CAN BUS AI-MODEL TO DETECT CYBER-ATTACKS

By Atif Khan C21078502

Supervised by Amir Javed

Moderated by Charith Perera

17/05/2024

School of Computer Science and Informatics, Cardiff University

BSc Computer Science

CM-3203

1 Abstract

This report uses machine learning models to detect cyber-attacks on a CAN bus and evaluate the results. The purpose of this report is to identify the best-performing machine learning model from the 4 that are being analysed. The models are to be used to detect cyber-attacks on a vehicle such as spoofing and DoS. To accomplish this, a realistic dataset will be used to calculate the accuracy of 4 well-known models. The dataset implements a way to differentiate between several ECUs. The report intends to aid researchers in better understanding the security of the CAN bus.

2 Table of Contents

1 Abstract 2
2 Table of Contents 3
3 Introduction 6
3.1 Aims and Objectives 6
3.2 Scope and Beneficiaries 7
4 Background 8
4.1 Why CAN Bus? 8
4.2 CAN Bus (Physical CAN Bus Components) 8
4.3 CAN Bus (Protocol Components) 10
4.3.1 CAN Bus Standard Frame 11
4.3.2 CAN Bus Extended Frame 12
4.4 Internet of Vehicles (IoV) 12
4.5 Python 13
4.5.3 Scikit-Learn 13
4.5.4 Pandas 13
4.5.5 NumPy 13
4.5.6 JupyterLab 13
4.5.7 Matplotlib 13
4.6 Past Work 13
4.7 Research Question 14
5 The Problem 15
5.1 Types of Attacks 15
5.1.1 Spoofing Attack 15
5.1.2 DoS Attack 15
5.2 Real-World Attacks on the CAN Bus 16
6 Methodology 16
6.1 Project Plan 16
6.2 Research Phases 16
6.3 Coding Phases 17
6.4 Visualisation Phase 17
7 Implementation 17
7.1 Dataset 17
7.1.1 How the Data was Obtained 18

7.1.2 Dataset Description 19
7.1.3 Dataset Representation 20
7.1.4 Personal Dataset Choices 21
7.2 Pre-Processing 22
7.3 Creating Models 23
7.4 Calculations and Displaying Calculations 23
7.5 Fixing Errors 24
7.6 Finding Parameters 24
7.7 Project Aim Changes 24
8 Machine Learning Models and Calculations 24
8.1 Multilayer Perceptron (MLP) 24
8.1.1 Input Layer 25
8.1.2 Hidden Layer 25
8.1.3 Output Layer 25
8.1.4 Connections and Weights 25
8.1.5 Bias Neurons 25
8.1.6 Activation Function 25
8.1.7 Diagram 25
8.2 Gaussian Naive Bayes (GNB) 26
8.2.8 Bayes' Theorem 26
8.2.9 Assumption of Feature Independence 26
8.2.10 Training 26
8.2.11 Classification 26
8.3 Decision Tree 26
8.4 Stochastic Gradient Descent (SGD) 27
8.5 Confusion Matrix 28
8.6 Precision 28
8.7 Recall 29
8.8 F1 Score 29
8.9 Macro Average 29
9 Results and Evaluation 30
9.1 Decision Tree 30
9.1.1 Analysis 32
9.2 Gaussian Naïve Bayes 32
9.2.2 Analysis 34

9.3 Multilayer Perceptron 34
9.3.3 Parameters 36
9.3.4 Analysis 36
9.4 Stochastic Gradient Descent 36
9.4.5 Parameters 38
9.4.6 Analysis 38
9.5 Graphs 39
9.5.7 Label 39
9.5.8 Category 40
9.5.9 Specific Class 41
9.6 Evaluation 42
10 Conclusion and Future Work 43
11 Reflection 43
12 References 45

3 Introduction

The CAN bus (control area network) is a serial communication protocol that allows devices to exchange data reliably and efficiently. This allows the electronic control units (ECUs) which control the electrical subsystems in a car to communicate with each other in a priority-driven fashion and reliably by not being interrupted by noise. It is widely accepted and is used in most vehicles, machines, and devices.

The current trend in the automotive industry is to have cars that can self-drive safely on the roads but having the cars to drive safely is already a tough objective. Innovations have increased the attacking surface of what cyber-attackers can exploit remotely, which will cause security issues to the passengers and the people around them. The most common attacks are spoofing and DoS and the frequency of cyber-attacks on vehicles has increased by 225% from 2018 to 2021 [3].

The Internet of Vehicles (IoV) is a way the security of the roads is trying to be improved. IoV aims to have a network infrastructure where everything on the road communicates with each other. This will improve the safety and efficiency of the roads. However, all parts of the IoV communicate with each other, and cyber threats are a big problem with the implementation and safety of the IoV.

Attacks have been widely reported as talked about in this article [19]. CAN packets are injected into the vehicle with a small device to act as the ECU. The device can send data to the CAN bus to wake it up. Then the device is used to block other devices from accessing the CAN bus and disables its error mechanism allowing the device to bypass the security. From there the attacker can do whatever it wants. In this case, it tells the CAN to unlock the car, which then gets stolen. The official Toyota UK has posted an article on the matter and how they release security updates to try and prevent the attacks [14].

Using machine learning to detect cyber-attacks is the main way moving forward. The machine learning models have many advantages. The models learn from all the data, allowing them to make better judgements as it runs and the data is updated. The constant updates mean it will only get better as time goes on. Another advantage is the ability of the models to detect trends and patterns in the data. Some types of attacks like DoS, overload the CAN bus with signals, which some models may detect as malicious.

The outcome of this project is to provide an analysis of the chosen machine learning models and find the best model out of the chosen models. The model should be the best at detecting cyber-attacks in real time. The project should provide all the analysis needed to select the best model from the tested ones, providing graphs and comparisons.

3.1 Aims and Objectives

The overall aim is to identify the best-performing AI model for a CAN bus dataset, seeing if it can detect all attacks on a dataset. Machine learning and deep learning are to be used. The objective to complete this is to find a CAN bus dataset with multiple attacks and then to preprocess the dataset to prepare it for the AI model. The next objective is to identify the best-performing AI models by testing multiple models on the

dataset. A desirable objective is to visualise the results to easily identify the best-performing AI models.

3.2 Scope and Beneficiaries

The scope of this project is wide. Any vehicle drivers (especially self-driving), researchers and the vehicle industry fall under the scope of the project. It is useful to researchers who are finding the best way to discover cyber-attacks on a CAN bus. The project is useful to individuals and companies in the vehicle industry as the security of the CAN bus is a big point in vehicle security. It is also important to the self-driving industry to protect the roads from cyber-attackers.

4 Background

CAN bus networks are widely used in vehicles today. It has 2 intertwined wires, the CAN-high and CAN-low. 3 layers deal with the signals it receives. There are 2 data frames, a standard and an extended frame. The CAN bus is linked to the Internet of Vehicles (IoV) as it is the main communicator between vehicles on the road. This project is very important for the safety of passengers in vehicles and the safety of car owners. Car theft and attacks are a big problem with new cars. The CAN bus was not designed with security in mind, so finding a way to detect cyber-attacks is a main aim for researchers.

4.1 Why CAN Bus?

CAN bus is the standard and widely accepted because of many reasons. Vehicles started the same way houses are wired today, relying on point-to-point wiring, which started to cause problems with the increase of devices in vehicles. This increased the weight of the vehicle making the public pressure the automobile manufacturers to improve fuel efficiency. CAN bus solved this and much more.

The CAN bus was simple and low-cost. This was because the system communicated via direct complex analogy signal lines which reduced errors, weight, wiring and cost which in turn helped with fuel efficiency with the reduced weight of the vehicle. The CAN bus is fully centralised. This enables centralised diagnostics, data logging and configuration. With how it is made, the CAN bus is extremely robust, meaning it is robust against electric disturbances and electromagnetic interference (noise) which is ideal for safety and safe against errors, but why is it so robust?

4.2 CAN Bus (Physical CAN Bus Components)

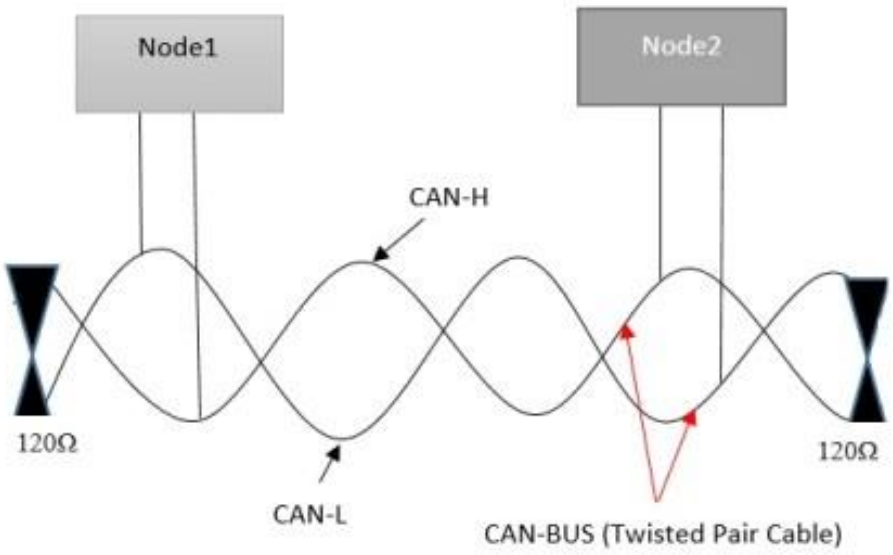

Figure 1.1: CAN bus physical architecture [6].

The CAN bus is made of a pair of twisted cables, a CAN-H (high) and a CAN-L (low) cable (See Figure 1.1). The cables are twisted to reduce cross-talk which is unwanted transfer of signals and reduces electromagnetic interference. This is why it is reliable/robust.

Having 2 wires also helps with the CAN bus being robust to noise. With 1 wire, when a receiver is looking at the voltage of the signal, the voltage indicates if a 1 or a 0 is being transmitted. Electromagnetic interferences can cause a false positive in the signal.

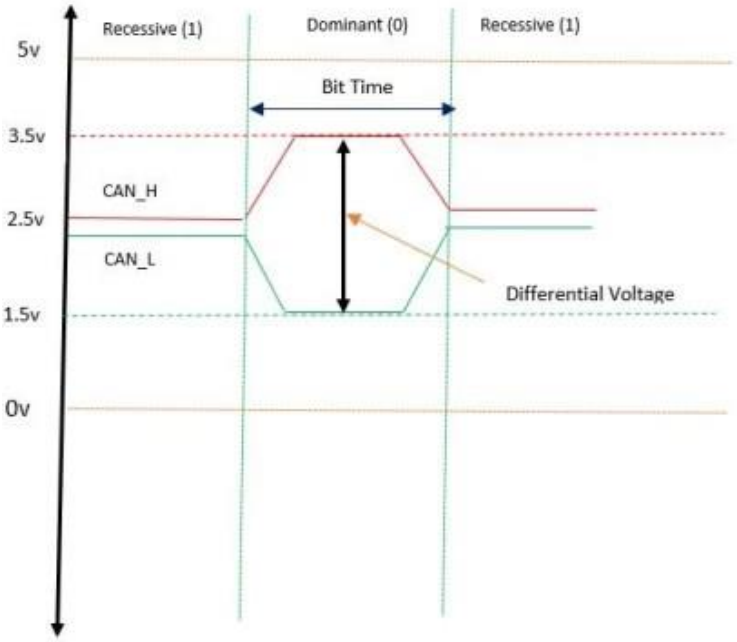

Figure 1.2: CAN bus signals [6].

2 wires work by having 2 signals that are mirrored. 0s and 1s are calculated by the difference of the signal between the CAN_H and CAN_L signal. In the case of the CAN bus, 0 is the dominant bit and 1 is the recessive bit. This makes the CAN bus robust to noise as any electromagnetic interference outside of the dominant bit will not be detected as a dominant bit and any interference in the dominant bit would not matter as the difference (differential voltage) is calculated to detect a dominant bit (see figure 1.2).

In Figure 1.1, the 120Ω at the end of each node are terminators that absorb the signal at the end to avoid reflection of the signal. The Terminator register must be equal to the impedance generated. This is usually 100-120Ω.

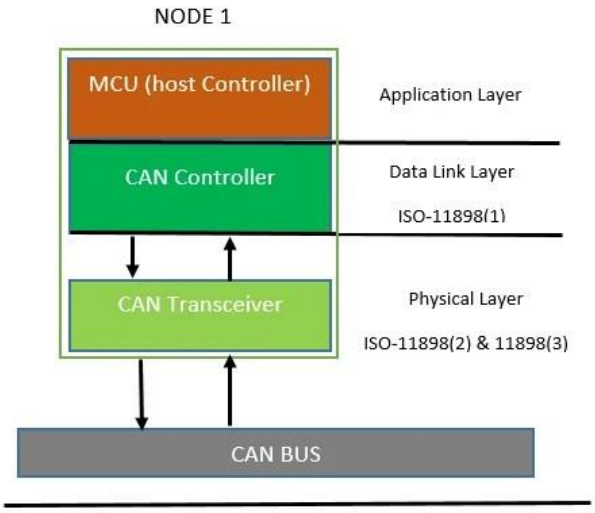

Figure 1.3: CAN bus node [6].

In Figure 1.1, node 1 and node 2 relate to what the CAN network is made up of, ECUs. These are known as nodes. Figure 1.3 shows the structure:

- Micro-controller unit – decides how to deal with received data or data to be transmitted.
- CAN controller – applies acceptance filtering as it receives whole data until the complete message is not received. Sends data on CAN bus serially when the bus is idle.
- CAN transceiver – converts the transmit signal from the CAN controller level to the CAN bus level and vice-versa.

There are 2 types of CAN, high speed, and low speed. High-speed CAN (ISO 11898-2) operates at higher bit rates, normally up to 1Mbps. It is commonly used in applications that need higher data transfer rates and short response times. Low-speed CAN (ISO 11898-3) operates at lower bit rates, normally up to 125kbps. It is used in applications where low transfer rates are okay. It is good for long distances and has low power consumption.

4.3 CAN Bus (Protocol Components)

There are 2 types of CAN bus frames. The standard (or classical) CAN frame and the extended CAN frame. The extended CAN frame is used more for heavy vehicles whereas the Standard CAN frame is used in passenger vehicles.

4.3.1 CAN Bus Standard Frame

Field Name	Sub-Field	Length(bits)	Purpose
Start of Frame	SOF	1	Indicates the start of frame on CAN Bus(must be dominant (0))
Arbitration Field	Identifier	11	Decides the Message Priority(Arbitration) on CAN BUS
	RTR	1	Differentiate between Remote Frame or Request frame
Control Field	IDE	1	Tells about frame format : standard or extended For Standard : Always Dominant(0) For Extended : Always Recessive(1)
	R0	1	Reserved(must be dominant 0)
	DLC	4	Data length on the CAN bus
Data Field	D0-D8	64	Data
CRC Field	CRC	15	CRC
	CRC Delimiter	1	Must be recessive(1)
Ack field	Ack	1	Acknowledgment by receiving node.
	Ack Delimiter	1	Must be recessive(1)
End Of Frame	EOF	7	Indicates end of current frame(Must be recessive(1))

Figure 2.1: CAN bus standard frame [5].

