(4), sep 2022.

# Appendix A

# PSUEDOCODE

**Algorithm 1** Data Balancing Psuedocode

```
# Index database to construct clusters and build inverted file system

from imblearn.over_sampling import SMOTE

# Initialize SMOTE for balancing the dataset
smote = SMOTE(random_state=42)

# Apply SMOTE to the training data
X_train_balanced, y_train_balanced = smote.fit_resample(X_train, y_train)
```

**Algorithm 2** Char (1,2)-Gram TF-IDF Psuedocode

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Initialize TF-IDF Vectorizer for char level
vectorizer = TfidfVectorizer(analyzer='char', ngram_range=(1, 2))

# Fit and transform the training data
X_train_tfidf = vectorizer.fit_transform(X_train)

# Transform the test data
X_test_tfidf = vectorizer.transform(X_test)
```

**Algorithm 3** DNN Psuedocode with chosen Hyperparamters

```
from keras.models import Sequential
from keras.layers import Dense, Dropout

# Build the DNN model
model = Sequential([
    Dense(512, activation='relu', input_shape=(input_dim,)),
    Dropout(0.5),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['
    accuracy'])

# Fit the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test,
     y_test))
```

**Algorithm 4** ML Models SVM, LR, GNB, DT and RF Psuedocode

```
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

# Train and evaluate multiple ML models
SVC_model = LinearSVC(random_state=123)
SVC_model.fit(X_train, y_train)
SVC_accuracy = SVC_model.score(X_test, y_test)

LR_model = LogisticRegression(random_state=123)
LR_model.fit(X_train, y_train)
LR_accuracy = LR_model.score(X_test, y_test)

# Other models like GaussianNB, DecisionTreeClassifier, and
   RandomForestClassifier are similarly trained and evaluated
```

**Algorithm 5** Quantization Psuedocode for Dynamic range quantization

```
import tensorflow as tf

# Convert Keras model to TFLite model with dynamic range quantization
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_quant_model = converter.convert()

# Save the quantized model
with open('models/dynamic_quant_model.tflite', 'wb') as f:
   f.write(tflite_quant_model)
```