

DETECTING INTRUSIONS ON CAN BUSES INSIDE PASSENGER CARS WITH MACHINE LEARNING ALGORITHMS

Candidate:

Nicoleta-Valentina VOINEA

Scientific coordinators:

Prof. Habil. Dr. Eng. Bogdan GROZA

Dr. Eng. Camil JICHICI

Session: June 2024

ABSTRACT

Scopul acestui proiect este de a construi un sistem de detecție ale atacurilor de pe magistrala de CAN a autovehiculelor de pasageri. Acest sistem, conectat la magistrala de CAN a mașinii, va primi mesaje și le va clasifica folosind un algoritm de machine learning. Sistemul propus de detecție al intruziunilor poate fi folosit pentru filtrare în timp real al atacurilor, având timpi de detecție de ordinul milisecundelor.

Acest proiect are ca limbaj de programare principal Python, folosit pentru partea de machine learning. Pe lângă acesta s-a folosit și Communication Access Programming Language (CAPL) pentru simularea atacurilor. Pentru dezvoltarea proiectului a fost nevoie de urmatorul echipament: un vehicul de pasageri pentru generarea datelor, un Vector VN1630A pentru simularea și stocarea traficului, Raspberry Pi 4 Model B împreună cu un Pi 3 Click Shield și un controller MCP2518FD pentru construirea dispozitivului final.

ABSTRACT

The goal of the project is to build an intrusion detection system which uses machine learning in order to detect CAN bus attacks from passenger vehicles. This system, connected to the CAN bus of the car, will receive CAN frames and classify them using a chosen machine learning algorithm. The intrusion detection system can be used for real-time data filtering, since it provides detection times in terms of milliseconds.

This project was developed using Python as the main programming language for the machine learning part. Besides this, Communication Access Programming Language (CAPL) was also used for simulating attacks. In terms of equipment there were used: a passenger vehicle for generating data, a Vector VN1630A for simulating and recording CAN traffic, a Raspberry Pi 4 Model B together with a Pi 3 Click Shield and a MCP2518FD controller for building the final device.

CONTENTS

1	Introduction and Motivation	6
1.1	Reported attacks on Vehicles	6
1.2	Thesis objectives	7
1.3	Structure of thesis	8
2	State of the art	9
2.1	Intrusion Detection Systems based on machine learning	9
2.2	Intrusion Detection Systems based on other approaches	10
3	Background Knowledge and Tools	12
3.1	Controller Area Network	12
3.2	Machine learning	14
3.3	XL driver library	19
3.4	CANoe	20
3.5	RaspberryPi	22
4	Data Collection from a real World vehicle	24
4.1	Experimental Setup	24
4.2	Data extraction from OBD port	25
4.3	Analysis of CAN traffic	26
5	Proposed solution for securing CAN Bus	28
5.1	Adversary model	28
5.2	Overview of the solution	31
5.3	Implementation	32
6	Experimental Results	40
6.1	Metrics for the assessment of the proposed intrusion detection system	40
6.2	Detection Results	41
6.3	Computational results	45
7	Conclusions and future work	46
8	Bibliography	47

LIST OF FIGURES

1.1	Typical CAN bus attack scenario [1]	6
3.1	CAN bus interface [12]	12
3.2	Standard CAN frame structure [13]	13
3.3	Machine Learning Working principle [15]	14
3.4	KNN representation for a 2 dimensional dataset for K=3 and K=6 [18]	16
3.5	Multi-Layer Perceptron architecture [19]	16
3.6	Perceptron structure [20]	17
3.7	LSTM architecture [22]	18
3.8	Access to the Vector network interfaces via the XL-Driver-Library [23]	19
3.9	Trace example from current project	21
3.10	Raspberry Pi board together with a Pi 3 Click Shield and a MCP2518FD Click	23
4.1	Setup for collecting data	24
4.2	Some of the CAN frames collected	25
4.3	CAN Messages Distribution	26
4.4	Cycle time variation histograms	27
5.1	Simulation Setup for the adversary model	28
5.2	Intrusion Detection System setup	31
5.3	Raspberry Pi configuration menu	34
5.4	Versions used for packages	35
5.5	Example of 'candump' command	35
5.6	Intrusion Detection System while running	39
6.1	Confusion matrix [31]	40
6.2	Confusion matrices for fuzzing dataset	42
6.3	Confusion matrices for replay dataset	43
6.4	Confusion matrices for general dataset	44

LIST OF TABLES

3.1	Most used event handlers in CAPL [24]	21
4.1	Cycle times for all messages	27
6.1	Metrics for evaluation of machine learning classification algorithms	41
6.2	Metrics for fuzzing dataset	42
6.3	Metrics for replay dataset	43

6.4	Metrics for general dataset	44
6.5	Detection times for TensorFlow implementation	45
6.6	Detection times for TensorFlow Lite implementation	45

LIST OF SNIPPETS

5.1	Fuzzing attack implementation	29
5.2	Replay attack implementation	29
5.3	Timers working principle	30
5.4	Input preprocessing	36
5.5	Classification function using TensorFlow	36
5.6	Conversion to TensorFlow Lite model	37
5.7	Model loading using TensorFlow Lite	37
5.8	Classification function using TensorFlow Lite	38
5.9	Main function for the final program	38

1. INTRODUCTION AND MOTIVATION

1.1 REPORTED ATTACKS ON VEHICLES

During the last years, automotive sector has encountered a huge growth in the digitalization of their products. This leads to increased number of vulnerabilities due to cybersecurity threats, interconnectivity risks or software related bugs.

Many concerns are brought by network type of attacks, specifically CAN bus attacks. The CAN is the protocol that allows communication between the ECUs from a car, but unfortunately it does not provide security features. Attacks on CAN bus can manipulate or disrupt data that is being transmitted, being a potential risk source. During the last years, there have been found multiple examples of attacks on the CAN network, intruding through OBD-II port or telematic control unit, as presented in figure 1.1. More details about attacks infiltrated on cars' network will be discussed in the following paragraphs.

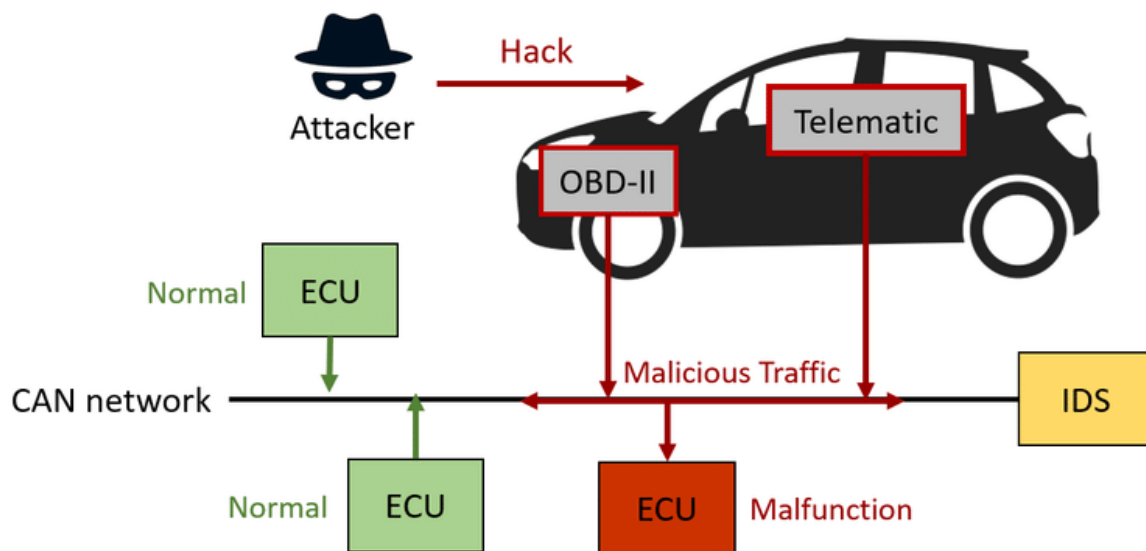


Figure 1.1: Typical CAN bus attack scenario [1]

A 2010 study [2] demonstrated the ability to insert malicious data in two passenger cars through the OBD-II port, succeeding to control multiple electronic control units (ECUs). They mainly focused on the weaknesses of the CAN bus protocol such as its broadcast nature, the weakness to denial-of-service frames (DoS), the absence of authenticator fields and poor access control mechanisms. Firstly, they succeeded to control the radio through the attacks they created: modifying the volume and controlling some of the car sounds. Then, they gained full control over the Instrument Panel Cluster and were able to display random messages, modify illumination and even control the speedometer value. Moreover, they discovered how to lock brakes and to control the engine speed and the A/C system.

In 2015, another study [3] showed how it was possible to gain remote access over multiple functionalities, through an after market telematic control unit (TCU) connected to the

OBD-II port. TCUs are part of modern world vehicles, representing the part that makes them stay connected with the outside world. Although by interconnecting the ECUs with the outside systems can bring new features to the vehicle, this connection can be the source of attacks. For this study, they also relied on the lack of authenticator fields of the CAN bus protocol, as no one can identify the source of the malicious frames. Their experiment involved multiple steps: taking control over the TCU in a remote way, learning how to make it send frames on the CAN bus and find the right messages to be sent in order to control some functionalities. They succeeded in following those steps, in the end being able to control the brakes and the windshield wipers via SMS.

Additionally, a 2016 research [4] has proven the ability to hack the cyber-physical systems (CPS). These CPS are responsible for the anti-lock braking systems (ABS), turning the steering wheel or controlling the accelerator. The paper focused on the same weakness previously mentioned: the broadcast nature and unverified node's authenticity. What they did to gain control over the CPS was to create an application that was cycling through the arbitration identifiers with random data payloads until there was a physical response. Once they found the right identifiers, they made a script containing the frames to be replayed through the OBD-II port, resulting in most of the CPS actions to be actuated.

Given the possibility of attacking the CAN bus and controlling critical functionalities from a vehicle, it is necessary to be implemented measures to prevent and diminish such effects. Development of intrusion detection systems would be a solution which would enhance the security of a vehicle's network. Furthermore, real-time detection and response to threats will significantly improve passenger's safety.

1.2 THESIS OBJECTIVES

The primary objective of this thesis is to enhance the security of passenger vehicles' bus systems by developing an application capable of filtering malicious data infiltrated on the CAN bus. The main steps that have to be performed are to analyze the CAN traffic from a passenger car, inject some attack frames into the data and develop a detection system capable of detecting and filtering these attacks in real time.

To achieve the first objective, there has to be performed a data collection on a real world passenger car, equipped with a CAN bus system. The recorded data will undergo a cleaning process that will remove any unwanted data, like error frames or irrelevant information. This process is followed by an comprehensive analysis on the dataset. This includes the analysis of the traffic and of each message sent, in terms of cycle time, length and frequency.

Secondly, to simulate attacks that will be injected on the CAN bus, an adversary model has to be created first. This model will encompass different types of attacks such as fuzzing and replay on the given dataset. By developing a complex and comprehensive adversary model, the simulation will emulate potential real-world security attacks.

Lastly, to finalize the development of the intrusion detection system, it is necessary to find a machine learning algorithm that is suitable for the dataset. This algorithm should later be deployed on a controller which will filter the data classified as attacks.

1.3 STRUCTURE OF THESIS

This thesis paper is divided in eight chapters which provide information for a comprehensive understanding of the subject: CAN bus intrusions detection system for passenger vehicles with the use of machine learning.

First chapter introduces the need of such a detection system by presenting some reported attacks on passenger vehicles. In addition, this chapter contains the objectives of this thesis.

The second chapter offers information about the latest features and technologies related to intrusion detection systems (IDS). This includes a dual-perspective analysis: one based on machine learning algorithms and the other based on alternative approaches and procedures.

The third chapter presents the theoretical knowledge needed for the development of the project. This includes basic knowledge about Controller Area Network (CAN) protocol and machine learning basics, together with some widely used classification algorithms. This chapter also presents the tools that were used, such as XL Driver Library, CANoe simulation and Raspberry Pi board.

The fourth chapter focuses on the creation of the dataset, by collecting data from a real-world vehicle. It presents the setup that was used and how the extraction was performed. The chapter concludes with an analysis of the collected CAN traffic.

The fifth chapter presents the adversary model designed to attack the CAN bus, together with an overview the proposed solution. For the solution, there is chosen a machine learning algorithm that performs the best on the given data.

The sixth chapter incorporates the result. It starts by introducing some used metrics in a theoretical way. Then, the detection results for the chosen algorithm are presented and interpreted. The chapter ends with some computational results, referring to the time of detection for the final detection system.

The seventh chapter summarizes the project and presents some conclusions. Moreover, there takes place a presentation of the future work.

The last chapter presents the bibliography and references used in this thesis. It includes articles, papers, user manuals and web pages.

2. STATE OF THE ART

The field of automotive cybersecurity has developed a lot in the past decade, since the modern vehicles incorporate more and more digital products. Either if we talk about systems that keep the car connected to the outside world, or simply telematic control units that bring new features, the car is susceptible to be the target of attacks. This section is going to present the state-of-the-art approaches of implementing an intrusion detection system to address this issue. The following sections will explore solutions from two perspectives: intrusion detection systems based on machine learning and intrusion detection systems based on other approaches.

2.1 INTRUSION DETECTION SYSTEMS BASED ON MACHINE LEARNING

With the advancements made in the domain of machine learning, through the last years, it started to be heavily integrated in automotive industry. Machine learning started to gain popularity in many automotive fields, including development of car security systems. As a result, it is published an increased number of research papers and articles regarding the use of machine learning in the development of CAN bus intrusion detection systems. Some State-Of-The-Art approaches of implementing such systems will be further presented in this section.

If talking about machine learning algorithms which can be used for the development of an intrusion detection system, we can enumerate Bayesian Networks, Long Short Term Memory networks, K Nearest Neighbours, Support Vector Machines and many more.

A 2021 article, called "Cybersecurity in automotive: An intrusion detection system in connected vehicles" [5] proposes an intrusion detection system based on a Bayesian network. To define such network, it is based on a probabilistic approach based on Bayes' theorem. It can be represented as a acyclic graph, where the edges represent relationships and to all vertices there are assigned some probabilities. This algorithm was preceded by a preprocessing phase of the data. In terms of attacks, there are defined three types: DoS attacks, fuzzing attacks and impersonation attacks. The experiment was performed using both an simulator that mimics a real vehicle and a dataset usually used in literature. In terms of detection, this network identified attacks with a percent of 98% accuracy on the simulated frames. On the other side, on the dataset used in literature they got an accuracy of 99%. Although there is a difference between the two datasets, the algorithm provided good results for the proposed attack scenarios.

Another article that is called "LSTM-based intrusion detection system for in-vehicle can bus communications" [6], which was published in 2020, presents a possible solution for implementing a detection system using LSTM networks. They tested the results on three datasets, containing DoS, fuzzing and spoofing attacks. These datasets contain data collected from a real vehicle, in raw format. After performing a binary classification using LSTM network ,having the configuration described in the paper, they got nearly perfect accuracies. For DoS and spoofing data set the model had an accuracy of 100%, while for fuzzing attacks it was

99.98%.

The study "Intrusion detection in vehicle controller area network (can) bus using machine learning: A comparative performance study" [7], published in 2023 presents a comparison of execution metrics for three machine learning algorithms: K Nearest Neighbours (KNN), Support Vector Machine (SVM) and Decision Tree (DT). These algorithms were tested on two different datasets that contained data collected from two different cars. In the datasets were inserted different types of intrusions, including fuzzing, impersonation and DoS for the first dataset, and fuzzing, flooding and malfunction for the second one. After they performed the classification they showed the results: KNN algorithm had an accuracy of 96% and respectively 97% for the second dataset, SVM had 97% and respectively 94% and DT had 99% accuracy for both datasets.