- SOF: Used to signal the start of a new message and that a new message has started.
- Identifier (2.0A): 11-bit identifier allowing for 2048 unique identifiers. This allows the CAN bus to determine the priority of messages and decide whether to transmit or yield the bus based on the identifier.
- RTR (Remote transmission request): Used to differentiate between remote frame and request frame.
- IDE (Identifier Extension): Indicates whether the frame format is standard or extended.
- R0 (Reserved bit 0): Reserved for future use.
- DLC (Data length code): Specifies the number of bytes of data.
- D0-D8: Indicates the data.
- CRC (Cyclic redundancy check): Verifies that the messages are properly sent over the bus.
- CRC Delimiter: A marker that gives time or space to the ECU to calculate the CRC.
- ACK (Acknowledgement): Confirms that the message was received with no errors.
- ACK delimiter: Distinguish between ACK errors and CRC errors.
- EOF (end-of-frame): Marks the end of a CAN frame and disables bit-stuffing.

4.3.2 CAN Bus Extended Frame

Field Name	Sub-Field	Length(bits)	Purpose
Start of Frame	SOF	1	Indicates the start of frame on CAN Bus(must be dominant (0))
Arbitration Field	Identifier	11	Decides the Message Priority(Arbitration) on CAN BUS
	SRR	1	Always Recessive
	IDE	1	Tells about frame format : standard or extended **For standard**: always dominant(0) **For Extended**: Always recessive(1)
	Identifier	18	Decides the Message Priority(Arbitration) on CAN BUS(Extended)
	RTR	1	Tells about remote or Data frame
Control Field	R0	1	Reserved(must be dominant 0)
	R1	1	Reserved(must be dominant 0)
	DLC	4	Data length on the CAN bus
Data Field	D0-D8	64	Data(As per DLC)
CRC Field	CRC	15	CRC
	CRC Delimiter	1	Must be recessive(1)
Ack field	Ack	1	Acknowledgment by receiving node.
	Ack Delimiter	1	Must be recessive(1)
End Of Frame	EOF	7	Indicates end of current frame(Must be recessive(1))

Figure 2.2: CAN bus extended frame [5].

The CAN Bus extended frame has the same sub-fields as the standard frame with a few differences.

- SRR (Substitute Remote Request): If the standard frame and extended frame have an equal base identifier, the standard frame will have higher priority because of SRR. SRR is always recessive.
- Identifier: Extended frame has an extra 18 bits compared to the standard frame. The identifier length is 29 bits, allowing for 536 million unique identifiers.
- R1 (Reserved bit 1): Reserved for future use.

4.4 Internet of Vehicles (IoV)

IoV is a network that connects pedestrians, cars, and parts of the infrastructure. It uses sensors, software, hardware, and types of connection to enable reliable and continuous communication. The driving factor of IoV is to make transportation autonomous, safe, fast, efficient and to reduce the impact on the environment. It works by relying on car manufacturers to install the correct hardware for networking and data gathering. It needs, for example, traffic lights to be connected to smart vehicles for them to work together.

The architecture has 3-man layers. The perception includes various sensors and devices to collect data to make it functional. Things like smartphones and vehicle cameras. The network layer is responsible for making vehicles that are connected visible on the IoV, so they can send information about traffic and other road information. The application layer stores and applies the data that is gathered. It is responsible for identifying people and cars connected to tell the car what to do.

IoV is important because of its main advantages. This includes safety on the road, as the cars are connected, it avoids driving errors. IoV will make the roads more

efficient by eliminating traffic and finding the fastest way around the roads. It will also improve the environment.

4.5 Python

Python is being used for this problem because of its variety of modules, ease of use and the number of classifiers that exist in the modules. I am proficient in Python, and it is the coding language with which I have the most experience. Python is also highly regarded for machine learning because of its simplicity and the code is shorter. It can take longer to run code but the ease of writing the code saves time when doing machine learning. Below are the packages used and the reason for using them. Python is widely used so troubleshooting by myself was easier than using a less-used language.

4.5.3 Scikit-Learn

Scikit-learn has many uses for machine learning that make doing machine learning in Python a lot easier. For example, it includes all the machine learning classifiers that I need and has train-test splitters that make the training and testing data with 1 command. It is simple to use and provides functions to get the calculations needed.

4.5.4 Pandas

Pandas is used to make graphs and to make the data easier to read. It is an open-source library which is widely used for data analysis and manipulation. Pandas is used in this project to read the dataset and manipulate it to fit the needs of the machine learning models. The dataset was slightly large which pandas is proficient in dealing with.

4.5.5 NumPy

NumPy is an open-source library that enables numerical computing in Python. It is used in this project to do calculations on arrays simpler compared to doing it without libraries.

4.5.6 JupyterLab

JupyterLab is a web-based environment used to run code in a modular way. I used it to save time by not having to repeatedly run time-consuming code and to separate the code in an organised fashion. It also provides simple outputs and sufficiently shows the graphs and outputs.

4.5.7 Matplotlib

Matplotlib is a comprehensive library for creating visualisations in Python. It is used to create the confusion matrices and to create the graphs to compare the calculations further in the report. It allows for easy-to-manipulate tables and works well with JupyterLab.

4.6 Past Work

This paper [4] uses the same dataset that I will be using in this paper. It converts the hexadecimal dataset into binary and decimal values, which is not necessary for this method. It uses 4 different machine learning models to test both the binary and decimal dataset, checking for each of the classification columns. The 4 models are widely used

but may not be the best possible model choices compared to the models used further in this research paper. The paper mentions for future work it intends to try different machine learning models which will be done further in this paper.

This research [9] focuses on a way to reverse engineer CAN messages to help researchers. It aims to speed up the reverse engineering of CAN messages to reduce the human effort required to do this. This can be used in the future to create more datasets for CAN bus cyber-attack data. The dataset I am working on does not have raw CAN bus data so this has no use in my research however, in the future, if CAN bus data is sourced it can be used.

This method [12] analyses the Inter-arrival time of signals in a CAN bus. The attacker in this scenario has 2 types of attacks. In the first scenario, it removes CAN messages for a variable amount of time. The second scenario is an attacker that permanently disables an ECU in the CAN. Although it yields great results, it is not indicative of a real-world, real-time attack. Non-simulated CAN traffic has a high variability of inter-arrival time, which can lead to many false positives occurring in the model.

This paper [2] uses real-world datasets to detect DoS, fuzzy, spoofing, flooding, and malfunctions. It uses SVM, DT and KNN models to analyse the dataset. The analysis provides results with high accuracy however the dataset has no differentiation to which ECUs are being attacked. The dataset used further in my project has different files for different ECUs in the vehicle, which will be classified. The paper also does not employ any deep learning models, which can provide a variety of models to be analysed and can potentially yield better results.

The research in [17] provides a machine learning model that is called AMAEID, auto attention mechanism and autoencoder for intrusion detection. It yields results better than most machine learning models, but it has the potential to miss small changes in CAN messages or may fail with massive attacks. Other models also exist that perform on a similar or better level which have also been widely used.

This research [15] uses anomaly detection using the HTM algorithm. It works by capturing the structure and algorithmic features of the new cerebral cortex. It provides great results, but the training time is high and can have high false positive detections.

4.7 Research Question

The project aims to develop machine learning models to detect cyber-attacks on CAN bus data, and then show the accuracy of each model on the predictions.

To achieve the aim, this project will fit models on a chosen CAN bus dataset and analyse the metrics of the predicted data, to find a model that is best fit to detect CAN bus cyber-attacks on vehicles. It is important to test models on the dataset that have not been tested before.

5 The Problem

The main problem with the CAN bus is that it is prone to attacks. CAN bus was not designed with security in mind. The specifications of the CAN bus message syntax are included in the communication database for CAN (DBC). The DBC has specifications that define the cycle time, the sender ECU, the intended destination ECU and all the other information needed to interpret the message. Each vehicle has a different DBC, and the specifications are kept confidential for each car manufacturer. However, researchers have already reverse-engineered these documents. The confidentiality of the information makes it harder for security researchers to develop ways to stop cyber threats from happening.

5.1 Types of Attacks

5.1.1 Spoofing Attack

A spoofing attack is when an attacker impersonates an ECU on the CAN bus and sends messages with a forged source address. This can allow the attacker to disable specific ECUs and allow the attacker to, for example, open doors or inject malicious data into cars on the road. The attacker would need to have data that the car manufacturers keep confidential but has been reverse-engineered. The data would be used to create malicious attacks.

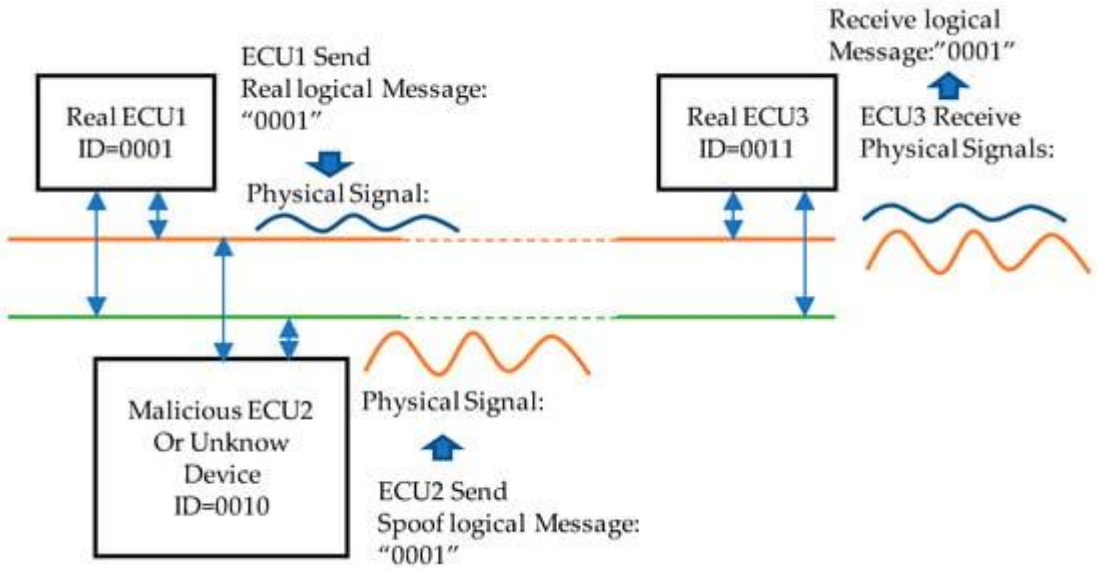

Figure 3.1: Spoofing attack example [20].

This research paper [20] provides an example of a spoofing attack. The ECU2 would send a signal using the ID from ECU1 as it will only approve messages sent from that ID. If a signal was sent with the ID of ECU2, it is unknown so it would not be accepted.

5.1.2 DoS Attack

DoS attack is when the attacker floods the CAN bus with many messages, overwhelming the CAN bus and rendering the CAN bus to not function properly. It can disrupt communication and possibly make the system fail. This research [7] mentions that there are multiple options to do this. The first option is to send as many messages as possible to the CAN bus with the lowest ID. This is done because when the CAN bus

is idle, the ECUs that want to transmit with the lowest ID will have priority. None of the regular messages will win the arbitration which will lead to the other ECUs never sending messages. The second option is to force the CAN low and CAN high into the dominant state and hold it there as long as needed for the attacker. This method can cause the CAN bus to trigger its error detection. The first solution follows the rules of the CAN bus, and the ECUs will assume everything is okay. Both methods are usable.

5.2 Real-World Attacks on the CAN Bus

Lexus wrote an article [13] on a vulnerability in their CAN bus. They provided enhanced security hardware. It tells its users that the thieves use a device to gain physical access to the CAN bus. It works by sending priorities CAN signals to bypass the vehicle's security and immobiliser systems, letting the thieves unlock the car. They disconnect the emergency start device and let the thief start the car. Toyota also released an article [14] to its users addressing CAN bus attacks. The method is the same.

Although Lexus has tried to combat the attacks, there are still recent posts on the forums of CAN bus attacks after the security update mentioned in the article, in 2021 [16].

This article [19] explains that the attackers access the CAN bus through the headlights. Security updates are constantly released however, the updates are temporary as the attacker can adapt to the changes and for example, reverse engineer the ECU messages and create another device.

6 Methodology

6.1 Project Plan

Using Microsoft Excel, a time management sheet was created to have a set structure on the order of work and whether it was done or not. It had columns on if work was done and a column for the date it was done. The order of the tasks was based on my initial plan. The structure of my initial plan was as below.

- Week 1: Initial Plan.
- Week 2: Research and learn machine learning in Python.
- Week 3-4: Find datasets and possible AI models to use.
- Week 5: Pre-process dataset and begin the final report.
- Week 6-9: Finish code for machine learning AI model and complete necessary final report sections.
- Week 10: Visualise data.
- Week 11-12: Finish report and do finishing touches.

6.2 Research Phases

The research phase began with learning all the information about the CAN bus and researching AI models on Scikit-learn. Going through research papers to see examples of AI models to detect cyber-attacks. It was also important to find datasets that will provide a variety of results and a variety of variables that can be analysed.

Phase 2 is collecting the research and gathering what is needed for the project. This includes all the research for the background, the chosen models, and the chosen dataset.

Phase 3 is to analyse the data and gather the results. All the results are documented and ready to be used in the project.

6.3 Coding Phases

The coding phase begins with preprocessing the dataset as needed to begin the models. The dataset needs to be split into 2 data frames. In the dataset, if there are any non-numerical data columns, they need to be replaced with dummy values. At the end of the phase, the data frames need to be split to create the training and testing sets.

The next phase is creating the models. Using the 4 models picked in the research phase, fit the models onto the dataset. Any parameters need to be tested, and calculate the time taken when running the models. Then predict the values for the testing set.

The last phase is to compute the calculations. Using the Python libraries, calculate the confusion matrix, recall, precision, F1 score, and time taken.

6.4 Visualisation Phase

The visualisation phase is gathering the data from the coding phase and creating graphs and tables comparing the data. Once this is completed, the project can be finished.

7 Implementation

7.1 Dataset

The chosen dataset is the CICIoV2024. The dataset aimed to propose a realistic benchmark to support the development/research of new solutions for cyber security for IoV. 5 attacks were done on a 2019 Ford car which had all its ECUs intact. The vehicle was rendered immobile to not cause any harm to a passenger.