All the above presented solutions for intrusion detection systems are highly effective in terms of accuracies and scores and could be safely used to identify intrusions of the mentioned types. However, what is missing from the latest technologies, is a method able to detect unnoticeable types of malicious data, such as replay attacks. The model proposed in this paper also covers replay attacks, and although there is room for improvements, this solution can be further developed to obtain better performance.

2.2 INTRUSION DETECTION SYSTEMS BASED ON OTHER APPROACHES

Although machine learning gained its popularity in the implementation of intrusion detection systems, it is not the only way of identifying such intrusions. Numerous methods have been studied and developed, offering some advantages when using them. This section will provide information about the latest technologies used in performing attacks detection on the CAN bus.

The following paragraphs will provide insights about some alternative methods used in identifying malicious data, based on remote frames, Hidden Markov Model (HMM), entropy or voltage IDs.

One of the chosen articles, named "OTIDS: A novel intrusion detection system for in-vehicle network by using remote frame" [8] talks about how intrusions can be easily detected with the use of remote frames and the measurement of some time intervals. They tested this theory on a dataset containing DoS, fuzzing and impersonation attacks, when they succeeded to detect the intrusions. To understand the process, a remote frame is a type of frame that is broadcasted by the receivers when they need information from other nodes. This frame has the RTR bit recessive, compared to a regular data frame which has it dominant and comes as a response to the other. Their detection method broadcasts remote frames and waits for responses. They measured how normal data reacts to remote frames in terms of time and established some intervals of reference. When an intrusion is sent by an adversary, the attack-free data will be delayed, and so the intervals of receiving the wanted frames will exceed the intervals of reference. Moreover, for the final implementation of the intrusion system, they connected a Raspberry Pi 3 to the OBD-II port of a car and ran the IDS prototype.

An article published in 2023, "Multiple Observation HMM-Based CAN Bus Intrusion Detection System for In-Vehicle Network" [9], proposes an intrusion detection system based

on multi-observation Hidden Markov Model. This model's working principle stands on a computed probability of a frame to be present, based on its timestamp, ID, payload and cycle time. This method was tested on three different datasets containing fuzzing, DoS and Replay attacks. The results came out really well, succeeding to outrun some machine learning algorithms. For fuzzing and DoS datasets, the accuracy was around 99%, while for the replay dataset it was 95%. Considering that replay attacks are really hard to be identified, the proposed model had a satisfactory result, also compared to machine learning models (SVM, KNN) that were ran on the same dataset and achieved only 87% accuracy.

A research paper called "VoltageIDS: Low-Level Communication Characteristics for Automotive Intrusion Detection System" [10] introduced an intrusion detection system based on the measurement of voltages. The main idea behind this technique is that CAN messages are sent as an electrical signal, being inimitable. Based on this characteristic, the CAN traffic was analyzed and there were extracted some key events that describe a legitimate trace. The analysis of the signals can be performed using oscilloscopes, a cheap yet effective method for detection. As the authors present, this method performs well, even on buss-off attacks, which were the hardest to detect at the moment. However, a downside of this IDS observed is the equipment needed, which can be pretty big and inappropriate to use in a passenger vehicle.

To conclude this section, the above presented intrusion detection systems represent effective solutions for securing the CAN bus and can be compared in terms of accuracies with the machine learning models. Perhaps, the future researches will consider combining the machine learning with some of the approaches presented in this section to improve detection algorithms and thus obtaining a hybrid model.

3. BACKGROUND KNOWLEDGE AND TOOLS

3.1 CONTROLLER AREA NETWORK

The Controller Area Network, or the CAN bus is a protocol mostly used in automotive industry, designed to improve data exchanging process between devices. The protocol was developed by Robert Bosch GmbH in the early 1980s and it became an ISO standard in 1993 as ISO 11898. CAN bus was originally designed to replace complicated wires that connected multiple devices with a single two-wired bus, featuring signaling rates up to 1 Mbps, strong immunity to electrical interference, and the capability to self-diagnose and correct data errors.[11]

The CAN bus is a multimaster and multicast protocol, allowing any connected node to send messages when the bus is available, and enabling all nodes to access the data transmitted on the bus. If more nodes are attempting to send messages at the same time, or while the bus is busy the arbitration mechanism comes to solve the conflict. It works based on priorities, which are predefined as identifiers for each message. Consequently, the message having the identifier with the highest priority will have immediate access to the bus. Moreover, CAN protocol incorporate five levels of error detection, both at message and bit level, ensuring data integrity. These include cyclic redundancy checks, bit stuffing, acknowledgment mechanisms, error frames, and bus monitoring.

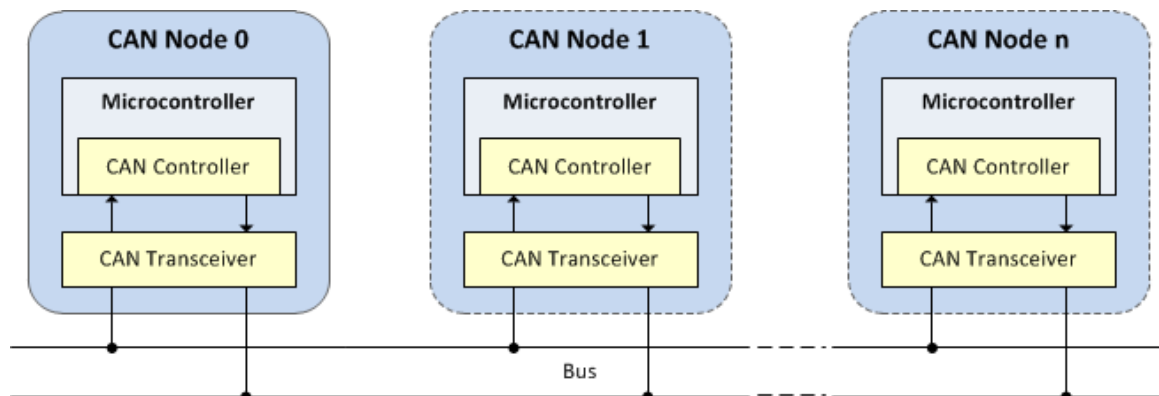


Figure 3.1: CAN bus interface [12]

When referring to the CAN bus interface, we consider multiple nodes connected to the bus lines, as illustrated in Figure 3.1. The two bus lines are known as CAN High Line (CANH) and CAN Low Line (CANL) and they are involved in the transmission and the reception of the data. Then, some essential elements for each node are the CAN controller and the CAN transceiver. The controller, as its name states is responsible for sending the messages, constructed according to the protocol standards. It also performs mechanisms such as error detection or arbitration. The CAN transceiver is responsible for transforming the signal from the controller into a differential signal and transmitting it over the bus. Moreover, it monitors the bus state and provides information back to the controller.

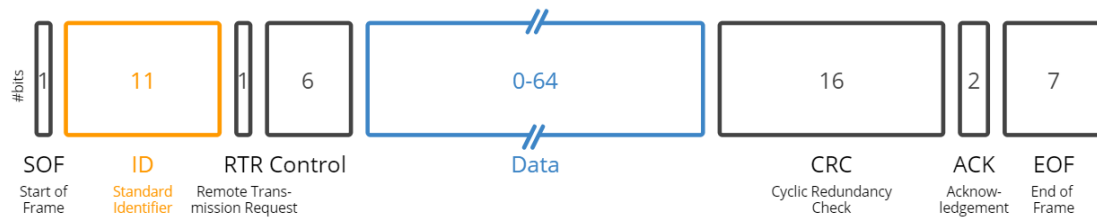


Figure 3.2: Standard CAN frame structure [13]

Another important aspect in understanding the CAN protocol lies in the structure of a CAN frame. It is composed of several fields, as illustrated in Figure 3.2, each serving a particular purpose. The standard CAN frame include the following components with their roles:

- Start of frame (1 bit) indicates the start of a message and helps the node in synchronization.
- Identifier (11 bits) has a preprogrammed value which is taken into consideration in case of arbitration.
- Remote Transmission Request (1 bit) indicates whether another node requests information and it is a recessive bit when a remote frame is sent.
- Control (6 bits) contains the data length code (DLC), which indicates the length in bytes of the data to be transmitted.
- Data (up to 8 bytes)
- Cyclic Redundancy Check (16 bits) contains the checksum and is used to ensure data integrity.
- Acknowledgement (2 bits) indicates whether the received data is correct.
- End Of Frame (7 bits) indicates the end of the frame.

In terms of types of messages, there can be found four categories: data frames, error frames, remote frames and overload frames. The data frame is the most common type and contains the fields previously described. An error frame is a particular type of message that consists of an error flag composed of six dominant bits and an error flag delimiter consisting of eight recessive bits. It is broadcasted over the network whenever a node detects an error. [14] Then, the remote frame is sent when a node needs information from another node. It looks like a regular data frame, having the RTR bit recessive and no data bytes. Overload frames, similar in format to error frames, are sent to introduce delays between messages when they are transmitted too rapidly for the nodes to process. Such a frame contains a flag composed of six dominant bits and a delimiter composed of eight recessive bits.

To elaborate on the types of bits previously mentioned, they are used to represent logical values. Dominant bits are represented by higher voltages on the CAN bus and they are interpreted as a logical 0. On the other hand, recessive bits correspond to lower voltages on the CAN bus and they are interpreted as a logical 1.

3.2 MACHINE LEARNING

Machine Learning is a branch of artificial intelligence, being a technology that enables machines to identify patterns and learn from data and past experiences. This technology is widely used today in many fields of application, including automotive. In terms of working principles, machine learning functions by using models that are trained on input datasets. These models are then used to make predictions, classify data, or to recognize patterns in new data.

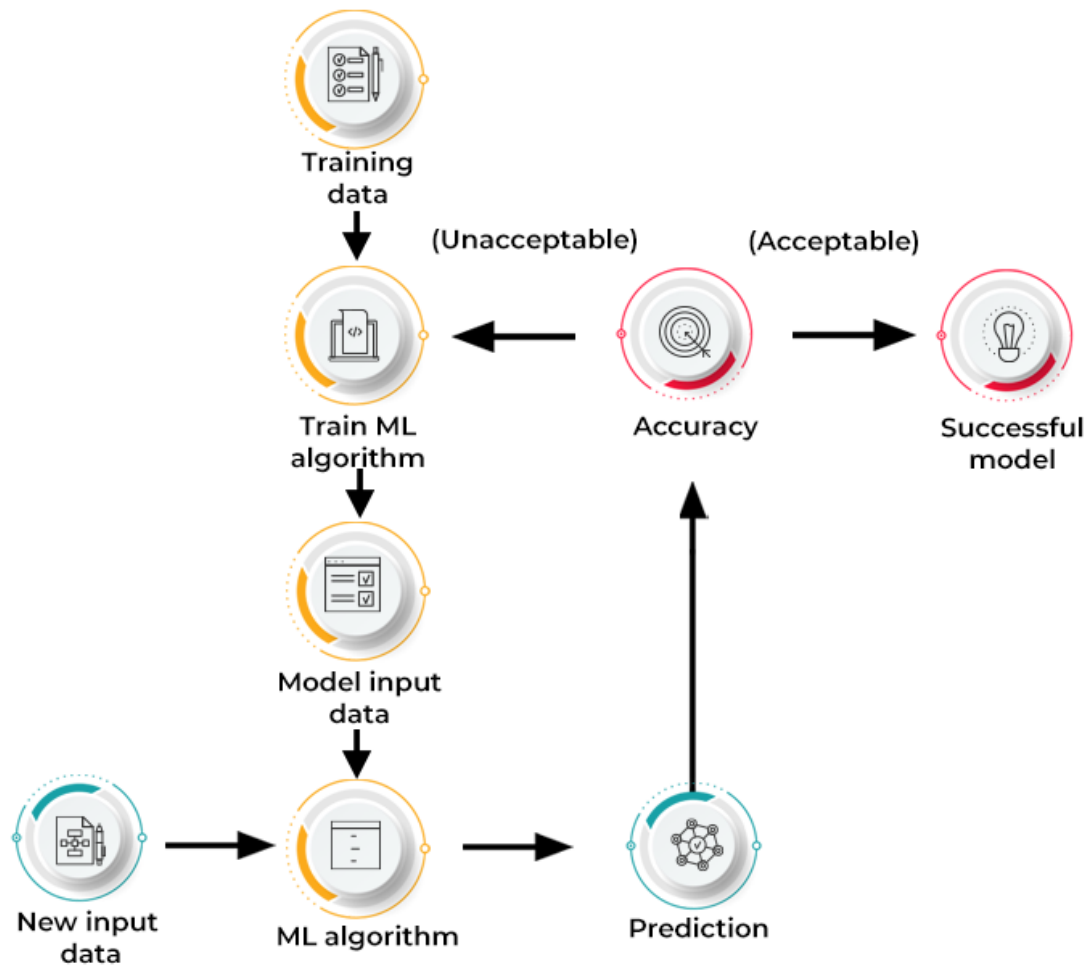


Figure 3.3: Machine Learning Working principle [15]

Figure 3.3 illustrates the general working principle of a machine learning algorithm, describing all its phases. The training data is used to train a certain algorithm, resulting in a trained model. To the trained model is provided new input data in order to make a prediction. Depending on whether the accuracy of the prediction is acceptable or not, the process either stops or starts over again for further improvement.

As the field of machine learning is continuously growing and improving, it has been divided into four types, each with its own purpose and mode of operation:

- a. Supervised Machine Learning is the type of machine learning that uses pre-labeled datasets and it is used to map inputs to their correct outputs. It can be divided into two main categories, depending on the task of the algorithm: classification and

regression. Classification models are usually used to sort the input data into a given number of categories. The most used classification algorithms include Decision Trees, K Nearest Neighbours, Support Vector Machines, Naive Bayes and Random Forest. Regression models predict outputs from a continuous domain, usually numerical values. They establish a numerical correlation between input and output data. The most used regression algorithms include Linear Regression, Ridge Regression or Lasso Regression.

- b. Unsupervised Machine Learning is a type of machine learning that doesn't need human intervention. It is performed on datasets with unlabelled data and its main goal is to discover new patterns and trends in the input data. The most popular algorithms used in unsupervised learning are K-Means Clustering Algorithm, Apriori Algorithm and Eclat Algorithm.
- c. Semi-Supervised Learning is a combination between supervised and unsupervised machine learning and it works with data that is partially labelled. It is a good approach for real-world data, where finding labeled entries might be rare, time consuming or costly. Two of the most used semi-supervised learning approaches are self-training and co-training.
- d. Reinforcement Learning is a type of learning where "an agent must learn a behaviour through trial-and-error interactions with a dynamic environment" [16]. There, the dataset is not labelled and the only way the machine can learn is through experiences. Using reward and penalty principle, the machine tries to learn patterns that minimize the punishment and maximize rewards.

The algorithms used in the development of this project are classification algorithms and include K Nearest Neighbours, Stochastic Gradient Method, Multi-Layer Perceptrons and Long Short Term Memory. These algorithms will be further described in the following paragraphs.

K Nearest Neighbours (KNN) is one of the most popular and simplest model of supervised learning, proving its efficiency in many cases. As a working principle, it stores the training dataset, but it does not perform any computation until the prediction time. This is why it is called a 'lazy' algorithm. In terms of complexity, "for the brute-force neighbor search of the KNN algorithm, we have a time complexity of $O(n \times m)$, where n is the number of training examples and m is the number of dimensions in the training set". [17]

When using KNN, it requires an user-defined K , which represent the number of nearest neighbours that should be considered when classifying an input. Then, it computes the distances between the input and all other points in the dataset using a distance function, most common being the Euclidian Distance. After that, the K nearest neighbours are identified based on the distance and an election based on 'majority voting' is made to classify the input.

Figure 3.4 illustrates a 2-dimensional representation of binary classification using K Nearest Neighbours. The goal is to classify the star symbol into one of the two classes: A or B. By choosing $K=3$, KNN algorithm classifies the star symbol as belonging to class B. On

the other hand, by choosing $K=6$, KNN algorithm classifies the same symbol as belonging to class A.

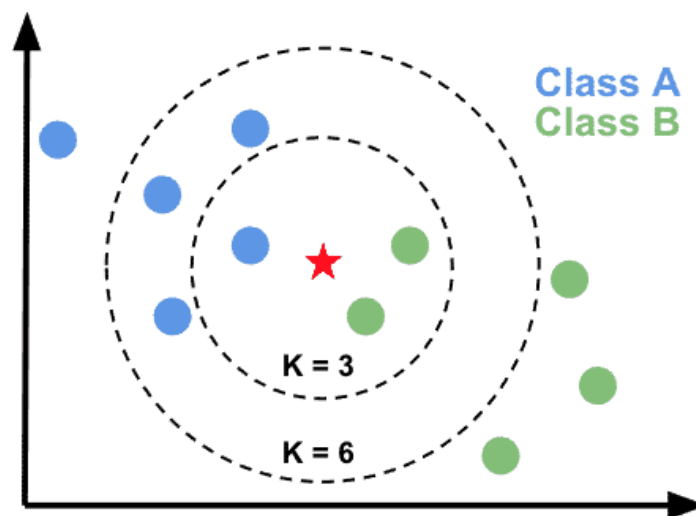


Figure 3.4: KNN representation for a 2 dimensional dataset for $K=3$ and $K=6$ [18]

Multi-Layer Perceptron (MLP) is one of the most popular artificial neural network structure that is used in a supervised way for classification tasks. They are capable of learning complex patterns by generating a nonlinear function model. The architecture of an MLP consists of fully interconnected layers of neurons. An MLP is composed of one input layer, hidden layers and one output layer, as illustrated in Figure 3.5.

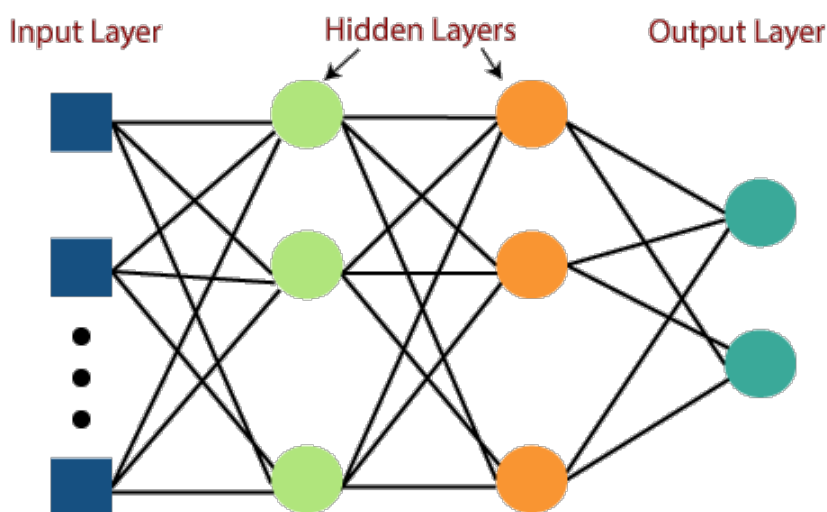


Figure 3.5: Multi-Layer Perceptron architecture [19]

The foundation of MLPs lies in perceptrons. A perceptron is a model invented by Frank Rosenblatt in 1957 and it has three main components, as presented in Figure 3.6:

- a. Inputs are numerical values that represent the features.

- b. Weights and bias are the hyperparameters. Weights represent the importance of each input feature. Bias allows the model to adjust the output independently of the input values.
- c. Activation function, also known as step function helps in deciding whether a neuron should activate or not. The most used activation functions are sigmoid, rectified linear unit (ReLU) and Hyperbolic Tangent (tanh).

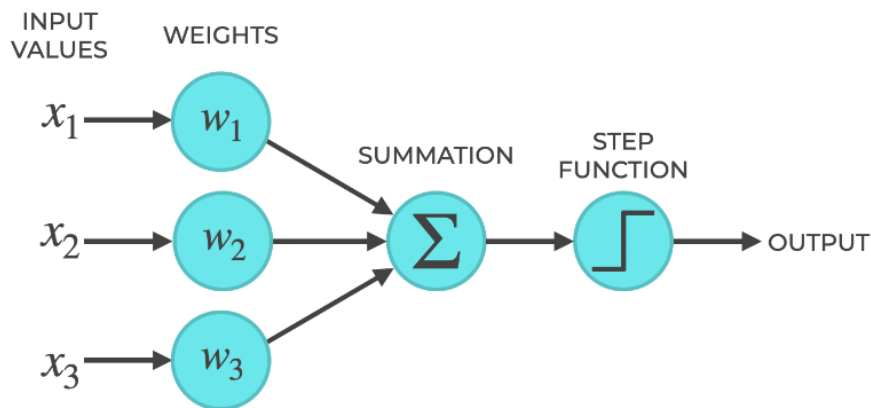


Figure 3.6: Perceptron structure [20]

The perceptron works as follows: the input values are multiplied by the weights and then added together. The activation function is applied over this sum, and the result is passed to the output.

The training of MLPs involves forward propagation and backpropagation. During forward propagation, input data is passed through the network, and predictions are made. During backpropagation, the error between the predicted and actual outputs is calculated, and the weights are adjusted to minimize this error.

Long Short Term Memory (LSTM) is a recurring neural network which reached popularity due to its capability to resolve the vanishing and exploding gradients problem. They were introduced by Hochreiter and Schmidhuber in 1997 to address the previously mentioned limitations of traditional RNNs.

The architecture of LSTM, presented in Figure 3.7, is based on memory blocks, which are some sub-networks recurrently connected. The task of a memory block is to keep its state through iterations and adjust the data flow using nonlinear gating units. [21] The architecture contains three gates, each having a particular role. The input gate controls the percentage of the input node's value that should be added to the current memory cell internal state. The forget gate determines if the current value from the memory is useful and decides whether to keep or delete it. The output gate decides whether the memory cell from the current time step should be taken into consideration when computing the output. [22] At a higher level, the outputs of the LSTM block are connected recurrently to the inputs.

In the development of this project, there were used multiple machine learning libraries, which helped in optimizing and building a final model. The libraries that were used include:

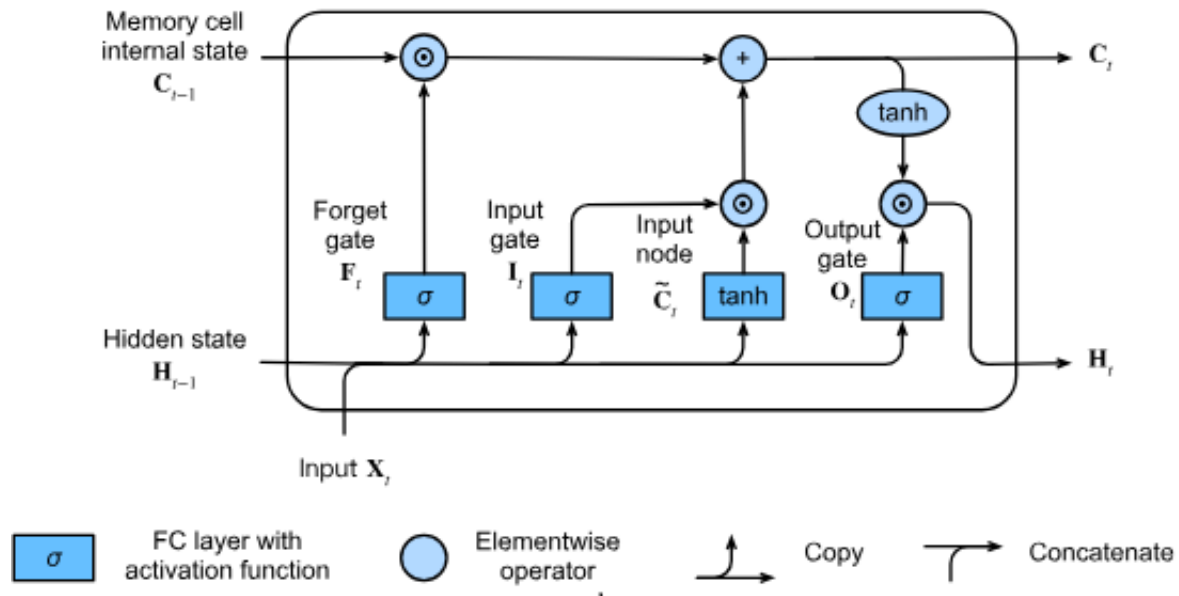


Figure 3.7: LSTM architecture [22]

- NumPy is a widely used library for data manipulation and numerical computations. Its specific popularity comes from its ability to handle N-dimensional arrays in a fast and versatile way.
- Pandas is another library used for data manipulation and analysis. It became popular due to the way of handling datasets and tabular data efficiently.
- Matplotlib is a library used for visual representation of data. It is widely used for generating plots and figures based on the input data. In addition, this library is highly customizable in terms of colors and shapes. Some of the most used plots generated using Matplotlib are line plots, pie charts, bar charts and histograms
- Scikit-learn or Sklearn is a machine learning library that uses Python programming language. It provides implementation for multiple ML algorithms in the area of regression, classification, clustering, model selection, dimensionality reduction and preprocessing.
- TensorFlow together with its API Keras is a powerful deep learning library that helps in building and training various models. This library supports wide range of machine learning and deep learning algorithms and enables efficient execution across different platforms, including CPUs, GPUs, and TPUs.
- TensorFlow Lite is a lightweight framework created to facilitate the deployment of machine learning models on embedded systems and other devices with limited resources. It also provides methods for conversion from a TensorFlow model into a compact and efficient format. The main advantage is that this framework is its optimizations, it improves detection speed and performance, especially for IoT and embedded devices.

For the development and execution of machine learning part of the code there was used Google Colaboratory. This environment created by Google is a free cloud-based platform

that allows users to write Python code. One big advantage offered by Google Colaboratory is that they give access to GPUs and TPUs, which are very useful in case of models and algorithms that require more computational power than one's computer has. Besides this, it is an easy platform to work with, without any setup to be required. Moreover, it is compatible with Google Drive, which can facilitate the development process by storing datasets, models and parameters on the cloud.

3.3 XL DRIVER LIBRARY

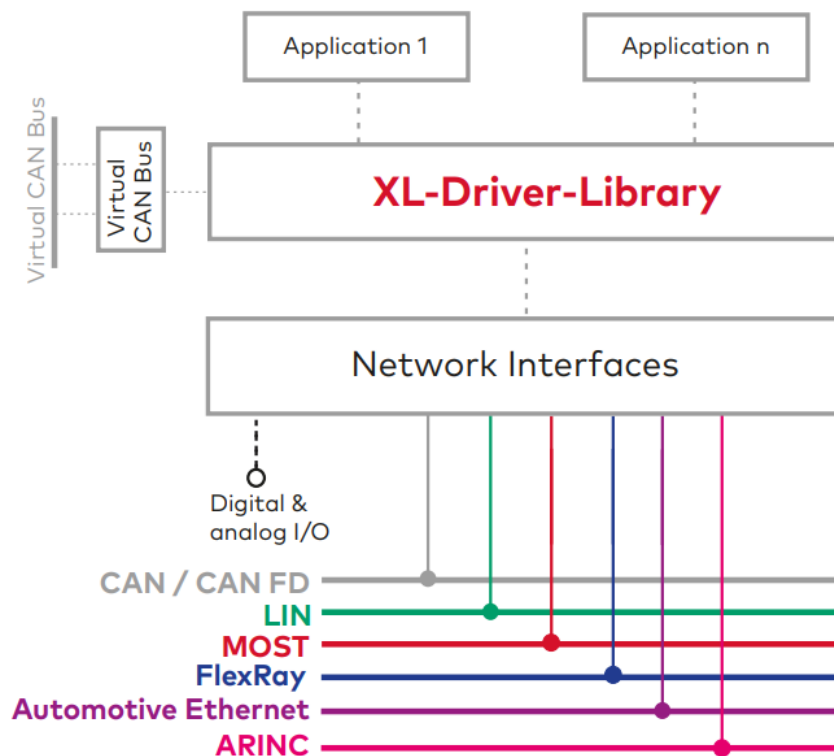


Figure 3.8: Access to the Vector network interfaces via the XL-Driver-Library [23]

The XL Driver Library is a versatile programming interface that allows you to develop custom applications using Vector's network interfaces. Figure 3.8 illustrates how one or more applications would access the interfaces offered by Vector through the XL Driver Library. The library offers both general and bus-specific methods, simplifying the operation of Vector's network interfaces. General methods are used for managing channels and ports, while bus-specific methods are used to configure network nodes and to send or receive frames. According to the interface factsheet published by Vector [23], there are available functions for the following bus systems:

- a. CAN
- b. LIN
- c. FlexRay

- d. Automotive Ethernet
- e. MOST
- f. ARINC
- g. I/O interface for acquiring analog and digital signals

Out of all the bus systems mentioned before, the one used in the development of this project is the CAN bus. The library offers methods for sending and receiving frames, setting the baud rate and monitoring transceiver errors. It also enables multiple simultaneous accesses to CAN interfaces.

3.4 CANOE

CANoe is a software tool from Vector Informatik GmbH, mainly used for development and testing. It is a widely used tool in automotive industry for simulating models which help in the development process. This tool provides a high variety of functionalities including database support for system description, network simulation, communication analysis, building customized user interfaces and programming.[24] As a result, the whole structure of the simulation can be customizable: from creating nodes, messages, signals, adding a database to creating special functionalities using CAPL scripting.

If we refer to the message as one of the smallest parts of a simulation, their transmission methods play an important role in the process. Some transmission methods provided by CANoe would be through interactive generators, replay blocks and automation sequences.

Interactive generators send frames based on their description on a pre-defined database. It can also send just raw values in the absence of a database. Using Interactive Generators, each field from a frame can be modified, by simply using CANoe interface.

Complementary to this, replay blocks are used when there is a desire of reproducing a sequence of frames that was previously recorded. The sequences are usually recorded using the Logging function provided by CANoe. In addition, the replay of the recorded frames can be done using multiple modes. The most used is the standard mode, which sends frames according to the recorded timestamps. Another replay mode sends the the data frame by frame, at user input. In terms of formatting, multiple formats are accepted by the replay block, including .asc and .blf.

The transmission method based on automation sequences is usually used when we need a reproducible set of operations. These operations can be automated in a visual manner, by macros or even by .NET snippets.

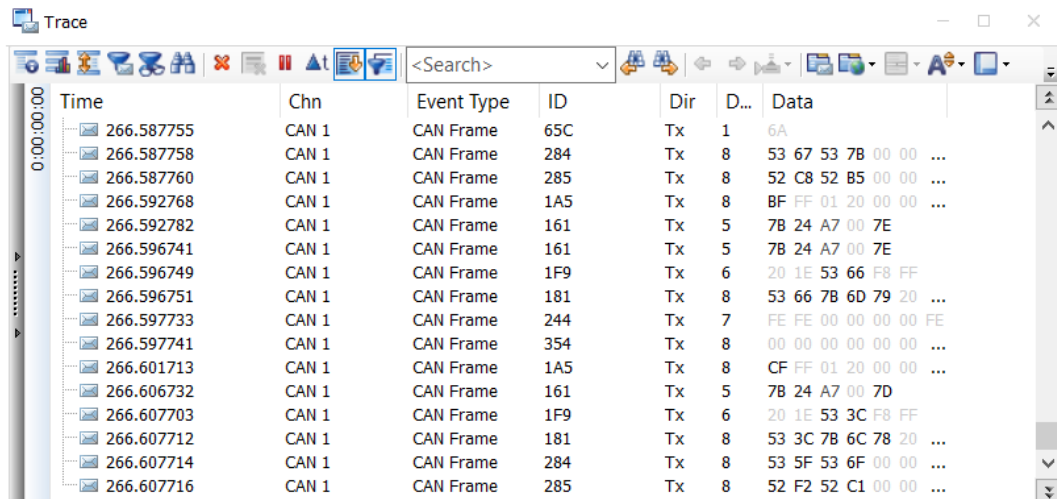
Another important feature incorporated in CANoe is the possibility of creating custom functionalities. This is done using Communication Access Programming Language (CAPL), which is a C-based, event driven and procedural programming language. CAPL Browser Window is a Vector tool integrated in CANoe and it is used to compile the CAPL code. A general structure for a CAPL program would contain an includes part, for integrating external implementations, variables section and event procedures. Being a C-based language,

it supports most of the data types, structures and statements provided by C language. Implementations in CAPL usually include data types like messages, built according to the supported protocols, and timers. When talking about events, they represent the base of a CAPL program. Table 3.1 presents a list of the most used event handlers, along with an explanation of their triggering conditions.