7.1.1 How the Data was Obtained

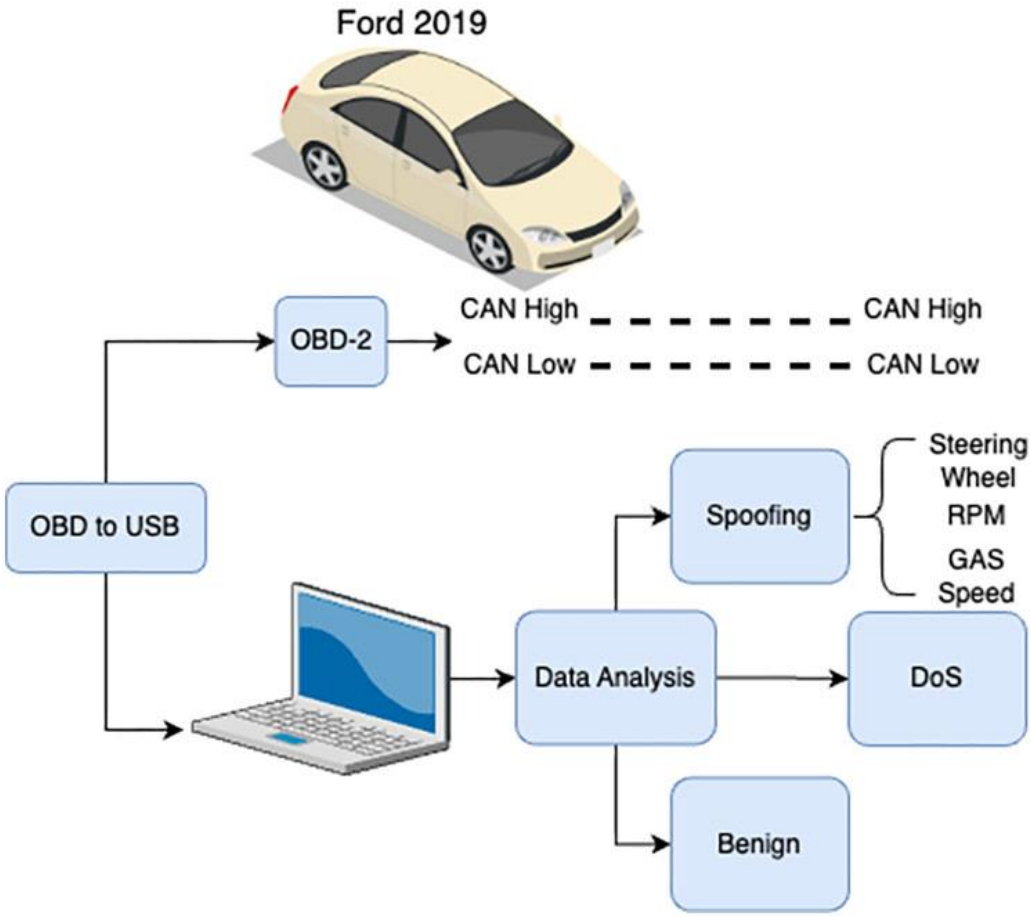

Figure 4.1: A diagram of how the data was obtained [10].

Figure 4.1 shows the layout of how the data was obtained. The OBD-2 port was used to do the attacks. The OBD-2 port is located by the steering wheel and is used to access the vehicle's computer. It was previously thought that attacking a car through this port would be unrealistic because it assumes that the car is only being attacked with physical access. However, with the many ways of connecting to a modern vehicle, it is now a valid approach. The OBD-2 port can also be accessed remotely.

The research [10] linked to the dataset writes about the hardware and devices used. It uses a USB2CAN device and ELM327 as a Bluetooth-based device. Macchina M2 [11] is software that can be used in the OBD-2 port and is an interface to communicate with the CAN bus. Finally, software like socketCAN, can-utils, can and

Wireshark is used to analyse the CAN packets. To send CAN messages, can-utils is used. To sniff the packets, cansniffer is used. You can find out more in the paper [4].

Figure 4.2: Pictures showing the 2019 Ford with all its ECUs, rendered immobile [11].

7.1.2 Dataset Description

Feature name	Description
ID	Arbitration - indicates the priority of the message and the type of data it carries.
DATA_0	Byte 0 of the data transmitted.
DATA_1	Byte 1 of the data transmitted.
DATA_2	Byte 2 of the data transmitted.
DATA_3	Byte 3 of the data transmitted.

DATA_4	Byte 4 of the data transmitted.
DATA_5	Byte 5 of the data transmitted.
DATA_6	Byte 6 of the data transmitted.
DATA_7	Byte 7 of the data transmitted.
label	The identification of benign or malicious traffic. (“Benign” or “Attack”)
category	The identification of the category to which the traffic belongs. (“Benign”, “DoS” or “Spoofing”)
specific_class	The identification of the specific class of the traffic. (“Benign”, “Gas”, “RPM”, “Steering_wheel”, “Speed”, or “DoS”)

Table 1.1: Explaining the features of the dataset [11].

7.1.3 Dataset Representation

	Mean	std	min	25%	50%	75%	max
ID	537.207946	322.479994	65	357	516	578	1438
DATA_0	71.0865995	88.9771748	0	0	16	127	255
DATA_1	69.9892503	95.5837431	0	0	12	128	255
DATA_2	55.0112724	72.7658378	0	0	13	125	255
DATA_3	57.4536383	90.3207664	0	0	0	92	255
DATA_4	45.2851673	64.4583498	0	0	6	86	255
DATA_5	53.8826134	94.3361202	0	0	0	63	255
DATA_6	71.7491441	101.687183	0	0	0	138	255
DATA_7	60.2747691	99.9654672	0	0	0	80	255

Table 1.2: Data representation of the dataset [11].

The table headers are mean, standard deviation, minimum, 25% percentile, 50% percentile, 75% percentile, and maximum from left to right.

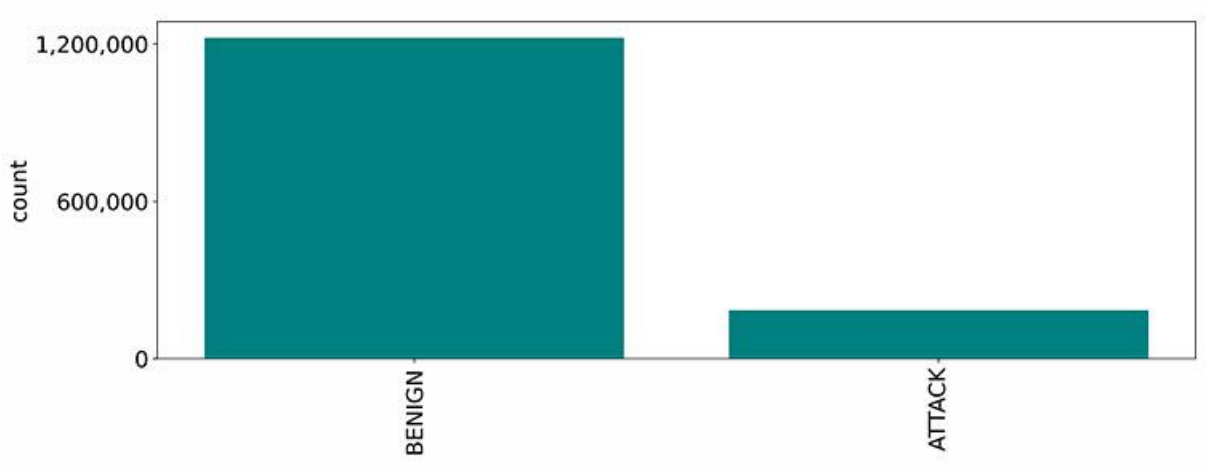

Figure 4.3: Graph showing “label” column representation in the dataset [11].

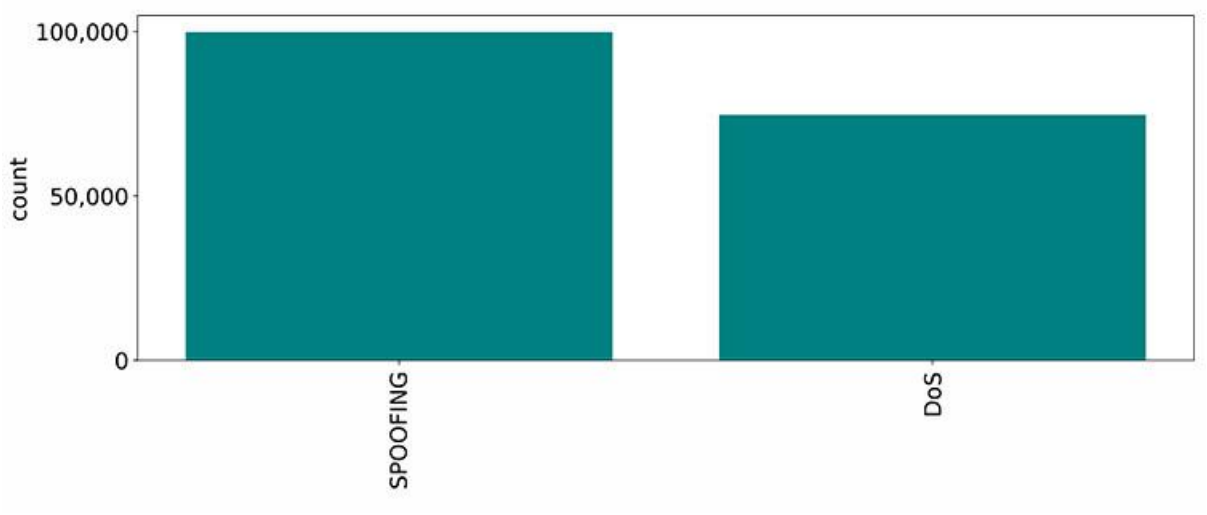

Figure 4.4: Graph showing "category" column representation in the dataset [11].

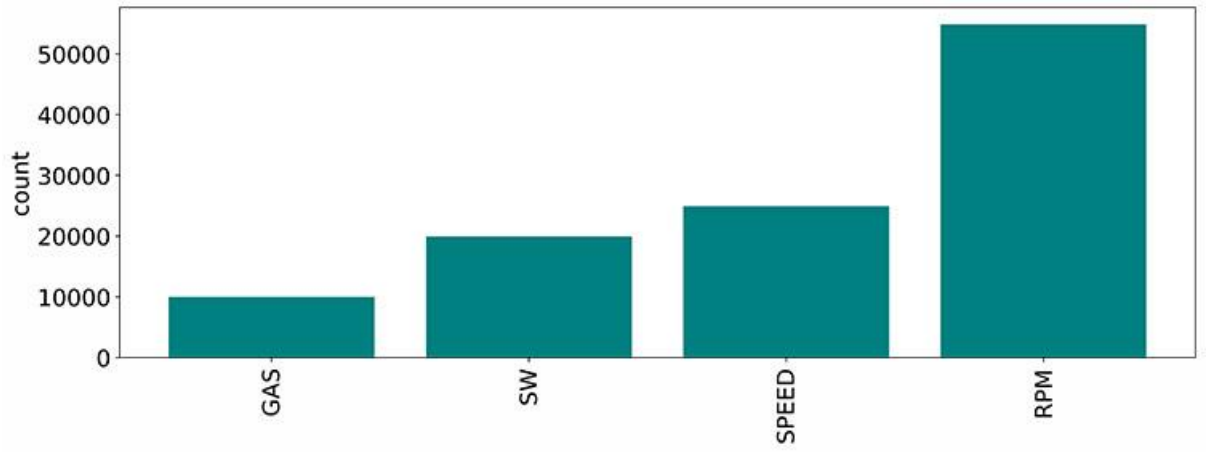

Figure 4.5: Graph showing "Specific_class" column representation in the dataset [11].

7.1.4 Personal Dataset Choices

The dataset has 3 files, hexadecimal, binary and decimal. I picked decimal because it has fewer columns for machine learning which speeds up the model process and the data does not need to be converted. Hexadecimal was not picked as it contains letters meaning it would need to be converted to decimal or binary. Binary was not picked as it had a high number of columns.

7.2 Pre-Processing

The dataset comes in multiple files for each specific class of the ECUs. To test models on the dataset, these files need to be combined into 1 file. To do this simply and effectively, a piece of code was used to combine all CSV files.

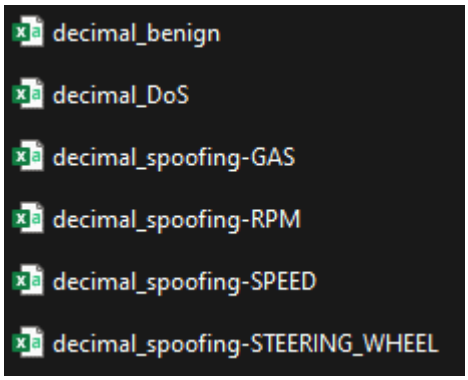

Figure 4.6: Dataset files before pre-processing.

```
import shutil
import glob
#import csv files from folder
path = r'decimal'
allFiles = glob.glob(path + "/*.csv")
allFiles.sort()  # glob lacks reliable ordering, so impose your own if output order matters
with open('combined_dataset.csv', 'wb') as outfile:
    for i, fname in enumerate(allFiles):
        with open(fname, 'rb') as infile:
            if i != 0:
                infile.readline()  # Throw away header on all but first file
            # Block copy rest of file from input to output without parsing
            shutil.copyfileobj(infile, outfile)
            print(fname + " has been imported.")
```

Figure 4.7: Code to combine dataset files.

The code in Figure 4.7 is how the files were combined. The files combined to create “combined_dataset.csv”. The dataset is read using the pandas “read_csv” command to read the dataset into JupyterLab.

```
X = (df.drop(["label","category","specific_class"], axis=1))
Y = (df.drop(["ID","DATA_0","DATA_1","DATA_2","DATA_3","DATA_4","DATA_5","DATA_6","DATA_7","category","specific_class"], axis=1))

# convert target column to dummy values
label = le.fit_transform(list(Y["label"]))
Y.drop("label", axis=1, inplace=True)
Y["label"] = label
```

Figure 4.8: Code to create data frames and create dummy values.

Next, the dataset is split into 2 data frames to split up the data and the columns that will be classified. After making the data frames, the classification columns are changed from characters to digits so the machine learning models can be used on the data. Figure 4.8 shows the creation of the data frames and the conversion of the target column to dummy values.

```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, random_state=15, test_size = 0.2)
```

Figure 4.9: Code to split the data to create train and test portions.

To run the machine learning models, the data needs to be split into training and test data. This was done by using Scikit-learn "train_test_split". I picked a test size of 20% and set a random state to keep the results consistent whenever the code was run. The preprocessing code was repeated for each classification column needed.

7.3 Creating Models

To create the models, the Scikit-learn library provides functions for each model needed and extra functions for the later calculations. First, the model needs to be initialised, then the model needs to be fit using the "X_train" and "Y_train" variables that I created when pre-processing. Below is an example of the code used to create the models. The example also has some code for the calculations of the "Time taken".

```
dtc = DecisionTreeClassifier()
```

```
start = time.time()
dtc.fit(X_train, Y_train)
end = time.time()
length = end-start
```

Figure 4.10: Example code of one of the models being initiated and fitted.

7.4 Calculations and Displaying Calculations

To do the calculations, the Scikit-learn library provides functions to calculate recall, precision and F1 measure. It also provides functions to display the confusion matrix in a nice image, used in conjunction with Matplotlib. The time taken is also shown being calculated.

```
length = str(length)
print("Time: "+length+" seconds")
```

```
 y_pred = dtc.predict(X_test)
```

```
cm = confusion_matrix(Y_test, y_pred, labels= dtc.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=["ATTACK","BENIGN"])
disp.plot()
plt.show()
```

Figure 4.11: Example code of one of the models' results being calculated and displayed.

To create the graphs, I used Matplotlib to display bar charts as subplots to allow them to be compared. The plots get printed into JupyterLab.

```
# Create the first subplot
ax1 = plot.subplot(1, 3, 1)
ax1.bar(models, c1, width=width, color=bar_colors, align="center")
ax1.set_title('Recall')
ax1.set_ylabel('Value')
```

Figure 4.12: Example code of graph creation.

7.5 Fixing Errors

When creating the confusion matrix for models that provide float values, the data needs to be rounded and the shape of the data needs to be changed. The confusion matrix does not accept float values. Below is the code used to fix this. For example, Gaussian Naïve Bayes needs the values rounded.

```
nb_pred = gnb.predict(X_test)
nb_pred2 = np.round(nb_pred).astype(int)
nb_pred2 = nb_pred2.ravel()
```

Figure 4.13: Example code of the data being rounded and changing shape.