Measurement system events	
on start	Program start
on stopMeasurement	End of measurement
Message events	
on *	Receipt of any message
on message <msg>	Specific message received
Timer events	
on timer	Elapse of a timer
Other events	
on key	Press of a key

Table 3.1: Most used event handlers in CAPL [24]

In addition, CANoe provides analysis features, the most used one being the Trace window. It is automatically added to the measurement setup window and offers information about the messages that are being sent in the current session. Figure 3.9 presents an example of a trace window from the current project. Another analysis feature is represented by the graphics that can be created. There can be mapped values of the signals contained in a message or just variables. These plots will show any changes during the measurement, which helps in analyzing and debugging.



Time	Chn	Event Type	ID	Dir	D...	Data
266.587755	CAN 1	CAN Frame	65C	Tx	1	6A
266.587758	CAN 1	CAN Frame	284	Tx	8	53 67 53 7B 00 00 ...
266.587760	CAN 1	CAN Frame	285	Tx	8	52 C8 52 B5 00 00 ...
266.592768	CAN 1	CAN Frame	1A5	Tx	8	BF FF 01 20 00 00 ...
266.592782	CAN 1	CAN Frame	161	Tx	5	7B 24 A7 00 7E
266.596741	CAN 1	CAN Frame	161	Tx	5	7B 24 A7 00 7E
266.596749	CAN 1	CAN Frame	1F9	Tx	6	20 1E 53 66 F8 FF
266.596751	CAN 1	CAN Frame	181	Tx	8	53 66 7B 6D 79 20 ...
266.597733	CAN 1	CAN Frame	244	Tx	7	FE FE 00 00 00 00 FE
266.597741	CAN 1	CAN Frame	354	Tx	8	00 00 00 00 00 00 ...
266.601713	CAN 1	CAN Frame	1A5	Tx	8	CF FF 01 20 00 00 ...
266.606732	CAN 1	CAN Frame	161	Tx	5	7B 24 A7 00 7D
266.607703	CAN 1	CAN Frame	1F9	Tx	6	20 1E 53 3C F8 FF
266.607712	CAN 1	CAN Frame	181	Tx	8	53 3C 7B 6C 78 20 ...
266.607714	CAN 1	CAN Frame	284	Tx	8	53 5F 53 6F 00 00 ...
266.607716	CAN 1	CAN Frame	285	Tx	8	52 F2 52 C1 00 00 ...

Figure 3.9: Trace example from current project

3.5 RASPBERRYPI

RaspberryPi is one of the most used mini-computers nowadays. It was originally designed for educational purposes, but the board gained popularity due to its affordability and efficiency. Some of the key merits offered by this mini-computer include its affordability, the extensive peripheral support, support for multiple sensors, support for all kind of codes (C, C#, C++, Ruby, Python, Java, etc), the fast processor and the fact that it can be used as a portable computer. [25]

In the development of this project, there was used a RaspberryPi 4 Model B. As for the specifications, according to the official website [26], the micro-computer is characterized by:

- Processor: Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.8GHz
- RAM: 1GB, 2GB, 4GB or 8GB LPDDR4-3200 SDRAM (depending on model)
- Wi-Fi: 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless
- Bluetooth: Bluetooth 5.0, BLE
- Ethernet: Gigabit Ethernet
- GPIO: Raspberry Pi standard 40 pin
- USB: 2 USB 3.0 ports; 2 USB 2.0 ports
- HDMI: 2 × micro-HDMI® ports (up to 4kp60 supported)
- Other ports: 2-lane MIPI DSI display port, 2-lane MIPI CSI camera port, 4-pole stereo audio and composite video port
- Storage: Micro-SD card slot
- Power Supply : 5V DC via USB-C connector

In addition, there was used a Pi 3 Click Shield, an extension board from MikroElektronika. This click board enables the RaspberryPi to interface with a wide range of components, that would expand the functionalities originally provided. Since the Raspberry Pi doesn't have an analog pin on the expansion connector, the shield contains an onboard ADC (Analog-to-Digital Converter) which enables the measuring of the analog levels. [27]

Moreover, on top of the Pi 3 Click Shield there was connected a MCP2518FD Click add on board. This component enables CAN communication for the RaspberryPi, allowing it to function as a control node in a CAN network. In terms of protocols, MCP2518FD Click supports CAN FD, which is an advanced version of the traditional CAN protocol. This board features the MCP2518FD, an external CAN FD controller with an SPI interface, and a high-speed CAN transceiver, the ATA6563 [28].

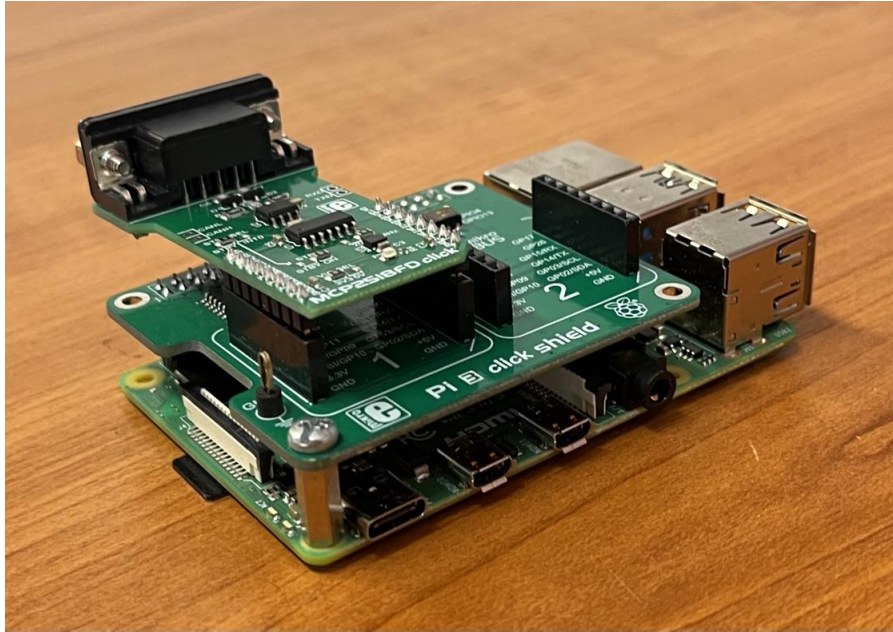


Figure 3.10: Raspberry Pi board together with a Pi 3 Click Shield and a MCP2518FD Click

The integration of the RaspberryPi with Pi 3 Click Shield and the MCP2518FD Click significantly enhances the RaspberryPi's capabilities, making it suitable for a variety of automotive applications, including an intrusion detection system. Figure 3.10 presents the above mentioned components, all connected together. The setup illustrated in the figure was used for the development of the project, along with some other elements presented in the next chapters.

4. DATA COLLECTION FROM A REAL WORLD VEHICLE

4.1 EXPERIMENTAL SETUP

For the purpose of this thesis, data collection, representing the CAN traffic, was performed on a real running vehicle. The dataset obtained will reflect what is happening in the day to day working vehicles. The CAN traffic was collected from a private vehicle, through the OBD port. Figure 4.1 presents the setup that was used. This includes:

- a. Laptop: it was used for running XL driver library program and recording the data. It also is the place for storing the dataset.
- b. Vector VN1630A: hardware interface used in automotive industry. It provides a fast and flexible access to CAN and , additionally, provides a recording functionality. In the setup it is used for capturing the CAN traffic.
- c. Cables: ensure connectivity and facilitate transfer of data from the vehicle to the storing location. There was used an OBD to Serial Port Cable for connection between the vehicle and Vector VN1630A and a USB to USB for connection between the Vector VN1630A and the laptop.



Figure 4.1: Setup for collecting data

4.2 DATA EXTRACTION FROM OBD PORT

When discussing the data extraction methodology, it is essential to consider some key aspects to ensure accurate results. Such details that need to be taken into consideration consist of vehicle information, including make, model and year, duration of the logging, driving conditions and events that occurred. Knowing these aspects one can debate on the relevance of the gathered data.

Regarding vehicle type, for the purpose of this project there was used a Dacia Duster model, fabricated in 2013, having a manual transmission. This vehicle was chosen because it supports direct access to the CAN bus via OBD port. Moreover, compared to some other vehicles, it does not have any security features implemented in order to restrict access to the CAN traffic.

Another important aspect concerning data collection would be the timing and the logging process. Besides the physical setup that was presented in the previous section, there was also a software running on the laptop in order to record the data. The program used in this scope can be found in the Vector XL Driver Library. Some frames that were collected, along with their format, can be found in Figure 4.2. There can be observed that through the data we can find info about the type of the message (Rx/Tx), timestamp, ID of the message, length of the message and lastly the message itself. Next, if we refer to the time of the collecting process, it sums up to 3490 seconds or 58 minutes. Recording the data for that period of time will ensure that multiple traffic events were caught, along with a higher variation in vehicle's speed.

```
RX_MSG c=2, t=1999333793792, id=0181 l=8, 35663F2440203F4E tid=00  
RX_MSG c=2, t=1999337897984, id=0244 l=7, FEFE00000000FE tid=00  
RX_MSG c=2, t=1999338184704, id=01A5 l=8, CFFF012000000080 tid=00  
RX_MSG c=2, t=1999342960640, id=0161 l=5, 4025A50025 tid=00  
RX MSG c=2, t=1999343157248, id=01F9 l=6, 201E3568F8FF tid=00
```

Figure 4.2: Some of the CAN frames collected

Last but not least, another significant detail in collecting useful and valid data, which ensures the dataset's robustness and comprehensiveness, would be the driving conditions. This means that during the recording time, the car and the driver should vary as many functions as possible to influence the CAN traffic. In this particular case, there were performed multiple transitions between idle state and engine running state, along with multiple activations of the parking sensors. In terms of speed, it varied between 0km/h and 120km/h. A variation was observed also regarding engine speed, that had a maximum value of 3500 RPM. As with every car, the braking system and the acceleration system play a crucial role in the movement of the vehicle, so in the collection process the two pedals were used with a variation in pressure and duration. In addition, some other features that the car provided were activated, used for a while and deactivated. These include windshield wiper, automatic car window opening, climate control, or interactions with the infotainment system.

4.3 ANALYSIS OF CAN TRAFFIC

The objective of this section is to analyze the collected CAN traffic data to gain insights into the network load, importance of the messages, their cycle times and to have an overview on the dataset.

After the collection process via OBD port, there followed a data cleaning process that consisted of removing the lines containing error frames. The final dataset ended up having 2140389 entries, so about 2 million records of CAN messages. For a further analysis, all different types of messages were considered, based on their ID. There came to be 12 different messages transmitted on the CAN bus of this specific car. Each one of them revealed a different distribution than the other, as illustrated in the pie chart of Figure 4.3.

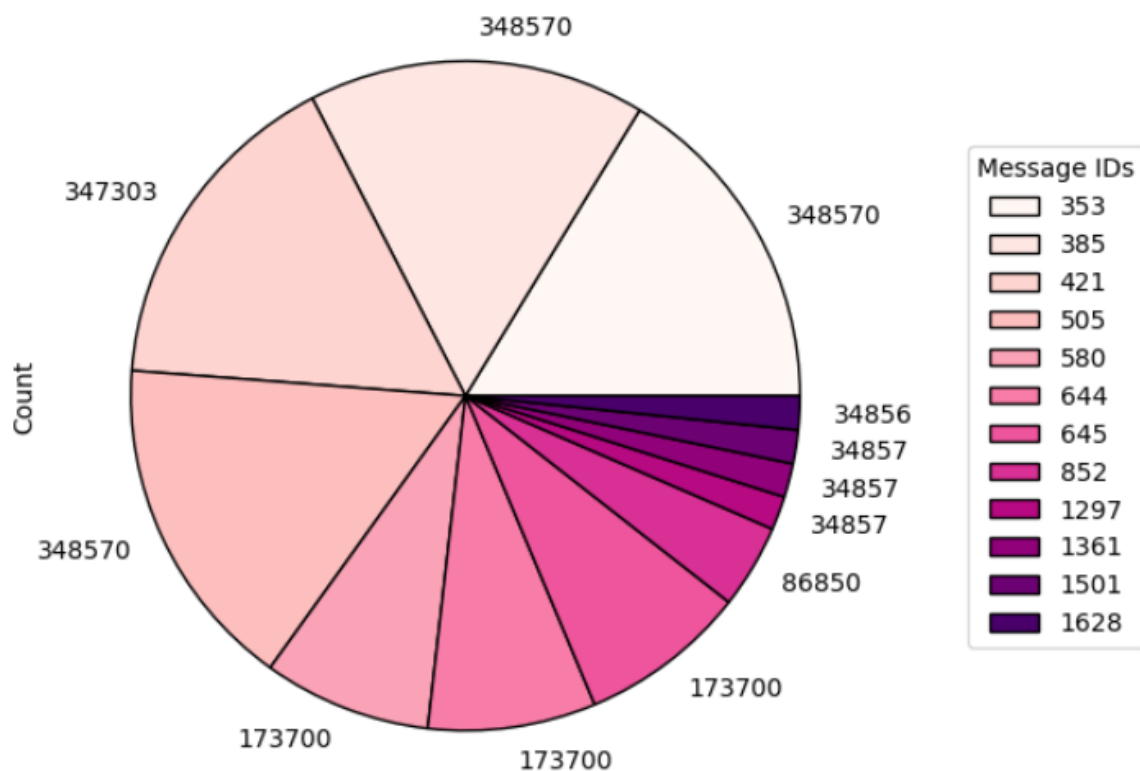


Figure 4.3: CAN Messages Distribution

In the following steps, there were examined the cycle times as they represent an important aspect in the analysis of the CAN traffic. They represent the intervals on which the messages are transmitted on the CAN bus and can be a powerful tool to describe the network.

While following the times when each message is sent, it can be observed that it is not constant, but varies around a constant value. This happens because frames are sent in rapid succession, which results in a burst on the bus. As a result of this, the frames will be delayed, especially the last one of them will be seen with a much higher delay than usual.[29] This statement is enforced by the histograms presented in the Figure 4.4, where the arriving time is plotted for messages having IDs 385,505,645 and 1361. In this plots, the corresponding value on the X axis of the the peak of the histograms would be the cycle time of the specified

CAN message. However, due to some frames not arriving on time, some delays of up to 4 milliseconds are observed.

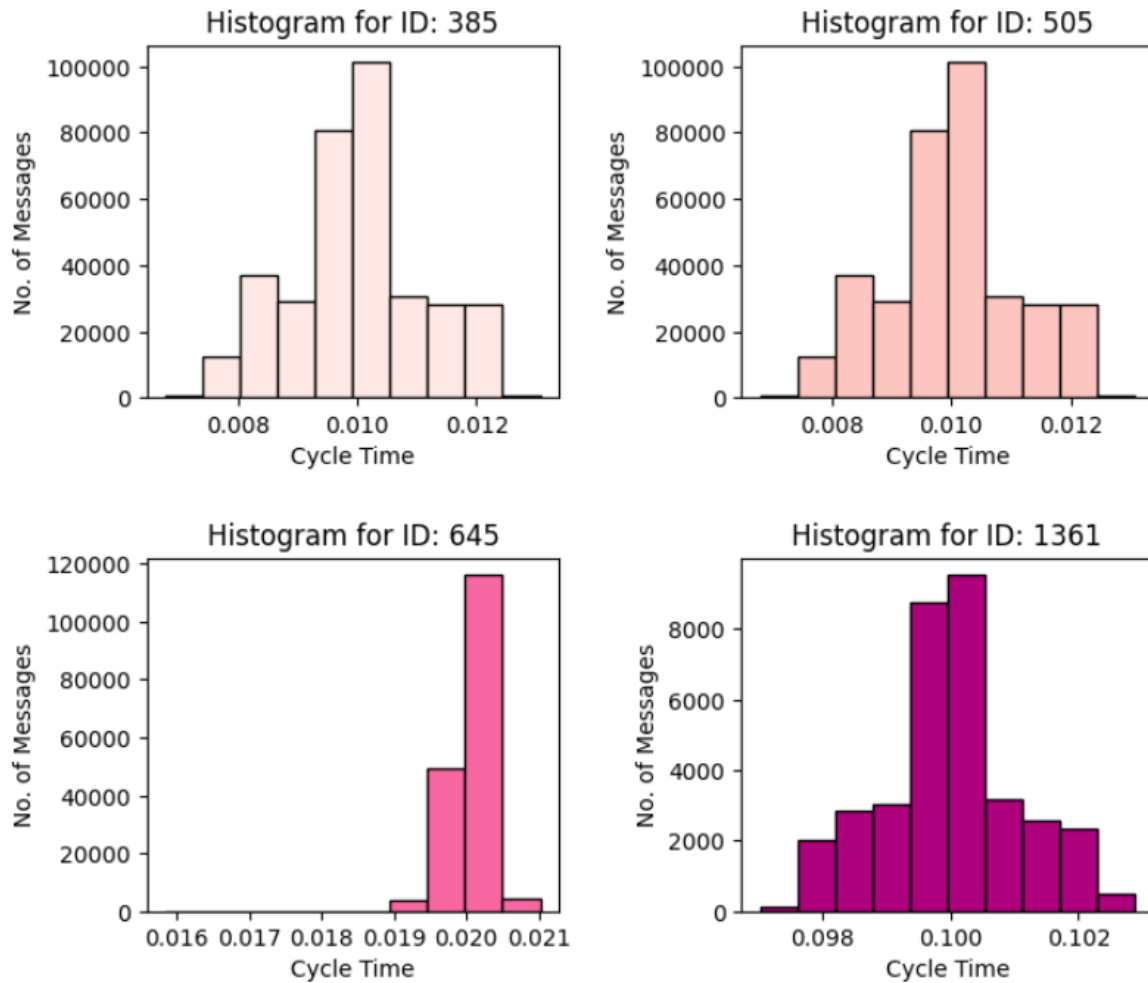


Figure 4.4: Cycle time variation histograms

As a result of the analysis, Table 4.1 was created to detail the cycle time for each message. It can be assumed that messages having ID 353, 385, 421 are the most important ones, carrying more sensitive information. This aspect can be assumed based on the cycle time of the specified messages, 10 milliseconds. On the other hand, messages with IDs 1297, 1361, 1501 and 1628 correspond to a lower traffic, their frames being sent less often.

Message ID	353	385	421	505	580	644	645	852	1297	1361	1501	1628
Cycle Time (ms)	10	10	10	10	20	20	20	40	100	100	100	100

Table 4.1: Cycle times for all messages

5. PROPOSED SOLUTION FOR SECURING CAN BUS

5.1 ADVERSARY MODEL

As the main focus of the project is to detect intrusions from the CAN bus, in order to test and validate the results, the dataset needs to contain some malicious data. In this scope, there was created an adversary model, that will infiltrate intrusions on our CAN bus. This model was implemented as a custom functionality in CANoe environment with the use of Communication Access Programming Language. For a comprehensive analysis of the solution on the proposed adversary model, there were created three types of attacks: fuzzing, replay and general, incorporating both of the previous mentioned ones.

In terms of structure, as illustrated in Figure 5.1, each adversary model is represented as a node on the CAN bus. The simulation setup also contains a replay block, used to repeat the frames sequence recorded during data collection phase. While active, an adversary node injects attacks real-time, taking into consideration the previous sent frames. The working principle of each type of attack will be further discussed in the next paragraphs.

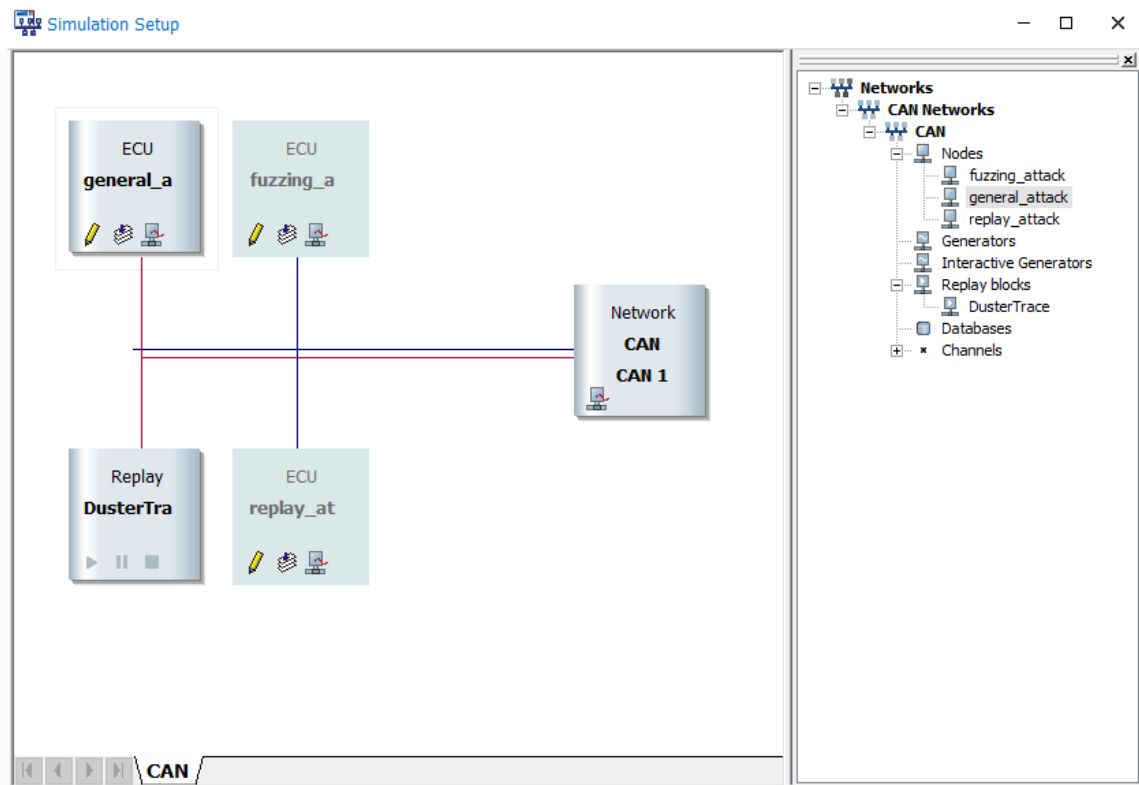


Figure 5.1: Simulation Setup for the adversary model

The key aspect highlighted by a fuzzing attack is the randomness. Despite the fact that it does not provide any information about the data and it does not target any particular functionality it can be quite effective, having some serious consequences such as steering wheel shaking or random gear shift changes [30]. In terms of structure, a fuzzing attack

consists of random generated data fields, assigned to a random message ID. The code used for the development of the fuzzing attack is presented in the Snippet 5.1. As brief summary, the legitimate frames that are sent on the CAN bus are targeted with a probability of MAX_PERCENT, set to 30%, to be used for attacks. Then, there is created a new message, having the same ID, DLC and direction, in order to resemble to the original message. Referring to the data field, each byte is populated one by one with a random generated value. The flag set using the attribute TYPE helps in later identifying and labeling the dataset.

```
1 if(random(100)<=MAX_PERCENT){ //attacks having probability=MAX_PERCENT
2     //populate fuzzing attack fields
3     m_attack.id=this.id;
4     m_attack.dir=tx;
5     m_attack.dlc=this.dlc;
6     for(i=0;i<=7;i++)
7         m_attack.byte(i)= random(255);
8     m_attack.TYPE=1; //set attack type flag
9 }
```

Snippet 5.1: Fuzzing attack implementation

Another easy way to inject malicious data on the CAN bus is through replay attacks. As their name states, they imply resending previously seen and valid frames after a period of time. The fact that the messages resemble the valid frames makes it hard for anyone to detect such attacks. Moreover, although they do not involve changing the payload, they can lead to serious safety issues. In terms of implementation, the replay attacks are the simplest ones to be created, as seen in Snippet 5.2. When a legit frame is targeted with the same probability discussed before, all of its fields are copied to the newly created replay attack (ID, DLC, direction, DATA, etc). The flag is also set to 1, and the intrusion frame is ready to be sent on the bus.

```
1 if(random(100)<=MAX_PERCENT){ //attacks having probability=MAX_PERCENT
2     //populate replay attack fields
3     m_replay=this;
4     m_replay.TYPE=1; //set attack type flag
5 }
```

Snippet 5.2: Replay attack implementation

When discussing about general attacks, they are a combination of fuzzing and replay intrusions. What makes them hard to be detected is the fact that one legitimate frame can be targeted by both fuzzing and replay at the same time, and such, two malicious frames are generated originating from just a valid one. In terms of implementation, it contains both of the previous presented snippets, merged. For this final adversary model it has been chosen the MAX_PERCENT to be 15%.

Equally important to the construction process of intrusions is their sending method. It is based on timers, provided by CAPL library. Snippet 5.3 illustrates the working principle of

those timers in the context of this project. Containing two time events and one message event, the proposed code succeeds provide unpredictable attack patterns. When timers `t_attack1` and `t_attack2` are triggered, they are used to output the attack frames `m_attack1`, respectively `m_attack2`. On the other hand, in the message event, after defining and populating the attack message as presented before, the timers are scheduled. Firstly, there is performed a check on the first timer. If it is active (set to send another intrusion) then the second timer is scheduled for the current attack. If the first timer is not used, then the attack frame is assigned to it. When deciding on the period to schedule the timers, they are different depending on the type of attack. For the fuzzing attacks, the timer is set after $\text{random}(\text{cycle_time}) \times 1000$, meaning a random time interval before a frame with the same ID is sent. On the other hand, replay attacks are sent after up to two frames with the same ID are received: $\text{random}(2 \times \text{cycle_time} - 1) \times 1000$.

```
1 on timer t_attack1{
2     output(m_attack1);
3 }
4 on timer t_attack2{
5     output(m_attack2);
6 }
7 on message CAN.*{
8     // {...} define & populate attack message
9     if(isTimerActive(t_attack1)){
10         m_attack2=m_attack;
11         setTimer(t_attack2,0,random(/*time*/);
12     }
13     else{
14         m_attack1=m_attack;
15         setTimer(t_attack1,0,random(/*time*/);
16     }
```

Snippet 5.3: Timers working principle

In order to create the dataset, after the attacks were infiltrated through the legitimate frames, the data needs to be logged. This logging process is essential for the development of the project, since in this phase the data gets recorded and labeled. For this purpose, during the simulation, there were used some logging functions provided by CAPL library. Such functions include 'openFileWrite' and 'fileClose', responsible for handling the file that will contain the dataset. Moreover, the data is recorded using 'filePutString' function, which enables data logging, in string format, to the previous opened file.

Through the logged CAN data, we can observe the following fields of a frame: timestamp, message ID, length of the message (DLC), eight bytes of data, direction of the message and channel. Out of these only a few will be used further, since the channel and the direction are always the same.

5.2 OVERVIEW OF THE SOLUTION

To address the problem of real-time detection of intrusions on the CAN bus of passenger cars, there is proposed an intrusion detection system (IDS) based on machine learning algorithms. The machine learning part of the solution is mainly based on a Long Short Term Memory type of network, but during the development phase, some other algorithms' performance was tested: K Nearest Neighbours, Stochastic Gradient Descent and Multi-Layer Perceptrons. This part of the project was developed using Google Colaboratory environment due to its flexibility. Then, for the actual development of the IDS there was used a Raspberry Pi controller that served as the hardware platform to run the final algorithm. The Raspberry Pi has as a task the real-time classification of CAN frames, utilizing the algorithm identified during the earlier phases of development.

The final setup used in the development of this solution is illustrated in Figure 5.2. The left hand side laptop is running the CANoe Simulation containing the adversary node. On the right of the figure, there is a laptop connected to the Raspberry Pi over the network via VNC Viewer. There can also be observed a VN similar to the one that was previously used for data collection. It is needed to establish the connection between CANoe Simulation and Raspberry Pi. Finally, the green controller is the Raspberry Pi 4 Model B, which contains the intrusion detection code.

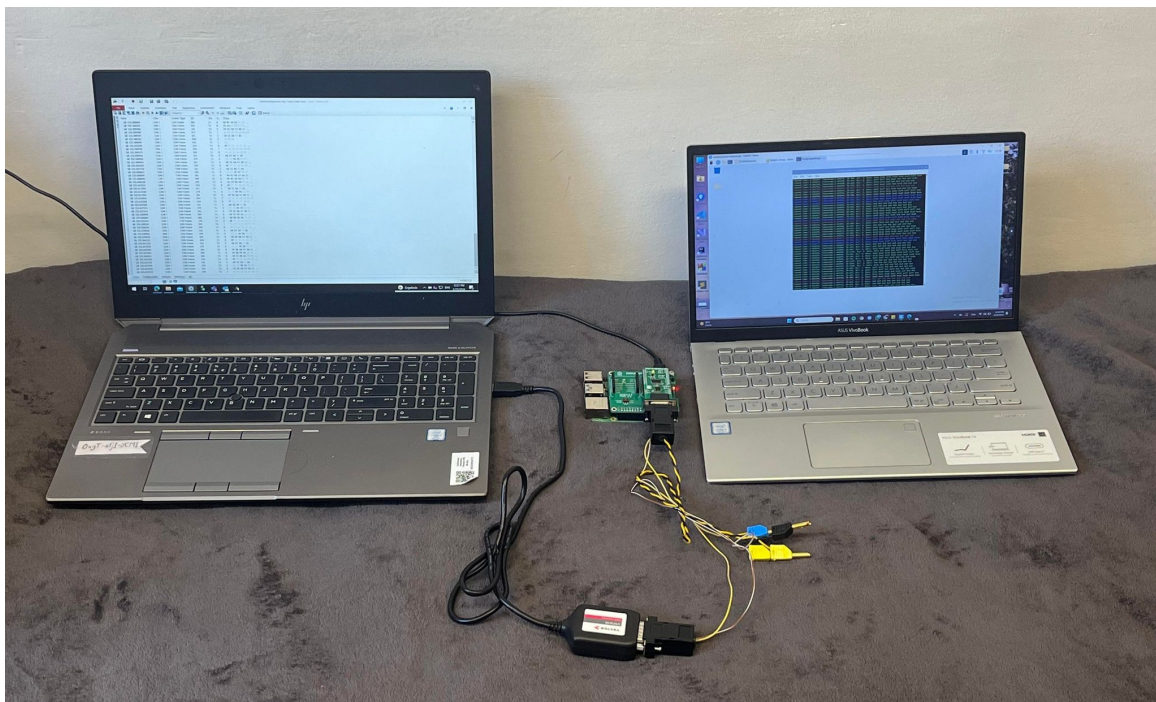


Figure 5.2: Intrusion Detection System setup

In addition to the presented setup, the intrusion detection system went through a comprehensive testing phase. Besides accuracy measurements, the system's performance was tested regarding detection times. After going through an improvement phase, the system's effectiveness demonstrated its potential applicability in real vehicles security systems.