7.6 Finding Parameters

When calculating parameters for models that needed them, I used a function in Scikit Learn called "GridSearchCV". This goes through all the combinations of parameters given and tests it with the given model. This was important to find the best parameters for the model as the default parameters cannot be suitable for the model. A problem with doing this is the time taken to run it, so every combination was not possible in the time frame. For example, trying to calculate the maximum iterations would increase the time exponentially.

```
parameters = {'max_iter': [10], 'alpha': 10.0 ** -np.arange(1, 10), 'hidden_layer_sizes':np.arange(0, 15),'learning_rate': ['constant','adaptive']}
clf = GridSearchCV(neural_network.MLPClassifier(), parameters, n_jobs=-1)

clf.fit(X_train, Y_train)
print(clf.best_params_)
```

Figure 4.14: example code showing how the best parameters were found.

7.7 Project Aim Changes

In my initial plan, an over-ambitious aim was testing machine learning models on multiple datasets. This was not feasible because of time constraints. Testing 4 models on multiple datasets and analysing the data would take too much time.

8 Machine Learning Models and Calculations

To analyse the dataset, I will be using Multilayer perceptron, gaussian naïve Bayes, decision trees and stochastic gradient descent. I am using these as they provide a variety of machine learning model types and test models that have not been tested before on the dataset. For the calculations to analyse, I chose precision, recall, F1 score, and time taken. I chose these to give a representation of how the model performed and if it can be used in real-time.

8.1 Multilayer Perceptron (MLP)

Multilayer perceptron (MLP) is a type of artificial neural network (ANN). It is multiple layers of interconnected nodes or neurons. It is a feedforward neural network which means all the information moves in one direction, going through the layers. This allows the model to learn nonlinear relationships in data making it powerful for the classification. I picked this model as it is very good at finding patterns in data however, it may take too long to calculate for a real-time scenario.

8.1.1 Input Layer

The input layer consists of nodes or neurons that receive the input data. Each neuron represents a feature or dimension of the data. The number of these neurons is determined by the dimensions of the data.

8.1.2 Hidden Layer

These layers are the intermediate layers between the input and output layers. It can have 1 or more layers of neutrons which take in inputs from the neurons of the layer before ad produce an output. The output is sent to the next layer and is repeated for each layer. The number of neurons and layers are parameters that need to be determined when designing the model.

8.1.3 Output Layer

The output layers consist of neurons that produce the final output of the MLP. The number of neurons depends on the nature of the task.

8.1.4 Connections and Weights

Neurons in one layer are connected to neurons of the subsequent layer through weighted connections. Each connection has a weight that determines the strength of influence a neuron has on another neuron.

8.1.5 Bias Neurons

Each layer usually has a bias neutron that provides a constant input to the neurons in the next layer. It has its weight associated with each connection which is determined during training. The bias neuron affects the activation function of the neurons in the subsequent layer, letting the network learn to control the threshold for activating and better fitting the data.

8.1.6 Activation Function

Each neuron in every layer except the input layer applies an activation function to the weighted sum of inputs.

8.1.7 Diagram

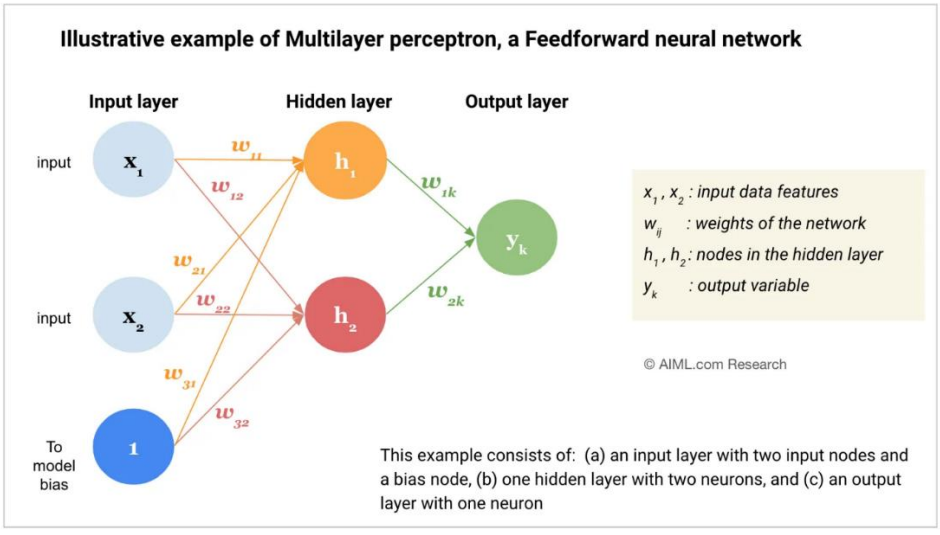

Figure 5.1: MLP diagram [1].

8.2 Gaussian Naive Bayes (GNB)

Gaussian Naïve Bayes (GNB) is a variant of Naïve Bayes. It is a probabilistic classifier based on Bayes' Theorem and assumes independence between features, which is why it is called "naïve". I picked this because of its simplicity and efficiency in training. It also works well with high-dimensional data. The model can handle large datasets well, making it good for this project. The speed of the model means it can be used in a real-time scenario.

8.2.8 Bayes' Theorem

Bayes's theorem is a way of finding probability when we know other probabilities. Below is the equation. "P(A|B)" is how often A happens given that B happens. "P(B|A)" is how often B happens given that A happens. "P(A)" is how likely A is. "P(B)" is how likely B is. We know of the values on the right of the equation.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

8.2.9 Assumption of Feature Independence

GNB assumes that the value of a feature in a class follows a Gaussian distribution and that different features and independent from each other within a class. So, the likelihood of the features can be calculated using the probability density function of the Gaussian distribution.

8.2.10 Training

During training, GNB calculates the mean and the standard deviation of each feature for each class.

8.2.11 Classification

For each new instance in the classification, GNB calculates the likelihood of the instance belonging to each class based on the Gaussian distribution of the features in that class. Bayes theorem is used to calculate the posterior probability of each class in the instance. The highest posterior probability is the one chosen for the predicted class.

8.3 Decision Tree

Decision trees are used for both classification and regression. It is named a tree as it is structured like a tree with branches and leaves. They work by splitting the data into regions based on the values of input features. At each node of the tree, a decision is made based on the feature. It traverses through the branches of the tree until it reaches the leaf node, which provides a predicted value or outcome. I picked this model to compare it to MLP. Decision trees are simple and quick, so the time taken is very low. It

is efficient when dealing with large datasets making it very scalable. Additionally, it can be adaptable to new data which is perfect for a real-time scenario.

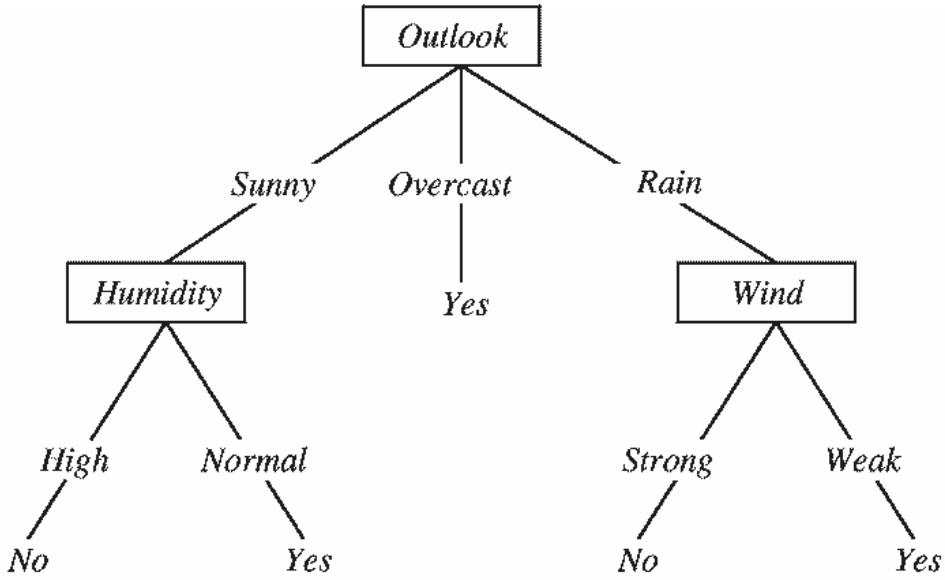

Figure 5.2: Example decision tree for Rain Forecasting [18].

8.4 Stochastic Gradient Descent (SGD)

Stochastic gradient descent is a variant of the gradient descent algorithm. It addresses the inefficiency of normal gradient descent when dealing with large datasets. Instead of using the entire dataset for each iteration, it randomly selects a single training example or small batch to calculate the gradient. It introduces randomness, hence the name "stochastic". I picked this because of its simplicity and efficiency. It is scalable because it can deal with large models. It is also flexible and is efficient with memory.

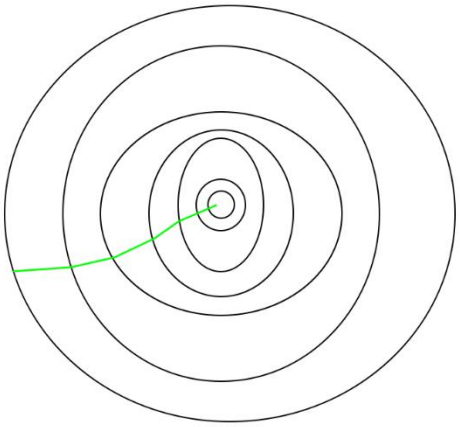

Figure 5.3: Batch gradient descent path [8].

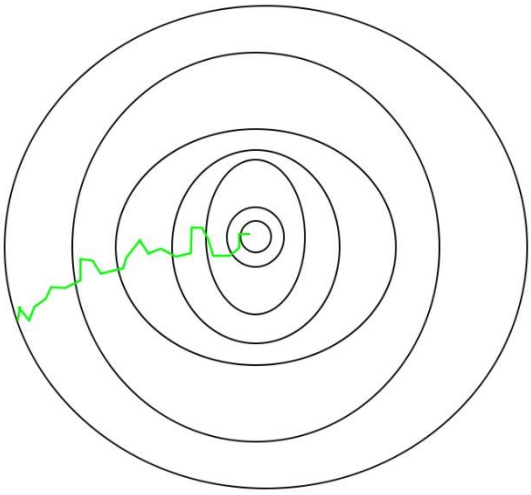

Figure 5.4: Stochastic gradient descent path [8].

8.5 Confusion Matrix

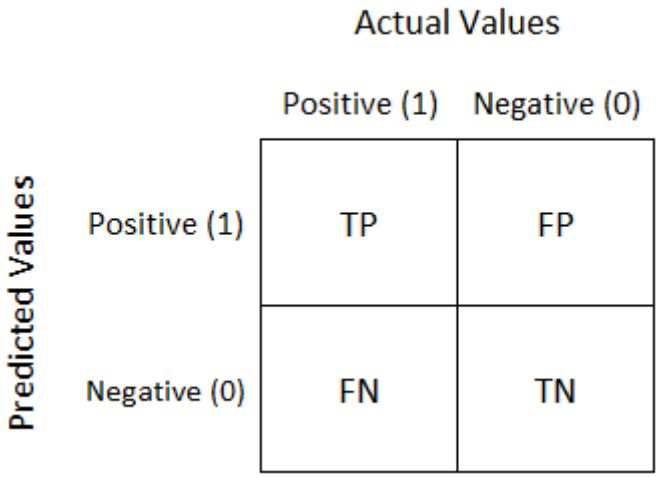

Figure 5.5: Confusion matrix

A confusion matrix is used to show how the model performs and for later calculations. Figure 5 shows an average confusion matrix. True positive (TP) is if the positive value is predicted as positive. False positive (FP) is if the negative value is predicted as positive. False negative (FN) is if the positive value gets predicted as negative. True negative (TN) is if the negative value gets predicted as negative.

8.6 Precision

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

Precision is the percentage of positives out of all the predicted positives. If the model is perfect, the precision is 1. Precision = true positive (TP) divided by true positive (TP) + false positive (FP).

8.7 Recall

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

Recall is the percentage of correct positive predictions out of all the actual positives. If the model is perfect, recall is 1. Recall = true positive (TP) divided by true positive (TP) + false negative (FN).

8.8 F1 Score

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

The F1 score is the harmonic mean of precision and recall. It is not an arithmetic mean. For example, if precision is 0 and recall is 1, the F1 score will be 0. F1 score = 2 * Precision * recall divided by precision + recall.

8.9 Macro Average

$$Precision = \frac{Precision_{ClassA} + Precision_{ClassB} + ... + Precision_{ClassN}}{N}$$

$$Recall = \frac{Recall_{ClassA} + Recall_{ClassB} + ... + Recall_{ClassN}}{N}$$

$$F1Score = \frac{F1Score_{ClassA} + F1Score_{ClassB} + ... + F1Score_{ClassN}}{N}$$

Above are the equations used for the macro average of the calculations used. Scikit-learn provides both the macro average and weighted average. I picked the macro average as it gives an average that lets each class contribute equally no matter the size.

9 Results and Evaluation

For each model, the calculations are shown in a table with its confusion matrix. The calculations will be made into graphs to visualize the data.

9.1 Decision Tree

	Recall	Precision	F1 Score	Time taken (s)
Label	1.00	1.00	1.00	0.62784
Category	1.00	1.00	1.00	0.72746
Specific Class	1.00	1.00	1.00	0.74363

Table 1.1: Results from the Decision Tree model.

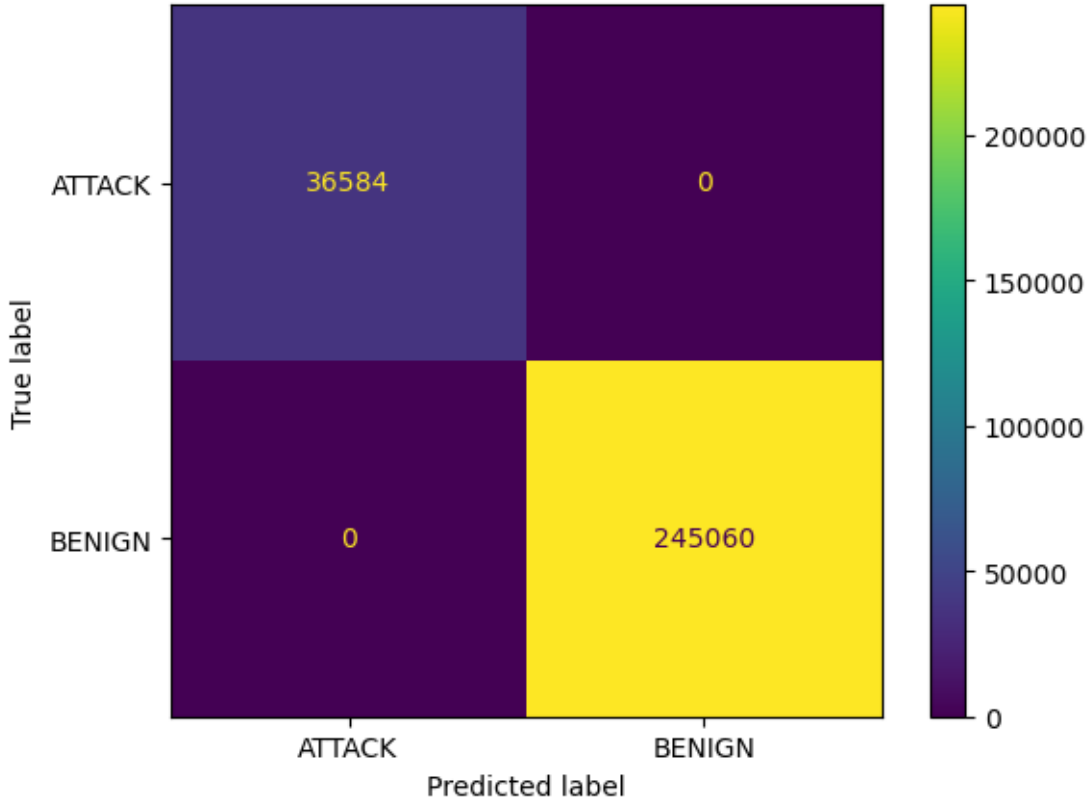

Figure 6.1: Decision Tree confusion matrix for predicting "label".