5.3 IMPLEMENTATION

To further detail the machine learning part of the project, there will be discussed the steps that were performed. First of all, a data cleaning process was executed in order to remove irrelevant data, such as the error frames, and build the final dataset. After this step, the features used in classification included:

- Timestamp
- Message ID
- Length of the payload (DLC)
- Byte 1
- Byte 2
- Byte 3
- Byte 4
- Byte 5
- Byte 6
- Byte 7
- Byte 8

For the messages that contained less than eight bytes in the payload, there was performed a padding with zeros in the remaining bytes. In terms of representation, even though originally the bytes and the ID of the message were represented in hexadecimal format, for passing them to the model they were converted to decimal format. Regarding labels, they were loaded from a different file created at the time of attacks insertion. Then, the dataset was split into train and test data. The training dataset is used for the training of the machine learning algorithm, corresponding to the phase where patterns are constructed and connections are strengthened. On the other hand, the test dataset's purpose is only to evaluate the performance of the chosen model, by making predictions. The ratio used for the division was 80% for train and 20% for test.

The K Nearest Neighbours algorithm was the first to be evaluated, due to its simplicity and effectiveness. There were performed tests on all three datasets that we got: one containing only fuzzing attacks, one containing only replay attacks and the last one having combined attacks. Regarding K term, there were performed tests that were using multiple values out of which K=5 produced the best results. For the implementation, there was used the KNN algorithm provided by scikit-learn.

The next model that passed through tests was the Multi-Layer Perceptron one. In the same manner as the other algorithms, it went through multiple configurations before having the final version, it was tested on the same datasets and was imported from scikit-learn library.

It ended up having 5 layers of 10 neurons each, a Rectified Linear Unit (ReLU) activation function, an adaptive learning rate of 0.01 and an Adam solver for weight optimization. Since at first try, the model did not perform that well, after an analysis it has been figured out that the datasets are imbalanced in terms of attacks. To solve this problem there has been used a method called Synthetic Minority Oversampling Technique (SMOTE). It is specifically designed for these types of datasets and it works by generating some samples for the minority class, in this case attack class. The result is a balanced dataset, on which MLP model worked pretty good.

The final model, that was also chosen for the further development of the intrusion detection system was the Long Short Term Memory model, which was imported from TensorFlow library. Similarly as in the MLP case, due to the imbalance of attack values in the dataset, there came the need of balancing using SMOTE. It performed well even since the first tries of classification on the given datasets, but similarly as the others, it had a final version with the best working configuration. Referring to structure, the final model has the following configuration: an embedding layer, used for converting input data into a dense set of vectors, a LSTM layer containing 50 hidden units and a Dense layer that uses sigmoid activation function, responsible for the needed binary classification. The model is compiled with the use of Adam optimizer and binary_crossentropy loss. As it was chosen for the final IDS implementation, the parameters of the defined model needed to be saved and loaded on the Raspberry Pi. For this purpose, there were used functions provided by TensorFlow library. Firstly, after the training on the given data, the model was saved using the following syntax:

```
model.save('model.h5', save_format='h5').
```

The .h5 format was chosen because it provides a higher compatibility with other libraries and frameworks. Finally, to load the model on the Raspberry Pi, there was used the function:

```
model = load_model(model_path)
```

All the above mentioned algorithms, together with the resulted metrics on the given datasets are listed in the next chapter. Moreover, there will be the place to find a comprehensive analysis and comparison of the results.

After the most suitable model was chosen for the detection system, the next steps presume building the device capable of running the chosen classification algorithm. For the hardware setup, there was used a Raspberry Pi 4 model B, together with a Pi 3 Click Shield and a MCP2518FD Click attached. Before the code could be deployed on the controller, it should firstly go through a configuration phase. This step involves installing an operating system together with the needed libraries and resolving dependencies.

Firstly, an operating system was chosen to be installed on the Raspberry Pi. This process was done with the help of Raspberry Pi Imager, a tool provided by the producers of the controller. The operating system that was chosen was a 64-bit architecture Raspberry Pi OS, or as the producer describes it, "a port of Debian Bullseye with security updates and desktop environment". This operating system was written on the SD card that will later be introduced in the controller for the first boot.

In addition, since there was not owned a proper equipment (monitor, keyboard, mouse) to directly connect the Raspberry Pi for visualization and control, there come the decision of using an alternative method. The solution was to connect a laptop to the controller via VNC Viewer, a software that allows users to connect and control a certain computer over the network. To achieve this goal, there was created a connection to the controller via SSH, and by using the command 'sudo raspi-config' in the terminal, we were able to configure the board as needed. Figure 5.3 shows the configuration menu from the Raspberry Pi from where we enabled the VNC option. After this option would have been successfully enabled, there could be initiated a connection to the controller via VNC Viewer. With the use of this software application, we gained full access over the Raspberry Pi, remotely.

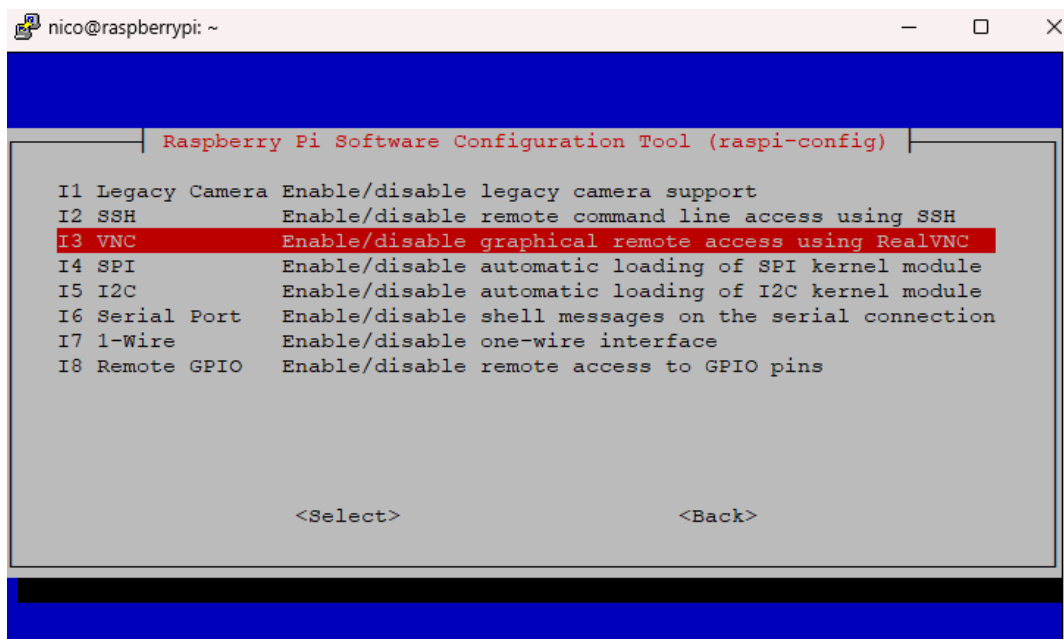


Figure 5.3: Raspberry Pi configuration menu

Additionally, as mentioned before, the configuration process must include the installation of the libraries and packages. Since the programming language that was used is Python, this is the first package to install. Then, the needed libraries include TensorFlow and TensorFlow Lite. Other needed packages and libraries include Keras and NumPy, but they are added by default when installing TensorFlow library. Another essential package in the development of this project is can-utils. All enumerated above were installed using 'pip' command, directly into the terminal, which has the following syntax:

```
pip install package_name
```

In terms of versions of the mentioned packages, they are shown in the Figure 5.4 as a screenshot of the terminal.

When starting the implementation, it was essential to establish a connection between CANoe Simulation and Raspberry Pi. This connection was obtained with the use of can-utils package. The two devices needed to have a common ground, so both of them were configured to have a 500k bitrate. On the controller, this was achieved by running the following terminal command:

```
sudo ip link set can0 up type can bitrate 500000
```

Once this setting is done on both devices we can start the communication, by running the CANoe simulation and running this command on the Raspberry Pi:

```
sudo ifconfig can0 up
```

```
nico@raspberrypi:~ $ python --version
Python 3.9.2
nico@raspberrypi:~ $ python -c 'import tensorflow; print(tensorflow.__version__)'
2.13.0
nico@raspberrypi:~ $ python -c 'import keras; print(keras.__version__)'
2.13.1
nico@raspberrypi:~ $ python -c 'import numpy; print(numpy.__version__)'
1.24.3
nico@raspberrypi:~ $ python -c 'import tfllite_runtime; print(tfllite_runtime.__version__)'
2.13.0
```

Figure 5.4: Versions used for packages

For visualizing the data, we can use the `candump` keyword, which prints on the screen all received frames. Figure 5.5 shows how this command is used, together with the outcome.

```
nico@raspberrypi:~ $ sudo ifconfig can0 up
nico@raspberrypi:~ $ candump -t z can0
(000.000000) can0 161 [5] 32 32 FF 00 10
(000.000030) can0 1F9 [6] 02 00 00 00 08 00
(000.000036) can0 181 [8] 00 00 32 10 32 C9 32 40
(000.000039) can0 161 [5] 32 32 FF 00 10
(000.000043) can0 1F9 [6] 02 00 00 00 08 00
(000.000047) can0 181 [8] 00 00 32 10 32 C9 32 40
(000.000050) can0 161 [5] 32 29 32 00 10
(000.000053) can0 1F9 [6] 00 1E 00 00 F8 19
(000.000057) can0 181 [8] 00 00 32 10 32 29 3B 4C
(000.000061) can0 161 [5] 32 29 32 00 10
(000.000064) can0 1F9 [6] 00 1E 00 00 F8 19
(000.000067) can0 181 [8] 00 00 32 10 32 29 3B 4C
(000.000070) can0 161 [5] 29 29 29 00 10
(000.000074) can0 1F9 [6] 00 1E 00 00 F8 00
(000.000078) can0 181 [8] 00 00 32 10 32 29 3B 4C
```

Figure 5.5: Example of 'candump' command

Once data is received, we want to redirect it to the file responsible for detection. To obtain this direct link we will be using pipes. Their working principle is that they connect the outcome of the first term to the input of the second. As an example, in this case there will be used the following command:

```
candump -t z can0 | python3 detect.py
```

Having all the connections done, the last step is to implement a program that reads frames from the input, line by line, and classifies them right away. Another important feature would be to compute the average time of detection, in order to analyze the duration.

In order to achieve the proposed goal, we will need to implement some functionalities that will do the work. First of all, it is necessary to load the model that we previously saved during training. This will be done using a predefined function in Keras library that is able to load a model by path.

Once we had the model loaded, one of the first functions that were implemented was one that preprocesses the data. As it can be seen in Snippet 5.4, every line has some unwanted symbols such as parenthesis or curly braces, or even unwanted fields. The function presented in Snippet 5.4, removes all the unnecessary data and converts the fields to decimal numerical values. Then, wraps everything in a NumPy array, which will be used by the detection algorithm.

```

1 def process_input(can_data):
2     can_data=can_data.replace('(', '')
3     can_data=can_data.replace(')', '')
4     can_data=can_data.replace('can0', '')
5     can_data=can_data.replace('[', '')
6     can_data=can_data.replace(']', '')
7     can_data = can_data.split()           #timestamp, id(hex), dlc,
8                                           #bytes(hex,no padding)
9     can_data[0]=float(can_data[0])       #timestamp (string->float)
10    can_data[1]=int(can_data[1],16)       #id (hex string ->int)
11    can_data[2]=int(can_data[2])         #dlc (string->int)
12    for i in range(can_data[2]):         #payload bytes
13        can_data[i+3]=int(can_data[i+3],16) # (hex string->int)
14    np_can_data = np.zeros(11, dtype=float)
15
16    np_can_data[:len(can_data)] = can_data
17    np_can_data=np.expand_dims(np_can_data, axis=0)
18
19    return np_can_data

```

Snippet 5.4: Input preprocessing

Having the input processed, the next step is to classify data. The implementation of this phase is presented in Snippet 5.5. To do this, there is used a predefined function for prediction. Moreover, this is the place where the timing is taken into consideration. There were used two variables that stored the times before and immediately after prediction. When finished, the function returns the duration of prediction together with the predicted value itself.

```

1 def classify_data(data,model):
2     start=time.time()
3     predicted= model.predict(data,verbose=0)
4     end=time.time()
5     return end-start,round(predicted[0][0], 0)

```

Snippet 5.5: Classification function using TensorFlow

After putting it together in the main function, there was observed a poor result in terms of detection times. The durations were higher than expected due to the way TensorFlow is designed. It was build for a various type of tasks, performing on powerful GPUs and TPUs. In order to solve this problem, the solution was implemented using TensorFlow Lite, a framework specifically designed for embedded devices. After using this library, detection

times decreased significantly and it has been decided to use it for the final implementation.

The first step performed in this direction was to convert the previously .h5 saved model to a model compatible with TensorFlow Lite framework. This goal was achieved using the code presented in Snippet 5.6, where our initial model is transformed into a .tflite model. Initially, the Keras model is loaded and there is initialised a TFLiteConverter object. Then, there are specified some configurations to ensure compatibility. The most important configuration is the one referring to supported operations. Since TF Lite does not support as many operations as the regular version, there is a need to specify a subset of operations that are mandatory for your model. Finally, the model is converted and saved at the specified path.

```
1 def convert_to_tflite(path_h5, path_tflite) :  
2     keras_model = load_model(path_h5)  
3     converter = tf.lite.TFLiteConverter.from_keras_model(keras_model)  
4     converter.optimizations = [tf.lite.Optimize.DEFAULT]  
5     converter.experimental_new_converter=True  
6     converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS,  
7         tf.lite.OpsSet.SELECT_TF_OPS]  
8     tflite_model = converter.convert()  
9  
10    with open(path_tflite, 'wb') as f:  
11        f.write(tflite_model)
```

Snippet 5.6: Conversion to TensorFlow Lite model

While talking about the way it works, TensorFlow Lite uses an interpreter interface to load and classify data. For the loading, part Snippet 5.7 describes how this process is performed. Firstly, the interpreter is initialized, loading the model from the specified path. Compared to the regular TensorFlow, the Lite version preplans tensor allocation in order to optimize inference by calling `allocate_tensors()`. Input and output details are some lists that provide information about the input/output data, including shape, and data types.

```
1 def load_tflite_model(path) :  
2     print('Model started loading')  
3     interpreter = tflite.Interpreter(path)  
4     interpreter.allocate_tensors()  
5     input_details=interpreter.get_input_details()  
6     output_details=interpreter.get_output_details()  
7     print('Model loaded successfully')  
8     return interpreter, input_details, output_details
```

Snippet 5.7: Model loading using TensorFlow Lite

The classification function works in a similar manner as the one used for TensorFlow. The only difference, also seen in Snippet 5.8, is that for TensorFlow Lite the prediction is computed with the use of the interpreter. By using the method `set_tensor(...)` we basically fill in the input data for the further classification. The `invoke()` method is responsible for computing the prediction. Finally, the `get_tensor(...)` extracts the prediction made

by the interpreter.

```
1 def classify_data(data, interpreter, input_details, output_details):  
2     start=time.time()  
3     interpreter.set_tensor(input_details[0]['index'], data)  
4     interpreter.invoke()  
5     predicted=interpreter.get_tensor(output_details[0]['index'])  
6     end=time.time()  
7     return end-start, round(predicted[0][0], 0)
```

Snippet 5.8: Classification function using TensorFlow Lite

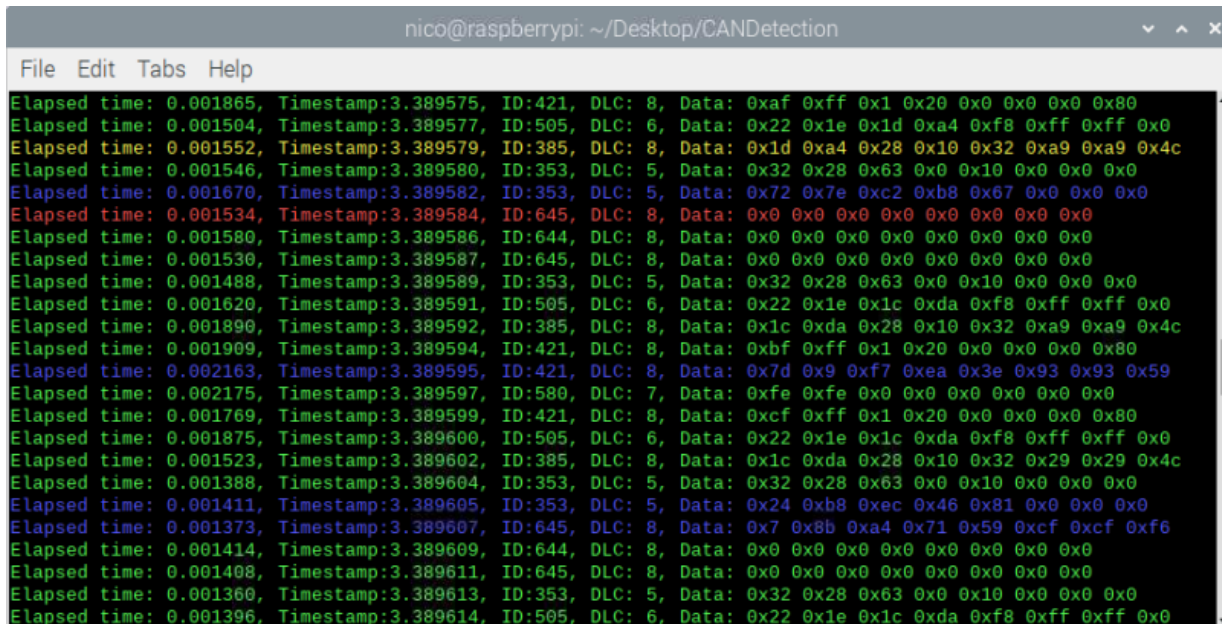
Having all the functions needed for the classification, they come together in a main function, presented in Snippet 5.9. At first, the model is loaded from the specified .tflite file. After that, in an infinite while loop, the program receives CAN frames, one by one, and stores them. They go through a preprocessing phase, and then straight to the classification part. For a better visualization of the outcome, there has been built a function (`print_message(...)`) for showing using colors the results of the classification.

```
1 def main():  
2     interpreter, input_details, output_details=load_tflite_model('model.  
3         tflite')  
4     while True:  
5         can_input = input('')  
6         can_data = read_input(can_input)  
7         can_data, attack= process_input(can_data)  
8         elapsed, prediction=classify_data(can_data, interpreter, input_details  
9             , output_details)  
10        attack=/*...*/           #extracting information whether the frame is an  
11        #attack or not, only for visualization purposes  
12        print_message(can_data, prediction, attack, elapsed)
```

Snippet 5.9: Main function for the final program

Figure 5.6 presents the outcome of the intrusion detection system implemented, where can be spotted some CAN frames that have been classified. In addition to the already known message fields, there is also specified an 'Elapsed' field, which specifies the duration of the classification phase. The color of the frames have the following explanation:

- Green: legitimate frames classified correctly
- Blue : attack frames classified correctly
- Red : attack frames that were not detected
- Yellow : legitimate frames that were classified as attacks



```
nico@raspberrypi: ~/Desktop/CANDetection
File Edit Tabs Help
Elapsed time: 0.001865, Timestamp:3.389575, ID:421, DLC: 8, Data: 0xaf 0xff 0x1 0x20 0x0 0x0 0x0 0x80
Elapsed time: 0.001504, Timestamp:3.389577, ID:505, DLC: 6, Data: 0x22 0x1e 0x1d 0xa4 0xf8 0xff 0xff 0x0
Elapsed time: 0.001552, Timestamp:3.389579, ID:385, DLC: 8, Data: 0x1d 0xa4 0x28 0x10 0x32 0xa9 0xa9 0x4c
Elapsed time: 0.001546, Timestamp:3.389580, ID:353, DLC: 5, Data: 0x32 0x28 0x63 0x0 0x10 0x0 0x0 0x0
Elapsed time: 0.001670, Timestamp:3.389582, ID:353, DLC: 5, Data: 0x72 0x7e 0xc2 0xb8 0x67 0x0 0x0 0x0
Elapsed time: 0.001534, Timestamp:3.389584, ID:645, DLC: 8, Data: 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0
Elapsed time: 0.001580, Timestamp:3.389586, ID:644, DLC: 8, Data: 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0
Elapsed time: 0.001530, Timestamp:3.389587, ID:645, DLC: 8, Data: 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0
Elapsed time: 0.001488, Timestamp:3.389589, ID:353, DLC: 5, Data: 0x32 0x28 0x63 0x0 0x10 0x0 0x0 0x0
Elapsed time: 0.001620, Timestamp:3.389591, ID:505, DLC: 6, Data: 0x22 0x1e 0x1c 0xda 0xf8 0xff 0xff 0x0
Elapsed time: 0.001890, Timestamp:3.389592, ID:385, DLC: 8, Data: 0x1c 0xda 0x28 0x10 0x32 0xa9 0xa9 0x4c
Elapsed time: 0.001909, Timestamp:3.389594, ID:421, DLC: 8, Data: 0xbf 0xff 0x1 0x20 0x0 0x0 0x0 0x80
Elapsed time: 0.002163, Timestamp:3.389595, ID:421, DLC: 8, Data: 0x7d 0x9 0xf7 0xea 0x3e 0x93 0x93 0x59
Elapsed time: 0.002175, Timestamp:3.389597, ID:580, DLC: 7, Data: 0xfe 0xfe 0x0 0x0 0x0 0x0 0x0 0x0
Elapsed time: 0.001769, Timestamp:3.389599, ID:421, DLC: 8, Data: 0xcf 0xff 0x1 0x20 0x0 0x0 0x0 0x80
Elapsed time: 0.001875, Timestamp:3.389600, ID:505, DLC: 6, Data: 0x22 0x1e 0x1c 0xda 0xf8 0xff 0xff 0x0
Elapsed time: 0.001523, Timestamp:3.389602, ID:385, DLC: 8, Data: 0x1c 0xda 0x28 0x10 0x32 0x29 0x29 0x4c
Elapsed time: 0.001388, Timestamp:3.389604, ID:353, DLC: 5, Data: 0x32 0x28 0x63 0x0 0x10 0x0 0x0 0x0
Elapsed time: 0.001411, Timestamp:3.389605, ID:353, DLC: 5, Data: 0x24 0xb8 0xec 0x46 0x81 0x0 0x0 0x0
Elapsed time: 0.001373, Timestamp:3.389607, ID:645, DLC: 8, Data: 0x7 0x8b 0xa4 0x71 0x59 0xcf 0xcf 0xf6
Elapsed time: 0.001414, Timestamp:3.389609, ID:644, DLC: 8, Data: 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0
Elapsed time: 0.001408, Timestamp:3.389611, ID:645, DLC: 8, Data: 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0
Elapsed time: 0.001360, Timestamp:3.389613, ID:353, DLC: 5, Data: 0x32 0x28 0x63 0x0 0x10 0x0 0x0 0x0
Elapsed time: 0.001396, Timestamp:3.389614, ID:505, DLC: 6, Data: 0x22 0x1e 0x1c 0xda 0xf8 0xff 0xff 0x0
```

Figure 5.6: Intrusion Detection System while running

This color based classification provides a visual representation of the system's accuracy and highlights areas for improvement. The high rate of green and blue frames indicates effective detection, while the presence of red and yellow frames suggests potential areas for improvement.

6. EXPERIMENTAL RESULTS

6.1 METRICS FOR THE ASSESSMENT OF THE PROPOSED INTRUSION DETECTION SYSTEM

For the assessment of the proposed intrusion detection system there was used some key metrics that helped in evaluating the machine learning algorithms. These metrics include the confusion matrix, the accuracy, the precision, the sensitivity and the F1 score.

Confusion matrix, illustrated in Figure 6.1, is a powerful metric when discussing about classification algorithms. This matrix presents the counts of the predicted and the actual values. True Positive value (TP) represents the number positive predicted values that were classified correctly. In the same way, True Negative value (TN) corresponds to the negative predicted values, classified correctly as negative. On the other hand, the False Positive value (FP) stands for the number of positive predicted values that were in fact negative. Similarly, the False Negative value (FN) represents the number of negative predicted values that were positive.

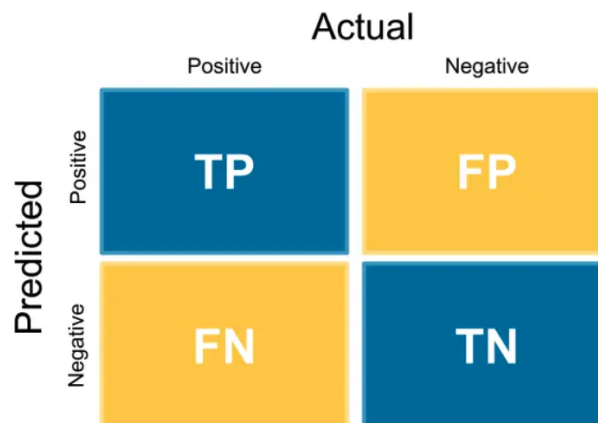


Figure 6.1: Confusion matrix [31]

Starting with the confusion matrix, there can be computed some other metrics. Table 6.1 presents the formulas based on which the metrics were computed.

The accuracy, or the overall success rate is the most popular metric used for evaluating the classification. Its popularity comes from its simplicity, corresponding to the percentage of correctly classified instances.

True Positive Rate (TPR) and True Negative Rate (TNR) fall under the category of marginal rates metrics and they are considered to be class-specific measures. TPR, or sensitivity represents the percentage of the positive values that were correctly classified. On the other hand, TNR, or Specificity, represents the percentage of the negative values that were correctly classified by the model. Positive Predictive Value (PPV) and Negative Predictive Value (NPV) are considered to be estimation-oriented measures. PPV, or the precision represents the percentage of the positive predicted values, that were correctly

classified. In the same way, NPV represents the percentage of the negative predicted values, that were correctly classified. When referring to their value, all four measures range from zero (0) to one (1).

The harmonic mean of PPV and TPR create the F1 score measure. It is considered to be class-specific and symmetric. "It can be interpreted as a measure of overlapping between the true and estimated classes, ranging from 0 (no overlap at all) to 1 (complete overlap)" according to [32].

Name	Abbreviation	Formula
Positive Values	P	$TP + FN$
Negative Values	N	$FP + TN$
Predicted Positive Values	PP	$TP + FP$
Predicted Negative Values	PN	$FN + TN$
Accuracy	ACC	$\frac{TP+TN}{P+N}$
True Positive Rate (Sensitivity)	TPR	$\frac{TP}{TP+FN}$
True Negative Rate (Specificity)	TNR	$\frac{TN}{TN+FP}$
Positive Predictive Value (Precision)	PPV	$\frac{TP}{TP+FP}$
Negative Predictive Value	NPV	$\frac{TN}{TN+FN}$
F1 score	$F1$	$\frac{2*PPV*TPR}{PPV+TPR}$

Table 6.1: Metrics for evaluation of machine learning classification algorithms

6.2 DETECTION RESULTS

During the development phase of the project, multiple algorithms were tested in order to find the best fit for our datasets. This section will provide information regarding the performance of three machine learning algorithms, tested on three types of datasets. The algorithms include K Nearest Neighbours, Multi-Layer Perceptrons and Long Short Term Memory networks. The configuration parameters of each algorithm is presented in the previous chapter. In terms of datasets, there have been used: a dataset containing only fuzzing attacks, a dataset containing only replay attacks and a dataset containing both types at the same time. Through the following paragraphs of explanation, the positive values would be represented by attacks and negative values as legitimate frames. While looking at the presented values, they were rounded to three decimal places.

Firstly, the algorithms were evaluated on the fuzzing dataset. Since it contains random data in the payload of the message frames, it should not be that hard for the algorithms to recognize the outliers through the dataset. Figure 6.2 illustrates the confusion matrices and Table 6.2 contains the detection metrics of the previous mentioned algorithms on this dataset, rounded as described above.

Looking at accuracies and F1 score, all of them worked really good, having really high performance, close to perfect detection. KNN algorithm had slightly lower precision if looking at TPR. Same metric shows that LSTM did not catch less than 0.01% of attacks, fact

also highlighted in the confusion matrix. MLP provided pretty accurate results, but did not outrun LSTM.

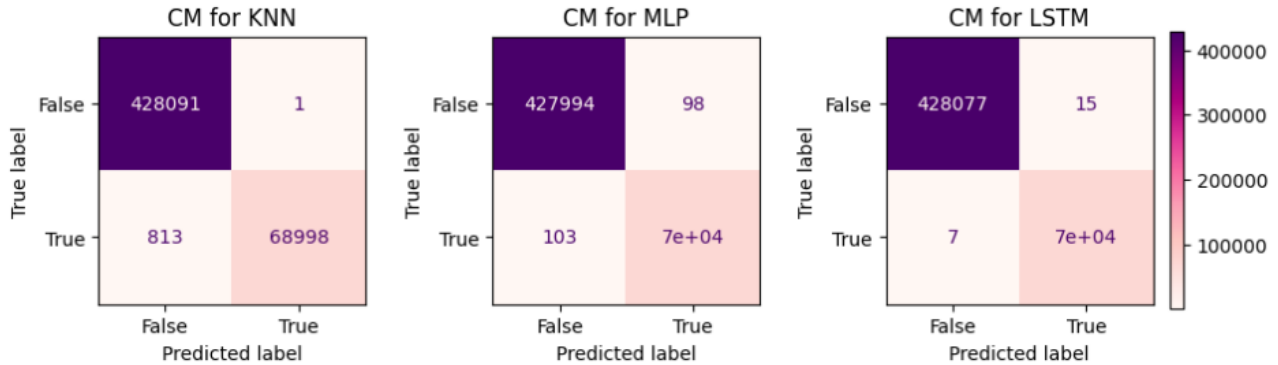


Figure 6.2: Confusion matrices for fuzzing dataset

Metric	KNN	MLP	LSTM
TN	428 091	427 994	428 077
FP	1	98	15
FN	813	103	7
TP	68 998	69 708	69 804
P	69 811	69 811	69 811
N	428 092	428 092	428 092
PP	68 999	69 806	69 819
PN	428 904	428 097	428 084
ACC	0.998	1	1
TPR	0.998	0.999	1
TNR	1	1	1
PPV	1	0.999	1
NPV	0.998	1	1
F1	0.999	0.999	1

Table 6.2: Metrics for fuzzing dataset

The next step in this analysis is to look over the performance of the algorithms over the replay dataset. Replay attacks are really hard to detect, especially if they are cleverly inserted. Due to the fact that they are identical to legitimate frames, most of the time it is nearly an impossible challenge to tell apart attacks from legitimate frames.