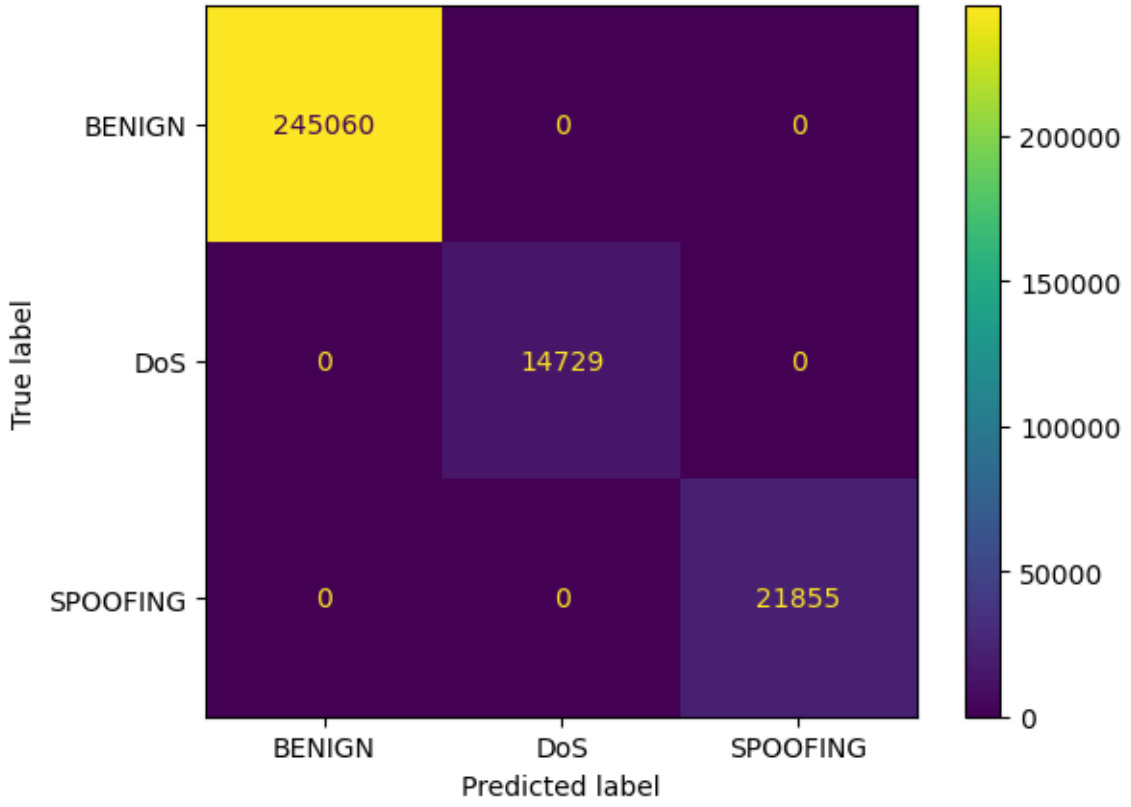

Figure 6.2: Decision Tree confusion matrix for predicting "category".

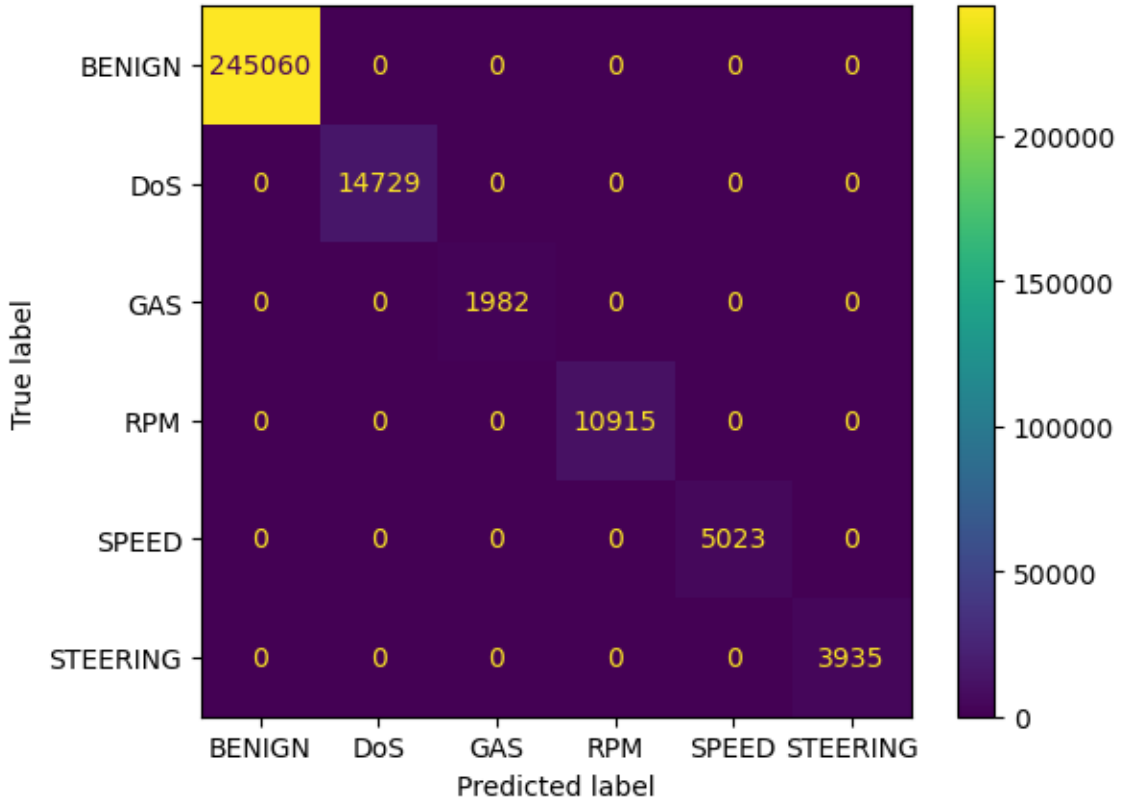

Figure 6.3: Decision Tree confusion matrix for predicting "specific_class".

9.1.1 Analysis

The calculations show that the model is perfect. The confusion matrices show that no malicious signals got through the model. The time taken also is very low so in a real-time scenario, the model should be able to detect the malicious signals before the attacks have done any harm. The dataset does not have complex data for the decision tree to miss.

9.2 Gaussian Naïve Bayes

	Recall	Precision	F1 Score	Time taken (s)
Label	0.69	0.84	0.71	0.22642
Category	0.74	0.87	0.74	0.23770
Specific Class	0.84	0.94	0.87	0.25031

Table 1.2: Results from Gaussian Naïve Bayes model.

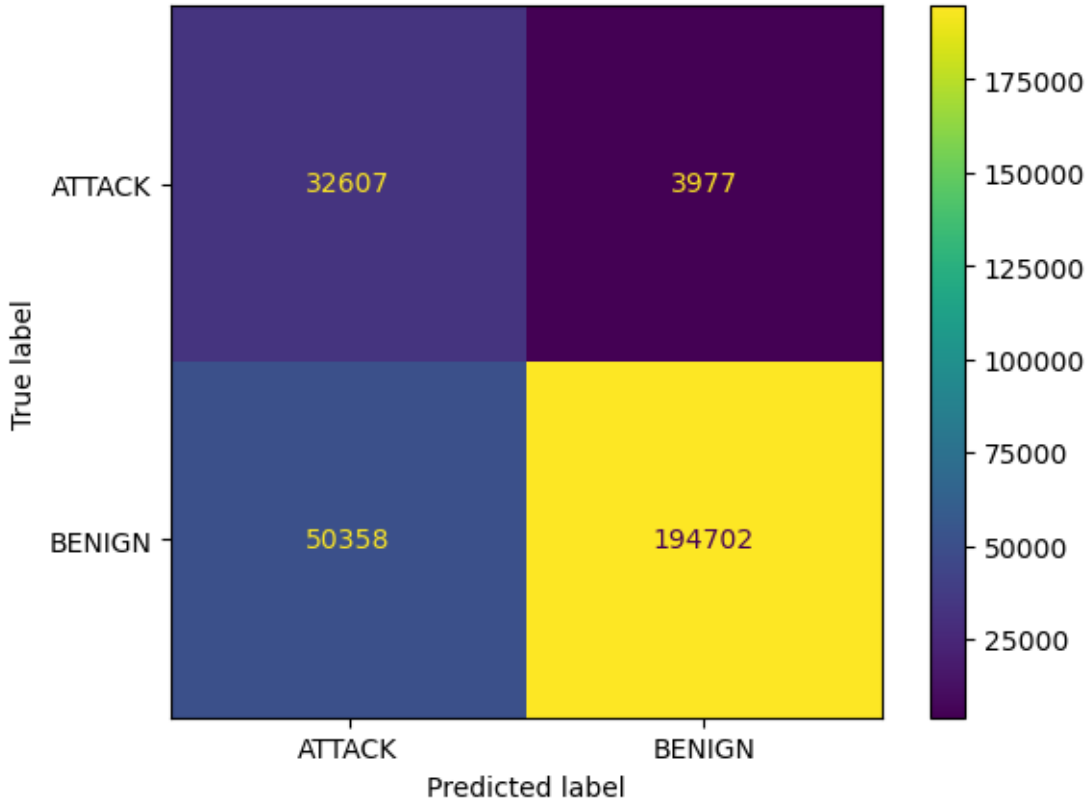

Figure 6.4: Gaussian Naïve Bayes confusion matrix for predicting "label".

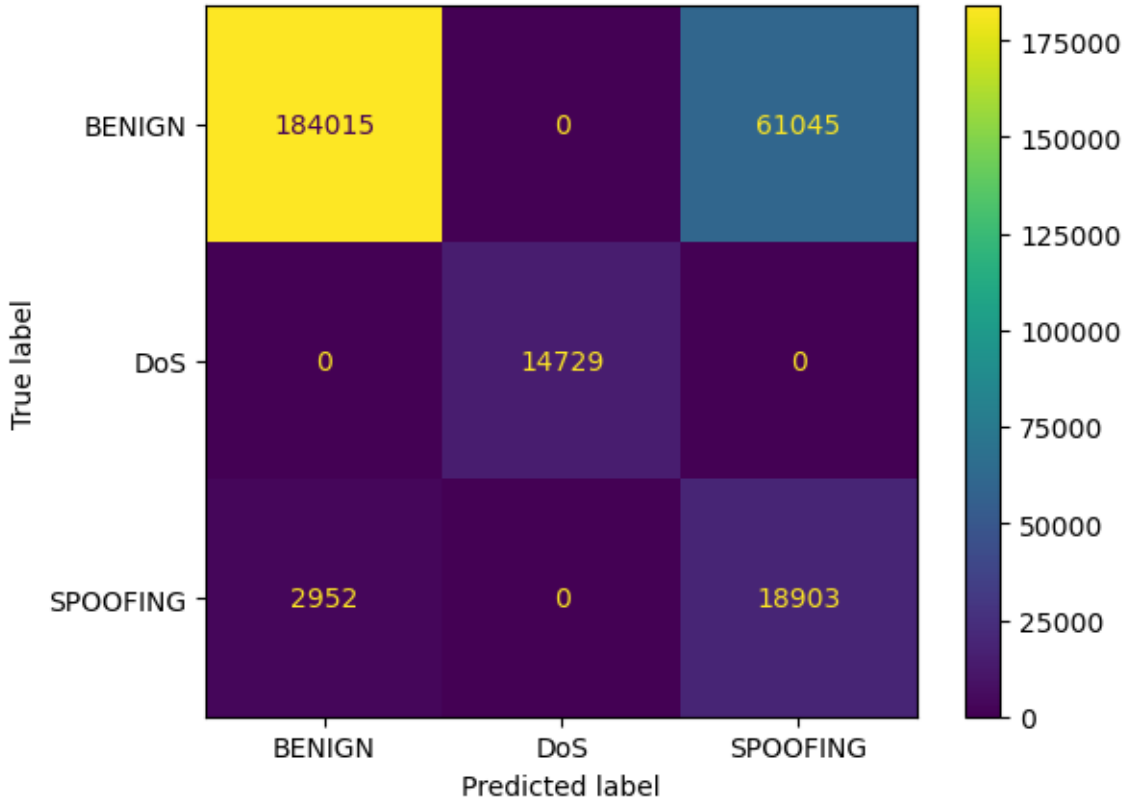

Figure 6.5: Gaussian Naïve Bayes confusion matrix for predicting "category".

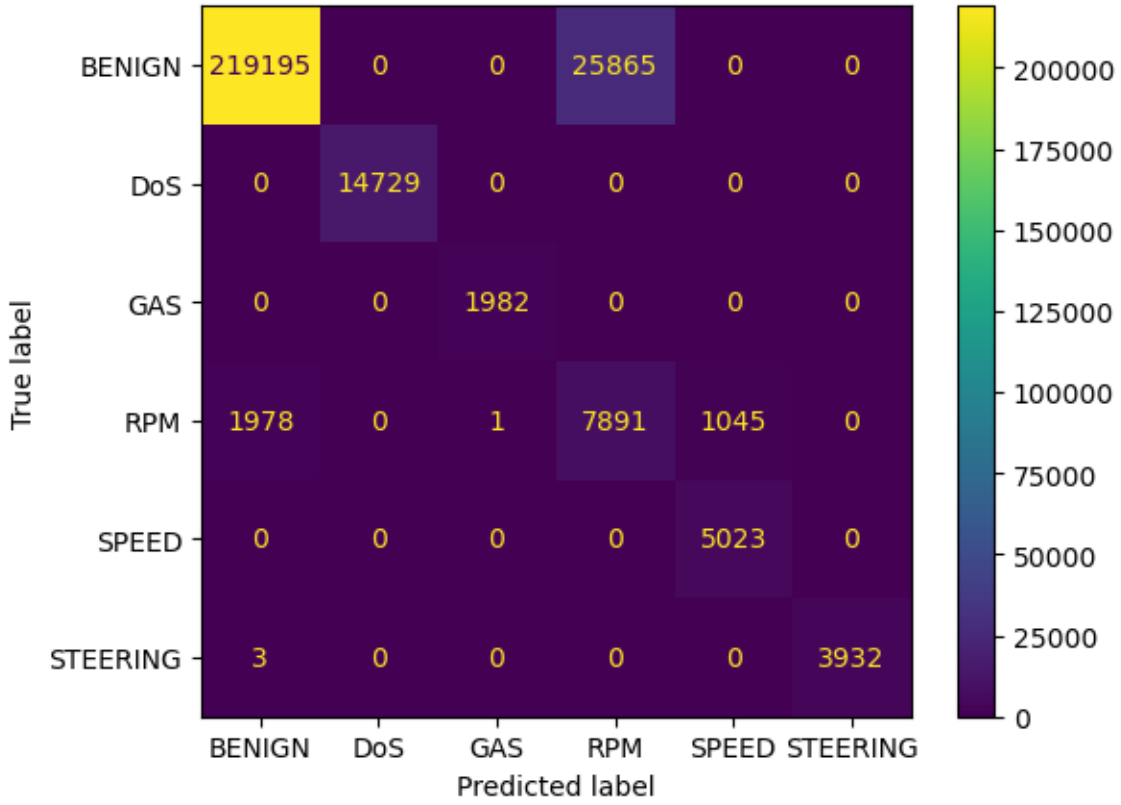

Figure 6.6: Gaussian Naïve Bayes confusion matrix for predicting "specific_class".