Table 6.3 together with Figure 6.3 detail the metrics obtained for the replay dataset. As expected, none of the algorithms succeeded have a high accuracy on this task, due to intrusions' structure. It can be observed that KNN algorithm identified correctly some intrusions, but in a very low percentage, close to 0. In terms of legitimate frames, KNN classified correctly almost all of them, according to TNR. Regarding MLP and LSTM, they identified more intrusions than KNN, but they also made mistakes by classifying legitimate frames as attacks. MLP model identified about 73% of attacks, but got wrongly a high percent

of legitimate frames, succeeding to have a final accuracy of approximately 39%. On the other side, LSTM had an accuracy of almost 84%, but it did not identify as many attack frames. In terms of legitimate frames, it made 94% of times a good classification. If looking at PPV metric, MLP and LSTM have close results. This means that for each of them, a bit more than 10% of the frames classified as attacks were actually attacks.

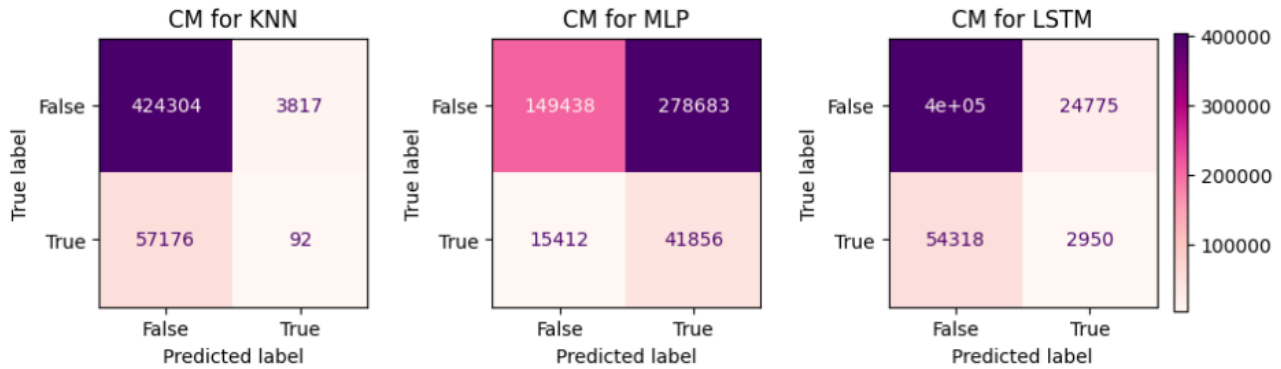


Figure 6.3: Confusion matrices for replay dataset

Metric	KNN	MLP	LSTM
TN	424 304	149 438	403 346
FP	3 817	278 683	24 775
FN	57 176	15 412	54 318
TP	92	41 856	2 950
P	57 268	57 268	57 268
N	428 121	428 121	428 121
PP	3 909	320 539	27 725
PN	481 480	164 850	457 664
ACC	0.874	0.394	0.837
TPR	0.002	0.731	0.052
TNR	0.991	0.349	0.942
PPV	0.024	0.131	0.106
NPV	0.881	0.907	0.881
F1	0.004	0.222	0.070

Table 6.3: Metrics for replay dataset

Lastly, the performance of those algorithms is evaluated on the general attack dataset. It contains both of the previous mentioned attacks, each with 15% probability. Figure 6.4 presents the confusion matrices for the performance of the algorithms, while Table 6.4 adds some other metrics to be analyzed.

The KNN model performs pretty good on the given dataset, although it occupies the last place in terms of accuracy and F1 score. According to TPR, the algorithm detects almost half of the attacks, but according to TNR it also makes some mistakes, classifying about 1% of legitimate frames as attacks.

MLP model occupies the second place referring to accuracy and F1 score, so it performs well. It also catches about half of the attacks, but regarding legitimate frames, it classifies correctly almost all of them.

Lastly, the LSTM model is the one that produced the best results so far, having slightly better results than MLP. Having an accuracy of almost 90%, it identifies successfully half of the malicious frames. According to PPV, when classifying a frame as an attack, the model is more than 99% of times right. While talking about legitimate frames, it also makes mistakes by classifying them as intrusions, but only for a small percent of data, smaller than 0.001% of all frames in the dataset. Considering the dataset also contains replay frames, which are very hard to detect, based on the accuracy and F1 score we decided to use it further as the main model for the intrusion detection system.

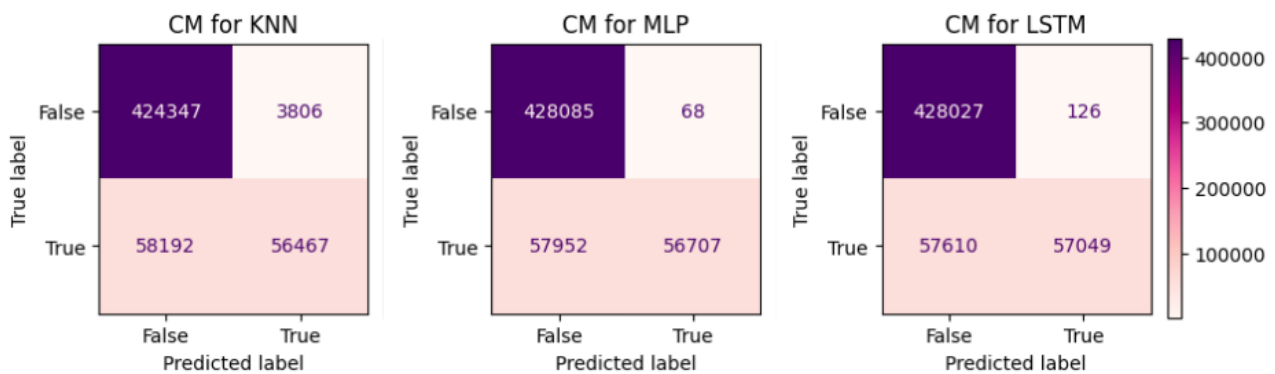


Figure 6.4: Confusion matrices for general dataset

Metric	KNN	MLP	LSTM
TN	424 347	428 085	428 027
FP	3 806	68	126
FN	58 192	57 952	57 610
TP	56 467	56 707	57 049
P	114 659	114 659	114 659
N	428 153	428 153	428 153
PP	60 273	56 775	57 175
PN	482 539	486 037	485 637
ACC	0.886	0.893	0.894
TPR	0.492	0.495	0.498
TNR	0.991	1	1
PPV	0.937	0.999	0.998
NPV	0.879	0.881	0.881
F1	0.645	0.662	0.664

Table 6.4: Metrics for general dataset

6.3 COMPUTATIONAL RESULTS

After moving further with the LSTM model, an essential factor to be taken into consideration is time. In order to have a real-time detection of malicious data, there is a need of an algorithm that classifies entries really fast.

To evaluate the classification time, there was implemented an algorithm which computes the average detection time from multiple frames. This process was performed for sets of 10, 100, 1000 and 10000 frames to analyse the behaviour over different sizes of sets of data.

Table 6.5 illustrates the detection time in milliseconds for the given sets of messages. As it can be observed, the elapsed time for classification is extremely high, making it impractical for use in a car's security system where rapid detection within milliseconds is crucial .

Number of frames	Detection time (ms/frame)
10	347.241
100	206.887
1 000	195.458
10 000	195.281

Table 6.5: Detection times for TensorFlow implementation

To address this issue, the LSTM TensorFlow model was converted to a TensorFlow Lite one. TensorFlow Lite is an optimized version of the regular framework designed specifically for embedded systems. After the conversion was performed, detection times were measured again, and recorded in Table 6.6. The result showed significantly lower values. This conversion improved the efficiency of this algorithm and made the model feasible to be implemented in a real car.

Number of frames	Detection time (ms/frame)
10	1.491
100	1.502
1 000	1.523
10 000	1.475

Table 6.6: Detection times for TensorFlow Lite implementation

As observed in Table 6.6, for the optimized model the average detection time has small variations around 1.5 milliseconds. It does not modify regardless of the size of the dataset, which means that it can be safely used for a real-time detection systems. Having the average time such small, it won't produce delays in the final system, ensuring an efficient intrusion detection.

7. CONCLUSIONS AND FUTURE WORK

The purpose of this thesis was to enhance a car's security system by implementing an intrusion detection system with machine learning. This system aimed to detect intrusions on CAN buses of passenger car's in real-time, using a Raspberry Pi as the hardware platform. The approach for obtaining such result involved multiple phases, including data collection from a passenger vehicle, implementation of an adversary model which served as the malicious node, finding an appropriate algorithm to perform a binary classification, establishing a connection to the hardware platform, and finally deploying the intrusion detection algorithm to perform classifications.

Data collection phase was performed with the use of software and hardware equipment, by connecting a laptop to the OBD-II port of a vehicle. Extracted data was further cleaned and analyzed, in order to have a comprehensive understanding on the collected frames. Next, there was implemented the adversary model using CANoe Simulation and the intrusions were inserted, creating a dataset. Then, with the use of machine learning, there were tested three candidate algorithms for the final implementation, out of which the LSTM model performed the best. Since it did not provide the best detection times while it was deployed on the Raspberry Pi, the model was converted to a lightweight format. The final model application showed detection times in terms of milliseconds, making it suitable for use in real world automotive industry.

The developed project presents a real solution for securing CAN buses with the use of machine learning. By enabling real-time filtering, both the security and the safety of the passengers can be improved. However, further improvements can be made to maximize these aspects. As a future work, there is seen the need to improve accuracy of the detection algorithm. For achieving this, some other architectures shall be tested and find one that proves more performance. There also can be improvements in detection times, since early detection is crucial for a secure CAN bus. Moreover, by implementing a feature that alerts the driver about potential attacks on their vehicle's CAN bus would add an extra layer of protection.

To conclude, this project proved the efficiency of using machine learning in enhancing security of CAN buses in passenger vehicles. Although there is room for improvement, this work represents a starting point for research and development in automotive security, for assuring safer and secure vehicles.

8. BIBLIOGRAPHY

- [1] *A binarized neural network approach to accelerate in-vehicle network intrusion detection*, Scientific Figure on ResearchGate, Accessed on 3 Jun, 2024. [Online]. Available: https://www.researchgate.net/figure/Typical-CAN-bus-attack-scenario_fig2_363703690.
- [2] K. Koscher, A. Czeskis, F. Roesner, *et al.*, “Experimental security analysis of a modern automobile,” in *2010 IEEE symposium on security and privacy*, IEEE, 2010, pp. 447–462.
- [3] I. Foster, A. Prudhomme, K. Koscher, and S. Savage, “Fast and vulnerable: A story of telematic failures,” in *9th USENIX Workshop on Offensive Technologies (WOOT 15)*, 2015.
- [4] S. Abbott-McCune and L. A. Shay, “Techniques in hacking and simulating a modern automotive controller area network,” in *2016 IEEE International Carnahan Conference on Security Technology (ICCST)*, IEEE, 2016, pp. 1–7.
- [5] F. Pascale, E. A. Adinolfi, S. Coppola, and E. Santonicola, “Cybersecurity in automotive: An intrusion detection system in connected vehicles,” *Electronics*, vol. 10, no. 15, p. 1765, 2021.
- [6] M. D. Hossain, H. Inoue, H. Ochiai, D. Fall, and Y. Kadobayashi, “Lstm-based intrusion detection system for in-vehicle can bus communications,” *Ieee Access*, vol. 8, pp. 185 489–185 502, 2020.
- [7] B. S. Bari, K. Yelamarthi, and S. Ghafoor, “Intrusion detection in vehicle controller area network (can) bus using machine learning: A comparative performance study,” *Sensors*, vol. 23, no. 7, p. 3610, 2023.
- [8] H. Lee, S. H. Jeong, and H. K. Kim, “Otids: A novel intrusion detection system for in-vehicle network by using remote frame,” in *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, IEEE, 2017, pp. 57–5709.
- [9] C. Dong, H. Wu, and Q. Li, “Multiple observation hmm-based can bus intrusion detection system for in-vehicle network,” *IEEE Access*, 2023.
- [10] W. Choi, K. Joo, H. J. Jo, M. C. Park, and D. H. Lee, “Voltageids: Low-level communication characteristics for automotive intrusion detection system,” *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 8, pp. 2114–2129, 2018.
- [11] S. C. HPL, “Introduction to the controller area network (can),” *Application Report SLOA101*, pp. 1–17, 2002.
- [12] *Can node schematic*, Accessed on 31 May, 2024. [Online]. Available: https://arm-software.github.io/CMSIS_5/develop/Driver/html/group__can__interface__gr.html.

- [13] *Standard can frame*, Accessed on 31 May, 2024. [Online]. Available: <https://www.csselectronics.com/pages/can-bus-simple-intro-tutorial>.
- [14] J. Cook and J. Freudenberg, "Controller area network (can)," *EECS*, vol. 461, pp. 1–5, 2007.
- [15] *How machine learning works*, Accessed on 1 Jun, 2024. [Online]. Available: <https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-ml/>.
- [16] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.
- [17] S. Raschka, *Stat 479: Machine learning lecture notes*, https://sebastianraschka.com/pdf/lecture-notes/stat479fs18/02_knn_notes.pdf, Accessed on 1 Jun, 2024, Department of Statistics, University of Wisconsin–Madison, 2018.
- [18] *Understand the k-nearest neighbors algorithm visually*, Accessed on 1 Jun, 2024. [Online]. Available: <https://www.jcchouinard.com/k-nearest-neighbors/>.
- [19] *Multi-layer perceptron in tensorflow*, Accessed on 1 Jun, 2024. [Online]. Available: <https://www.javatpoint.com/multi-layer-perceptron-in-tensorflow>.
- [20] *How to make a python perceptron from scratch*, Accessed on 1 Jun, 2024. [Online]. Available: <https://www.sharpsightlabs.com/blog/python-perceptron-from-scratch/>.
- [21] G. Van Houdt, C. Mosquera, and G. Nápoles, "A review on the long short-term memory model," *Artificial Intelligence Review*, vol. 53, no. 8, pp. 5929–5955, 2020.
- [22] *Long short-term memory (lstm)*, Accessed on 2 Jun, 2024. [Online]. Available: https://d2l.ai/chapter_recurrent-modern/lstm.html#fig-lstm-3.
- [23] *XL-driver-library, driver library for vector network interfaces*, Accessed on 2 Jun, 2024. [Online]. Available: https://cdn.vector.com/cms/content/products/XL_Driver_Library/Docs/XL_Driver_Library_Factsheet_EN_01.pdf.
- [24] P. S. Murvay, *An Introduction to Simulation and Analysis For In-Vehicle Network Communication with CANoe*. Editura Politehnica, 2021, p. 92.
- [25] H. D. Ghael, L. Solanki, and G. Sahu, "A review paper on raspberry pi and its applications," *International Journal of Advances in Engineering and Management (IJAEM)*, vol. 2, no. 12, p. 4, 2020.
- [26] *Raspberry pi 4 tech specs*, Accessed on 5 Jun, 2024. [Online]. Available: <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/>.
- [27] *Pi 3 click shield*, Accessed on 5 Jun, 2024. [Online]. Available: <https://www.mikroe.com/pi-3-click-shield>.
- [28] *Mcp2518fd click*, Accessed on 5 Jun, 2024. [Online]. Available: <https://www.mikroe.com/mcp2518fd-click>.

- [29] *Can frame cycle times not constant - reasons and remedies*, Accessed on 30 May, 2024. [Online]. Available: https://support.vector.com/kb?id=kb_article_view&sysparm_article=KB0012490&sys_kb_id=19c500bd1b1a4e10afd085d2604bcbe6&spa=1.
- [30] M. D. Hossain, H. Inoue, H. Ochiai, D. Fall, and Y. Kadobayashi, "Lstm-based intrusion detection system for in-vehicle can bus communications," *Ieee Access*, vol. 8, pp. 185 489–185 502, 2020.
- [31] *What is a confusion matrix in python (scikit-learn example)*, Accessed on 5 Jun, 2024. [Online]. Available: <https://www.jcchouinard.com/confusion-matrix-in-scikit-learn/>.
- [32] V. Labatut and H. Cherifi, "Accuracy measures for the comparison of classifiers," *arXiv preprint arXiv:1207.3790*, 2012.