9.2.2 Analysis

The calculations provide a variety of results. For labels, the recall is low with good precision. For category the recall is good, and the precision is good. For specific classes, the recall is good, and the precision is very high. The variety shows that the strengths of Gaussian Naïve Bayes is with the specific class. This is because of the variety of classification values that can be picked, and the model can detect the patterns of this at a highly successful rate. The F1 score provides a better representation of the average accuracy of recall and precision. The time taken is very low meaning the model can be run in a real-time scenario. The confusion matrices show that benign data gets detected as an attack often as a false positive. The table shows that overall, the model is more accurate at predicting positive values.

9.3 Multilayer Perceptron

	Recall	Precision	F1 Score	Time taken (s)
Label	1.00	1.00	1.00	25.189
Category	1.00	1.00	1.00	25.857
Specific Class	1.00	1.00	1.00	32.542

Table 1.3: Results from MLP model.

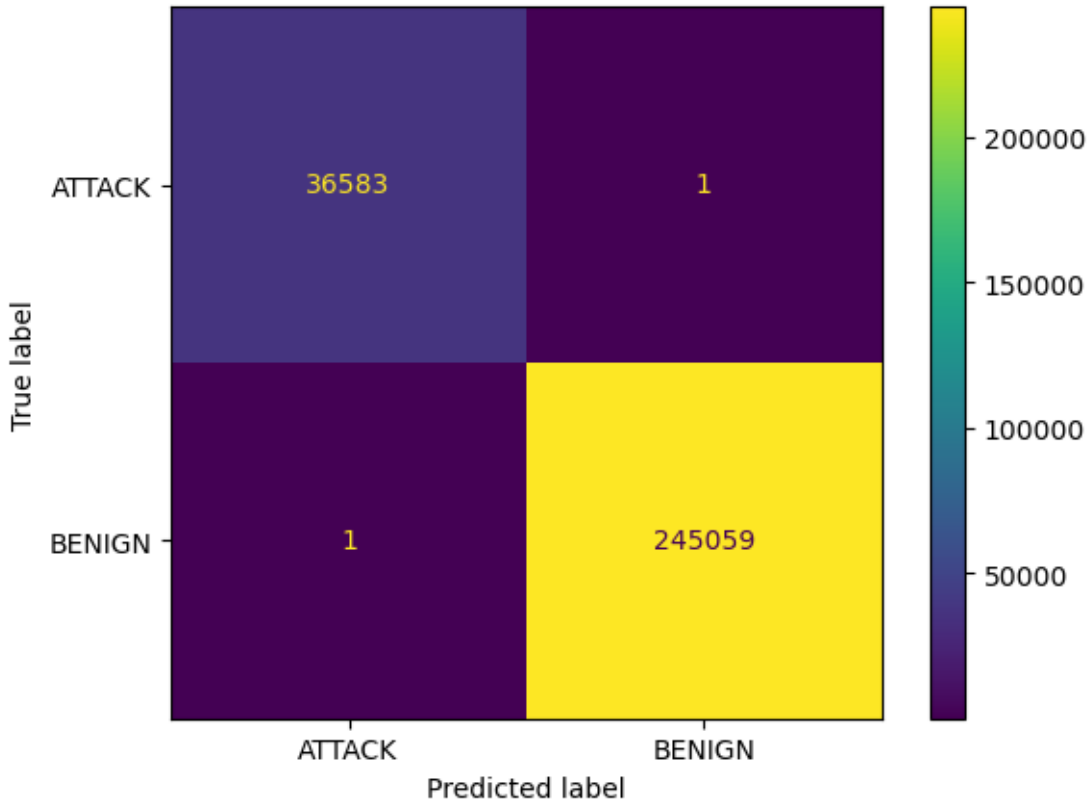

Figure 6.7: Multilayer perceptron confusion matrix for predicting "label".

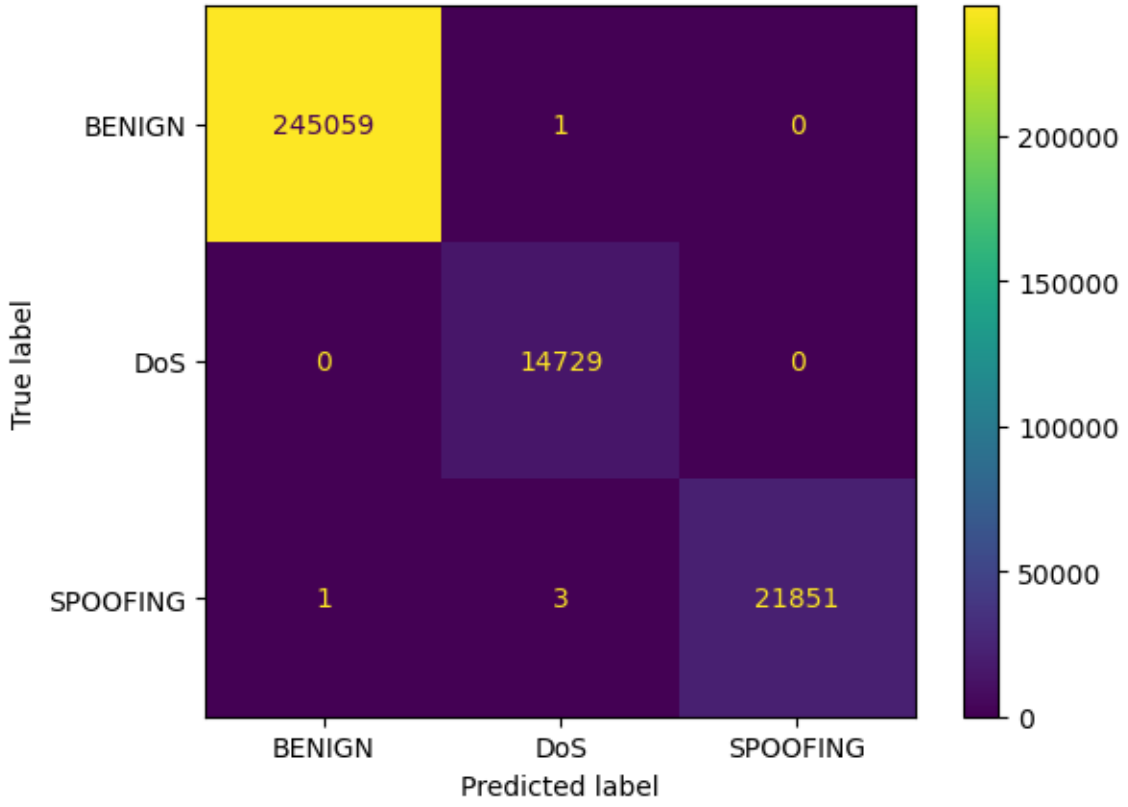

Figure 6.8: Multilayer perceptron confusion matrix for predicting "category".

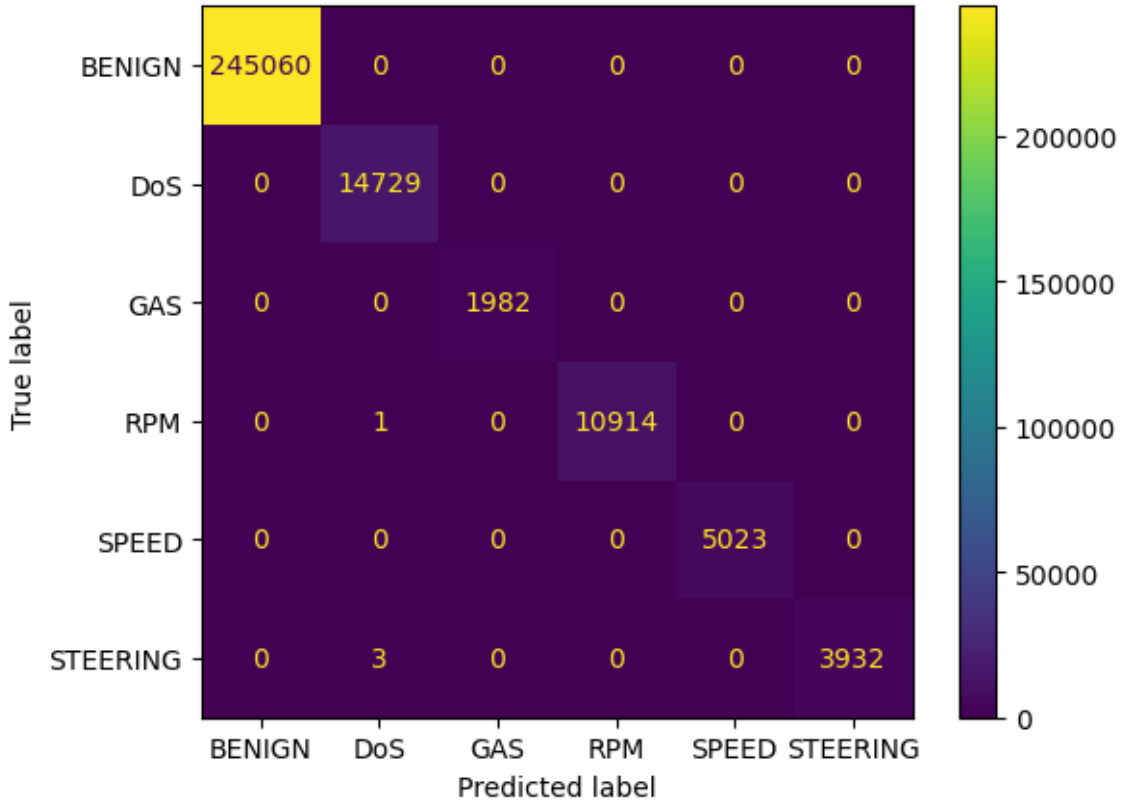

Figure 6.9: Multilayer perceptron confusion matrix for predicting "specific_class".

9.3.3 Parameters

The parameters could have been improved if there was more time. There were time constraints, so not every parameter was tested when optimising them. To further optimise, tests would need to be done on each classified column and test the iterations.

```
nn = neural_network.MLPClassifier(alpha= 0.1, hidden_layer_sizes= 5, learning_rate= 'adaptive',early_stopping=True)
```

Figure 6.10: MLP parameters code.

9.3.4 Analysis

The results show that in terms of the calculations, the model was perfect for detecting cyber-attacks. The confusion matrices show that a few signals slipped through the model as false positives and negatives. A problem with this model is the time taken to run. The time taken could be lowered if the parameters were changed. In a real-time scenario, the time taken to create a model could be a problem when trying to detect cyber-attacks and it may not be quick enough to fit the model before the attacks are finished.

9.4 Stochastic Gradient Descent

	Recall	Precision	F1 Score	Time taken (s)
Label	0.83	0.79	0.81	1.5022
Category	0.72	0.78	0.74	3.8517
Specific Class	0.55	0.48	0.47	6.8152

Table 1.4: Results from Stochastic gradient descent model.

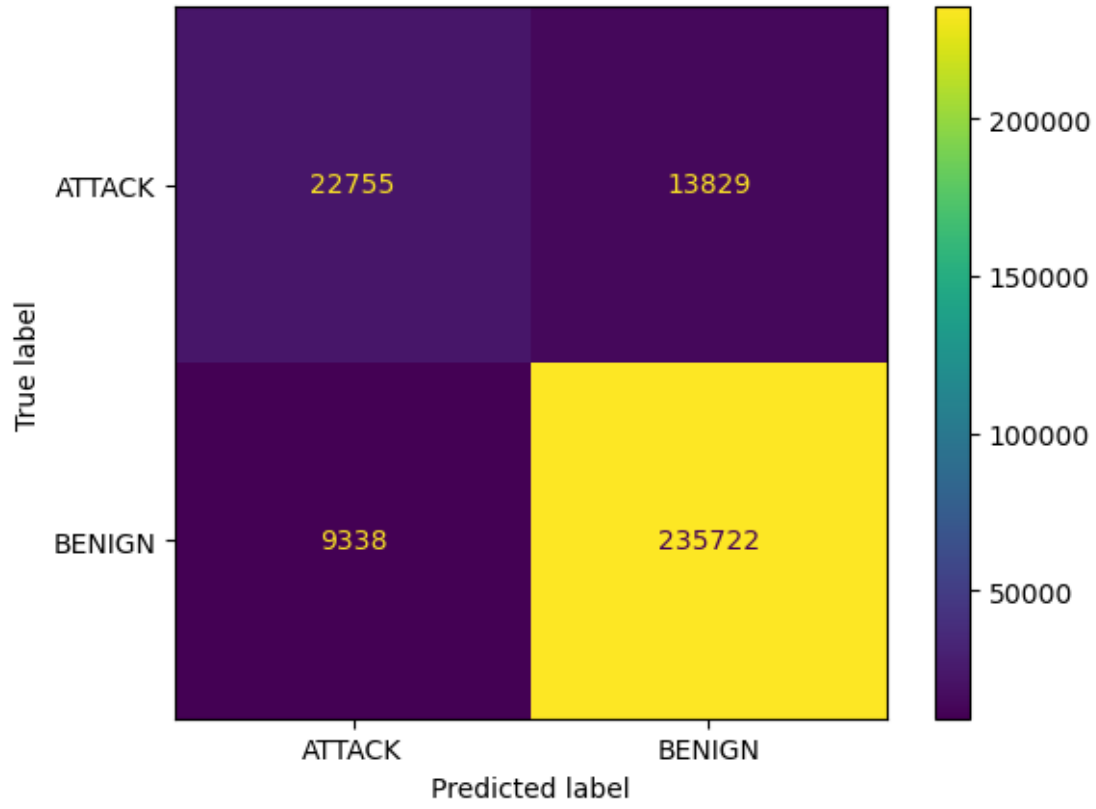

Figure 6.11: Stochastic gradient descent confusion matrix for predicting "label".

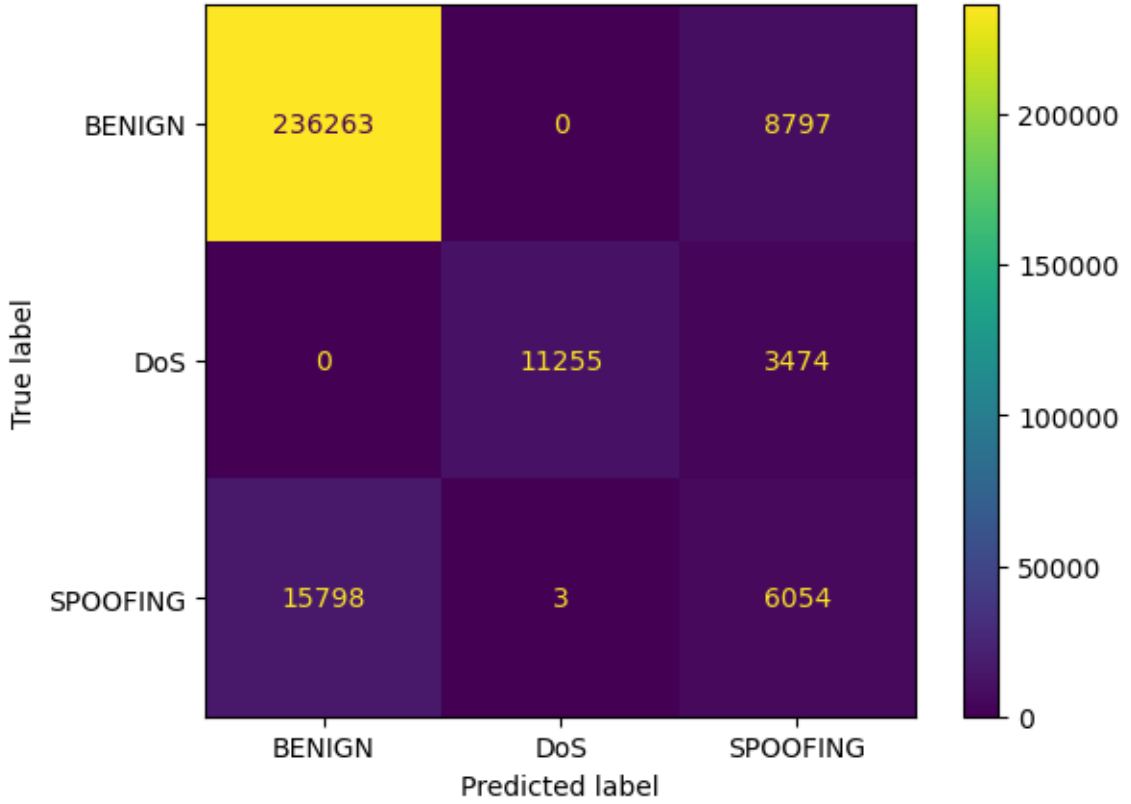

Figure 6.12: Stochastic gradient descent confusion matrix for predicting "Category".

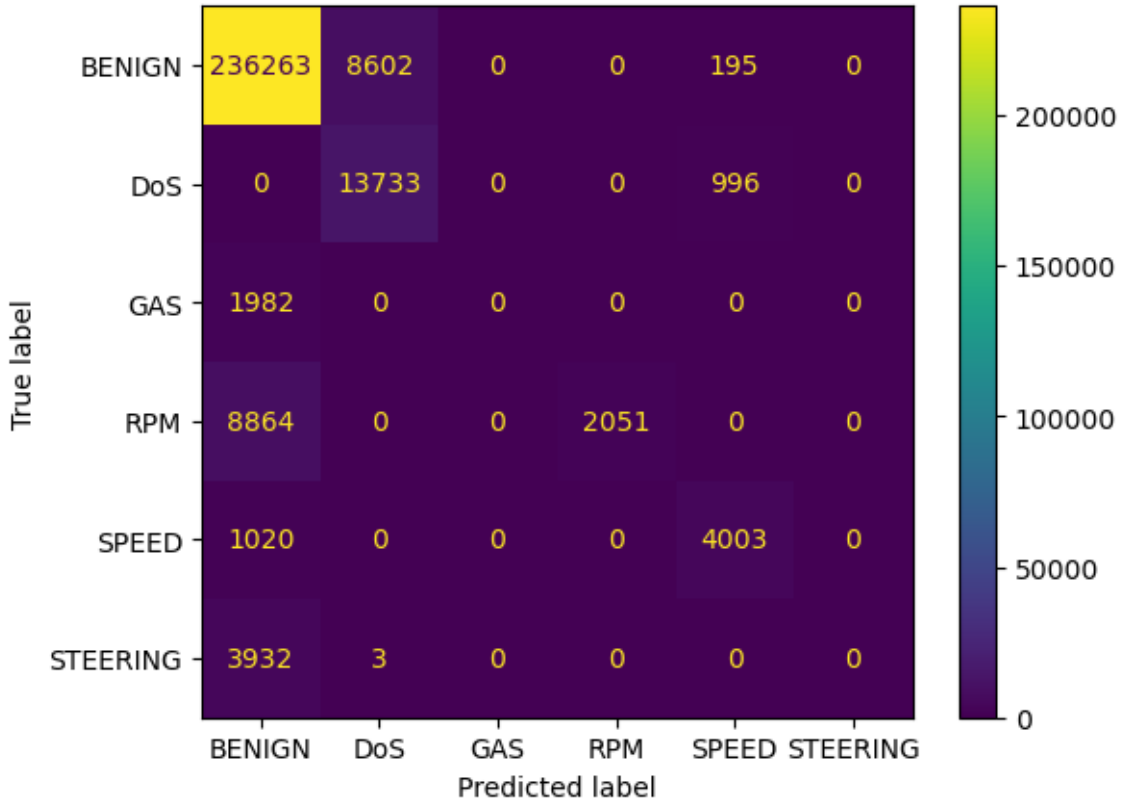

Figure 6.13: Stochastic gradient descent confusion matrix predicting "specific_class".

9.4.5 Parameters

```
clf = SGDClassifier(loss="hinge", penalty="l2", early_stopping=True)
```

Figure 6.14: SGD parameters code.

9.4.6 Analysis

The calculations show good results for label and category. The label has a high recall and precision score and the same with the category score. However, specific class analysis shows that the model cannot handle the high amount of classification values or the patterns in the data. This could be because of the parameters used but I did not have the time frame to try this. The F1 score provides an average of the recall and prevision and shows that the model is best at analysing the label. The time taken is sufficient for the label and category analysis but slightly high for specific classes. It can be possible for the model to be used in a real-time scenario.

9.5 Graphs

In the graphs, DT is a decision tree, GNB is a Gaussian naïve Bayes, MLP is a multilayer perceptron and SGD is stochastic gradient descent.

9.5.7 Label

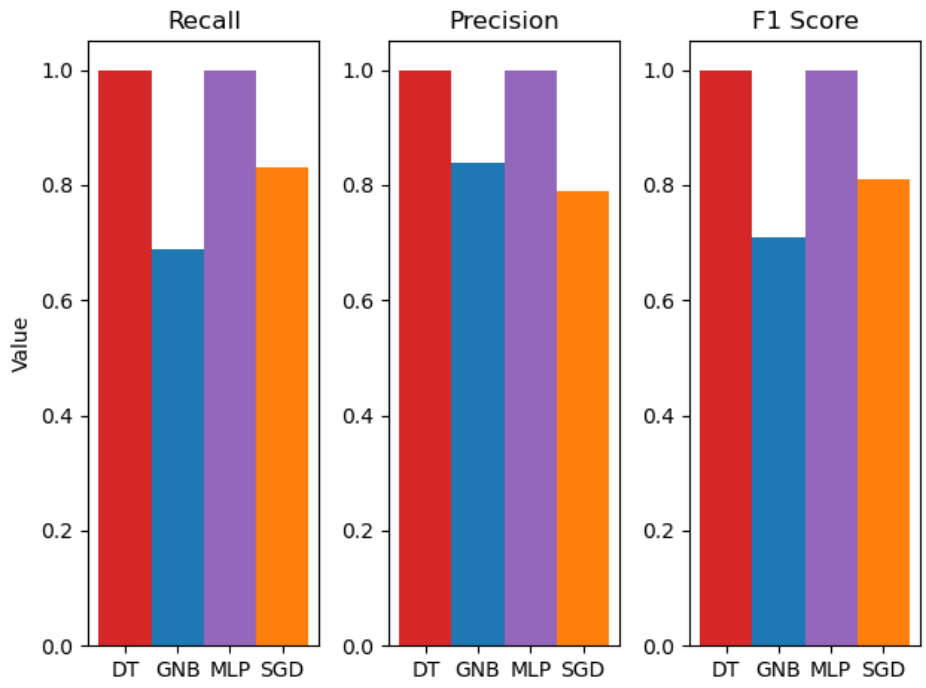

Figure 6.15: Graph showing "label" classification calculations.

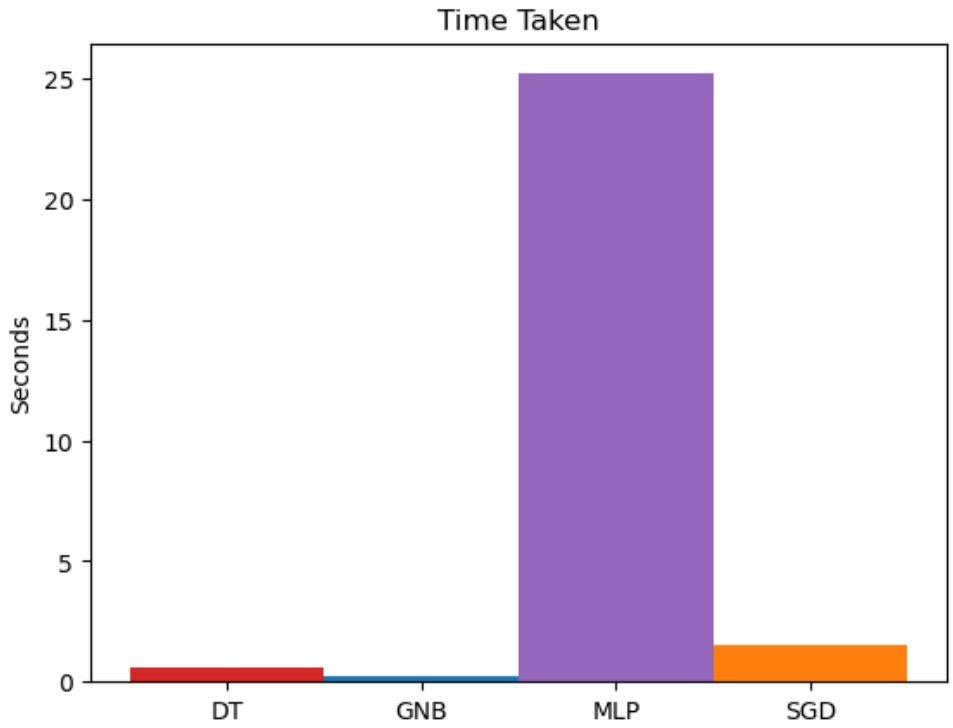

Figure 6.16: Graph showing Time Taken to run models for "Label".

9.5.8 Category

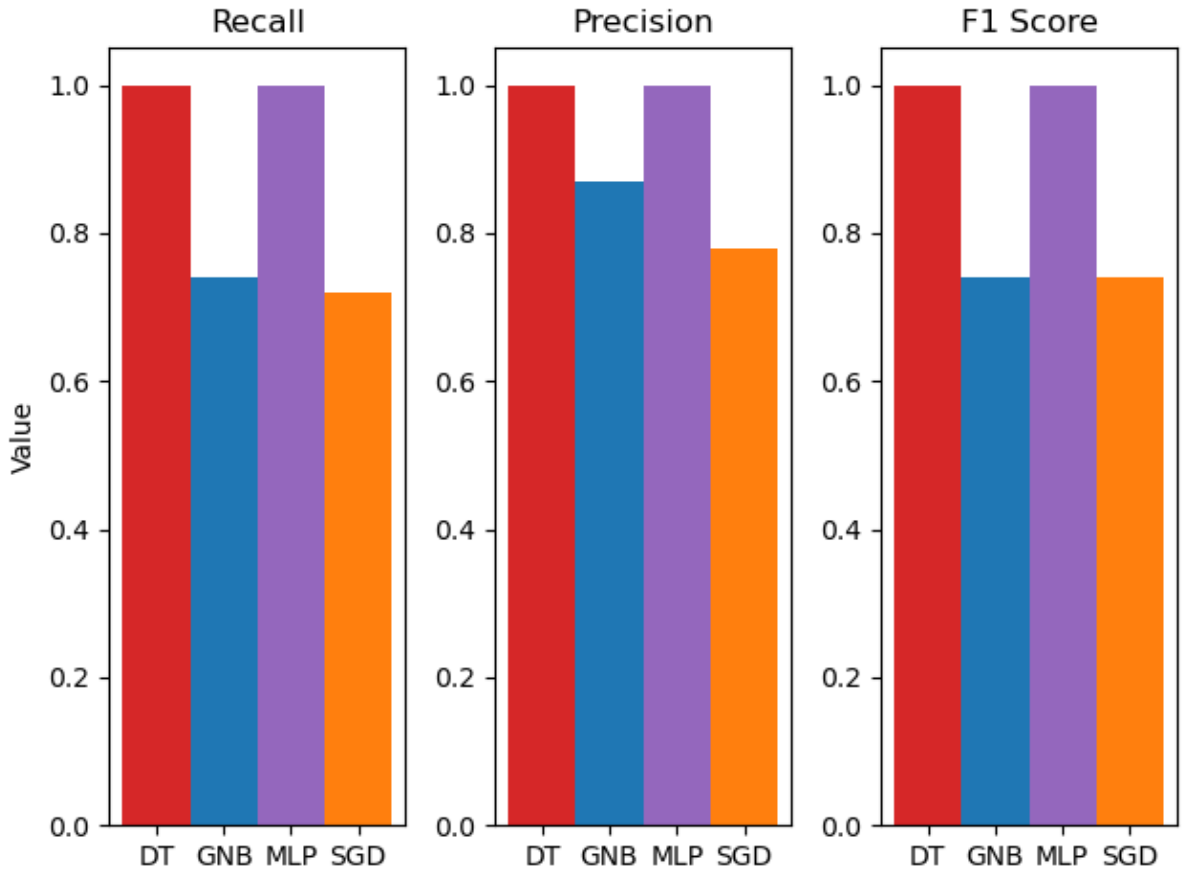

Figure 6.17: Graph showing "category" classification calculations.

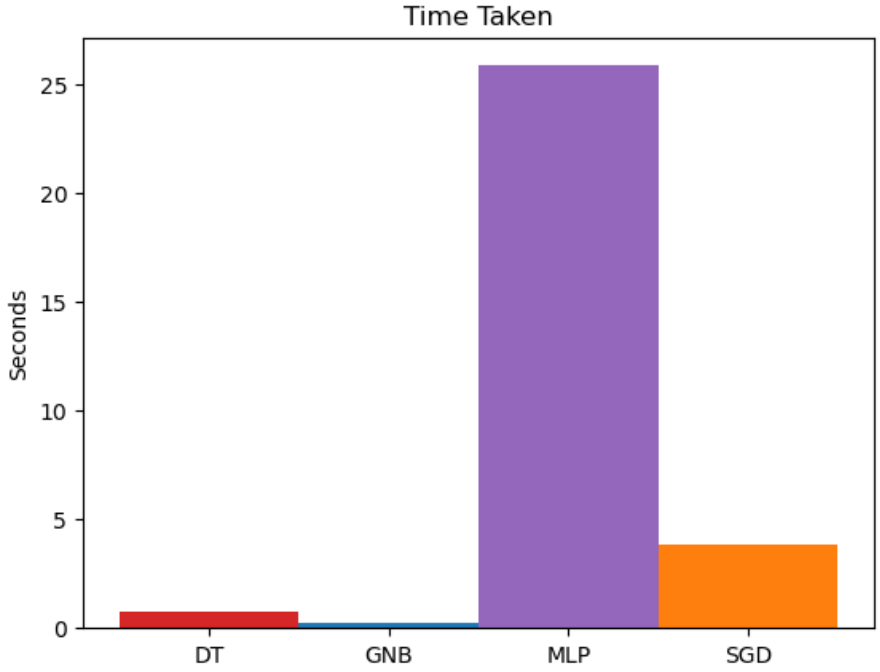

Figure 6.18: Graph showing Time taken to run models for "Category".

9.5.9 Specific Class

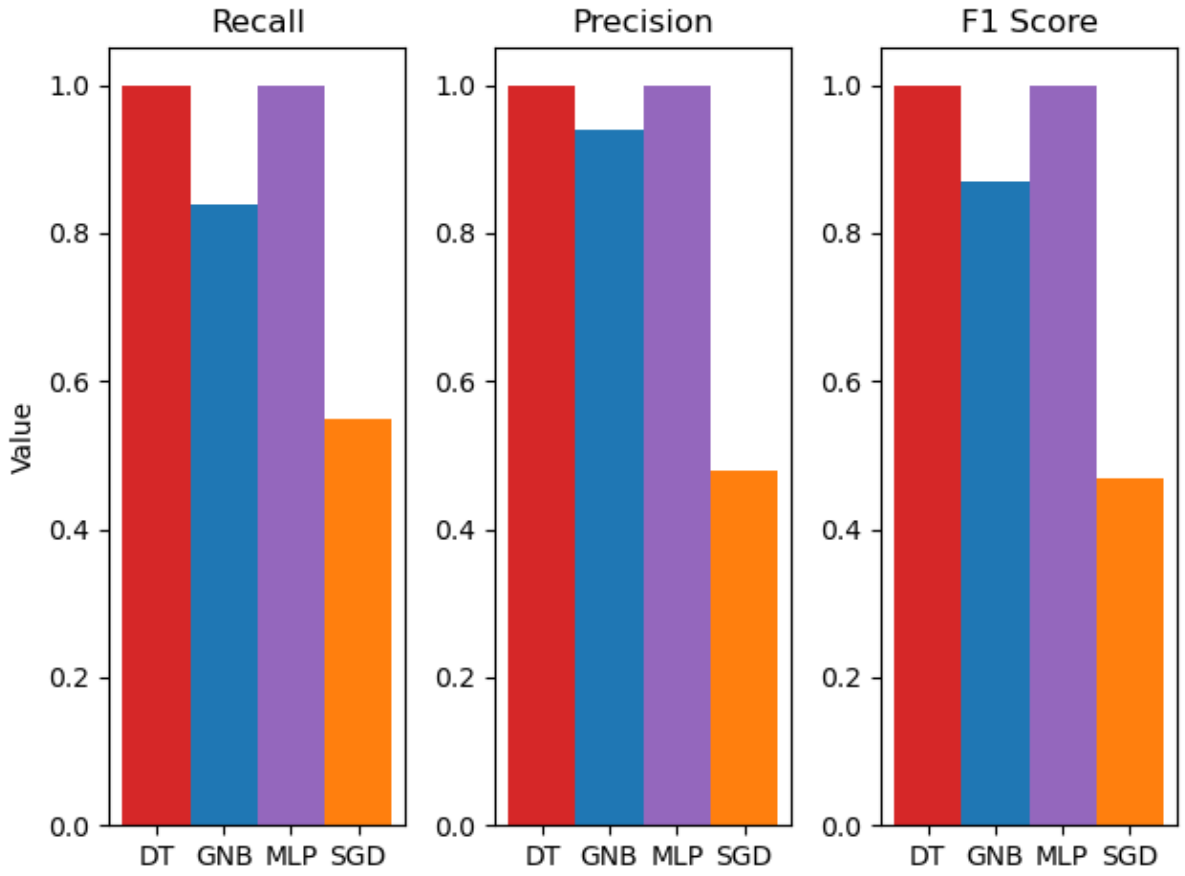

Figure 6.19: Graph showing "specific_class" classification calculations.

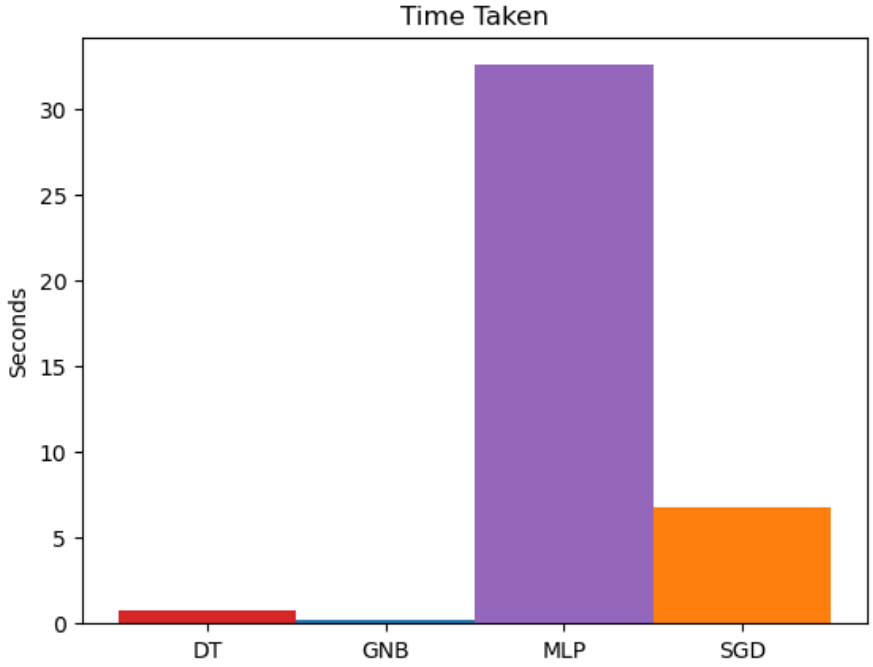

Figure 6.20: Graph showing Time Taken to run models for "specific_class".

9.6 Evaluation

Figures 6.15 – 6.20 show the comparisons between the models. For the labels, the best-performing models were decision tree and multilayer perceptron which both scored a perfect 1.0. However, when comparing the time taken, multilayer perceptron takes significantly more time to run, over 20 seconds more time. The time-taken tests were executed to show if the model could be used in a real-time scenario. If the time taken is extremely high, it can cause problems and let attacks through before it's too late. Especially considering, the time taken for each classification combined. Multilayer perceptron takes around 70 seconds to complete all 3 classifications. Compared to decision trees which take a few seconds. Gaussian naïve Bayes was the quickest model. With it being the quickest, it sacrifices performance, in some tests, it does the worst out of the other models. The worst model overall is stochastic gradient descent. The time taken was average and the performance overall was not sufficient for detecting the cyber-attacks. From this information, decision trees appear to perform the best. However, if the parameters were optimised better, the results could change. If these rigorous tests were carried out, I believe stochastic gradient descent would perform better and multilayer perceptron may cut the time taken.

Figures 6.1 - 6.9 and 6.11 – 6.13 show the confusion matrices for each model. Excluding multilayer perceptron and decision trees, the models had trouble detecting spoofing attacks. Overall, DoS attacks were discovered with very high accuracy, the true positives were always a high percentage. Spoofing attacks got through the models because of the way the attack works. Multilayer perceptron and decision trees could discover these attacks because of the way the models work.

I am confident in these results but the perfect results from decision trees with the instant time taken is concerning. To gain further confidence, I would run the optimiser for longer on stochastic gradient descent and multilayer perceptron. I would also test decision trees on another similar dataset to see if the results were an anomaly.

The results show an accurate representation of each model, showing their strengths and weaknesses, and giving enough information to differentiate which model is better or worse. The results could be improved by optimising the parameters. I made the mistake of not leaving enough time to run the optimisers. A way to improve my results would be to add a multiclass classifier. This would allow for another way to analyse the data, adding some variety. Another way to improve is to try more machine learning models and test more databases to see if the models work on all types of data.

The project overall was successful. Scikit-learn was the perfect module to run the models and JupyterLab was useful in saving time. Python was a good choice for programming language because whenever I got stuck, I could troubleshoot as it is a popular choice. The methodology/implementation was appropriate. The models chosen in the implementation gave a variety of results and had no errors when running.

10 Conclusion and Future Work

This project aimed to find the best-performing AI model on the dataset to detect cyber-attacks. The research and implementation found that the decision tree model performed the best because of its speed and high accuracy. I found that stochastic gradient descent and Gaussian naïve bayes performed decently and were quick, but not as good as decision trees. Multilayer perceptron performed as well as decision trees, but the time taken was extremely high. This project will provide information for researchers on the performance of models on the database and give a possible direction in choosing models.

All the objectives stated in the initial plan were completed. Machine learning and deep learning were used in the models. Graphs and tables were created to visualise the data for easy comparisons. I also found a good dataset that had multiple types of attacks and types of ECUs.

The project began with a brief overview of the CAN bus and the types of attacks that can happen on it. Next, the project goes through the data representation and the meaning of the data in the dataset (CICIoV2024). This leads to the implementation which explains the code used and what libraries are used to implement the code. The results section portrays the data in tables and graphs to be compared and analysed. Finally, the evaluation completes the aim of analysing which model performed best at discovering cyber-attacks.

Future work I am interested in doing is getting the CAN bus data myself and testing ways to interpret the data. When retrieving the data myself, it can let me create my dataset. Creating a new dataset can help researchers with more data to analyse. It would be desirable to get data on a vehicle that has not been used before. This would be a good experience and would help my understanding of the CAN bus. Future work with the chosen dataset would be to test out more models and optimise the parameters properly for the models that need it. Also, testing out a multiclass classifier would be useful to compare the single classifier. Creating a new model that is not in scikit-learn would be a desirable aim for future work.

11 Reflection

This project was a great experience this term. There were problems with sickness and my schedule when working on the project, but I learnt a lot about machine learning and did the project to the best of my ability and knowledge. The skills I learnt can be used in the future in the workplace and future projects. A project that I have in mind that can use machine learning is to do with football such as an “xG calculator”.

The project was important for me to learn about the problems I have with completing long-term work. It was hard for me to do work consistently over the term while following the schedule strictly. Bad habits like procrastination caused me to leave a lot of work to the end but it helped me realise the importance of setting a structure to the work. I had to try and break the habit while working on the project. Procrastination is a big problem I’ve had over the university years but after this, I think I have improved in spending more time on my work.

This project taught me a lot about machine learning and mathematics. Coding the models in Python has helped me try a new type of coding in Python that I haven't done before. This has motivated me to try a new project that I mentioned above. Learning about the models has improved my mathematics knowledge, improved my problem-solving skills and my understanding of equations. If I could go back, I would spend more time trying other datasets to improve this project.

In terms of time management, I started to procrastinate and leave the work to the side, and this made me miss meetings. I regret not doing meetings consistently. However, when I did the meetings, it helped me move forward with my project every time. If I went to meetings more, it would've improved my project and sped up the working process.

The project helped me learn about new libraries like Scikit-learn which made coding the models easier and more intuitive. I also learnt about "MachineLearningMastery" which had all the lessons about machine learning in Python in one place and helped me easily order my learning.

The project was my first time trying a project and helped me realise how fun learning new things is and have it combined into a big project to be proud of. I have not had much work experience with coding, so this experience has helped me have a career opportunity in machine learning. I had to take an extenuating circumstance because of my illness but if I had followed the schedule and worked consistently, I would not have needed an extra week.

12 References

[1] AIML (2024) *What is a multilayer perceptron (MLP) or a Feedforward Neural Network (FNN)?, AIML.com*. Available at: https://aiml.com/what-is-a-multilayer-perceptron-mlp/

[2] Bari, B.S., Yelamarthi, K. and Ghafoor, S. (2023) *Intrusion detection in Vehicle Controller Area Network (CAN) bus using Machine Learning: A Comparative Performance Study, Sensors (Basel, Switzerland)*. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10099193/

[3] Blum, B. (2022) *Cyberattacks on cars increased 225% in the last three years, ISRAEL21c*. Available at: https://www.israel21c.org/cyberattacks-on-cars-increased-225-in-last-three-years/

[4] Carlos Pinto Neto, E. *et al.* (2024) *CICIOV2024:advancing realistic IDS approaches against DOS and Spoofing Attack in iov can bus, SSRN*. Available at: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4733521

[5] EmbedLogic (2022) *Standard can vs extended can, Embedclogic*. Available at: https://embedclogic.com/can-protocol/standard-can-vs-extended-can-protocol-frame/

[6] EmbedLogic (2023) *Can physical layer, Embedclogic*. Available at: https://embedclogic.com/can-protocol/can-protocol-physical-layer/

[7] Gazdaga, A., Ferenczib, C. and Buttyán, L. (2020) *Development of a man-in-the-Middle Attack Device for ...* Available at: http://www.hit.bme.hu/~buttyan/publications/GazdagFB2020citds.pdf

[8] GeeksforGeeks (2024) *ML: Stochastic gradient descent (SGD), GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/ (Accessed: 17 May 2024).

[9] Marchetti, M. and Stabili, D. (2019) *Read: Reverse Engineering of Automotive Data Frames | IEEE Journals & Magazine | IEEE Xplore*. Available at: https://ieeexplore.ieee.org/document/8466914/

[10] Neto, E.C.P. *et al.* (2024) *CICIoV2024: Advancing Realistic IDS Approaches against DoS and Spoofing Attack in IoV CAN bus, E. C. P. Neto, H. Taslimasa, S. Dadkhah, S. Iqbal, P. Xiong, T. Rahmanb, and A. A. Ghorbani*. Available at: https://www.unb.ca/cic/datasets/iov-dataset-2024.html

[11] rocketjosh (2020) *Macchina M2, GitHub*. Available at: https://github.com/macchina/m2-hardware

[12] stabili, dario and marchetti, mirco (2019) *Detection of Missing CAN Messages through Inter-Arrival Time Analysis*. Available at: https://ieeexplore.ieee.org/document/8891068

[13] UK, L. (2024) *Lexus UK statement on Vehicle Theft*, *Lexus UK Magazine*. Available at: https://mag.lexus.co.uk/lexus-uk-statement-on-vehicle-theft/#:~:text=Once%20connected%20to%20the%20vehicle's,turn%20the%20vehicle's%20ignition%20ON

[14] UK, T. (2024) *Toyota GB statement on Vehicle Theft*, *Toyota UK Magazine*. Available at: https://mag.toyota.co.uk/toyota-gb-statement-on-vehicle-theft/

[15] Wang, C. *et al.* (2018) *A distributed anomaly detection system for in-vehicle network using HTM | IEEE Journals & Magazine | IEEE Xplore*. Available at: https://ieeexplore.ieee.org/abstract/document/8274979/

[16] wewillnotbefobbedoff (2024) *Canbus theft scandal*, *Lexus Owners Club*. Available at: https://www.lexusownersclub.co.uk/forum/topic/145825-canbus-theft-scandal/

[17] wei, P. *et al.* (2022) *A novel intrusion detection model for the CAN bus packet of in-vehicle network based on attention mechanism and autoencoder*, *Digital Communications and Networks*. Available at: https://www.sciencedirect.com/science/article/pii/S2352864822000700

[18] Yadav, P. (2019) *Decision tree in machine learning*, *Medium*. Available at: https://towardsdatascience.com/decision-tree-in-machine-learning-e380942a4c96 (Accessed: 17 May 2024).

[19] Yang, O. (2023) *How to get away with car theft: Unveiling The dark side of the can bus*, *VicOne*. Available at: https://vicone.com/blog/how-to-get-away-with-car-theft-unveiling-the-dark-side-of-the-can-bus

[20] Yang, Y., Duan, Z. and Tehranipoor, M. (2020) *Identify a spoofing attack on an in-vehicle can bus based on the deep features of an ECU fingerprint signal*, *MDPI*. Available at: https://www.mdpi.com/2624-6511/3/1/2