CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Physical fault detection on the Controller Area Network

An investigation into the feasibility of detecting physical faults on the CAN bus with machine learning

Master's thesis in Computer science and engineering

Pontus Engström
Rikard Roos

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Master's thesis 2026

Physical fault detection on the Controller Area Network

An investigation into the feasibility of detecting physical faults on the CAN bus with machine learning

PONTUS ENGSTRÖM

RIKARD ROOS

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

Physical fault detection on the Controller Area Network
An investigation into the feasibility of detecting physical faults on the CAN bus with machine learning
Pontus Engström
Rikard Roos

© Pontus Engström, Rikard Roos, 2026.

Supervisor: Carl-Johan Seger, Computer Science and Engineering
Advisor: Kent Lennartsson, Kvaser AB
Examiner: Carl-Johan Seger, Computer Science and Engineering

Master's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2026

Physical fault detection on the Controller Area Network
An investigation into the feasibility of detecting physical faults on the CAN bus with machine learning

Pontus Engström
Rikard Roos
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Detecting physical faults on a Controller Area Network is a complex task due to unpredictable signal behavior under normal circumstances, and signal behavior unique to the configuration of the specific network and its components. Furthermore, physical faults can lead to disruptive signal behavior, which are not always detected by the protocol's error handling. If it does not activate the fault handling of the protocol, the disruptive behavior is only seen on the physical layer, potentially resembling unpredictable behavior under normal states. Such physical faults can imply a marginal system, leading to unpredictable signal behavior over time. If the fault is detected by the protocol, the source of the fault is not detected, and further investigation of how and where the fault occurred is required.

Traditional fault detection methods include manual analysis, which requires time and expertise. This thesis investigates the feasibility of applying machine learning methods to achieve physical fault detection on a CAN bus. The aim was to determine if, and to which extent, this could be achieved. A physical test bench was configured to enable CAN traffic under normal and faulty configurations. By calculating the offsets of the expected position of recessive signal edges, minimal data was used to distinguish physical faults from normal conditions. A range of machine learning classifiers were trained and evaluated to exhaust the possibilities of achieving high accuracy on this task.

The problem was narrowed down to detect termination faults, and in turn, to localize the faults. The results indicates that it is feasible to differentiate termination faults with data-driven methods. In particular, Gradient Boosting, a decision tree ensemble architecture, achieved a 99.98% accuracy on this task. However, localizing the faults was best achieved with Gradient Boosting with an accuracy of 93.74 %. Two distinct fault classes were of too similar shape to be separable to a perfect degree.

The main limitation of this study included the lack of generalization capabilities. The signal data proved to be tightly connected to the physical setup of the CAN bus, and applying a model to different network than it was trained on is very likely to fail. Furthermore, the data was generated under stable, indoors conditions, with high-quality hardware. A further investigation of the real-world practical use is required.

Keywords: Controller Area Network, CAN bus, physical fault detection, machine learning, classification

Acknowledgements

First and foremost, we would like to express our gratitude towards Kvaser and the Kvaser team. Without you, this study would not have been initiated, and without your help, it would not have been completed. The knowledge we have gained, and the contribution this study brings to the field of CAN fault detection, was achieved under a great deal of influence from the knowledgeable and helpful people of Kvaser. Furthermore, we want to mention our gratitude to Kvaser for engaging in the study and providing the technical tools required to realize it. We want to direct a special thank you to Kent Lennartsson, whom without this thesis would be of a great lesser quality. Thank you for sharing your technical expertise and for guiding us through the complexity of CAN, and for doing so with a smile on your face.

We also want to express our sincerest gratitude towards our supervisor and examiner Carl-Johan Seger, whom with engagement and constructive criticism has supported us throughout the study. Thank you for providing us with assistance and encouragement on a weekly basis. In the details as well as the broader picture, the value of your feedback can not be exaggerated.

Pontus Engström, Rikard Roos, Gothenburg, 2026-05-25

Contents

List of Figures

List of Tables

1

Introduction

Vehicles have experienced a radical shift the last decades, transitioning from being predominantly mechanical to becoming digitally controlled. Currently, the Controller Area Network (CAN or CAN bus) forms the backbone of in-vehicle communication, linking electronic control units (ECU) responsible for safety and performance [1]. CAN has the benefit of maintaining communication under harsh environmental conditions and reducing the amount of cables that otherwise would be needed to connect units in the vehicle, since multiple devices can run on the same network.

The number of ECU:s in high-end vehicles grew from less than ten in the late 1980's to over 100 in 2008 [2]. While the automotive industry indicates a trend of implementing centralized systems able to maintain complex software such as Advanced Driving Assistance Systems (ADAS), the number of ECU:s in modern vehicles is similar to 2008 [3]. Furthermore, complex centralized architectures such as ADAS would still rely on networks such as the CAN bus to communicate with sensors and actuators. Consequently, the CAN bus remains important for ensuring reliable data exchange within vehicles. Examples of ECU:s in modern vehicles include safety critical systems such as the anti-lock braking system, as well as convenience systems such as the air conditioning unit [4]. Due to the potentially critical safety risk of a failing CAN bus, it is of great importance that faults are detected and corrected.

The physical layer of the CAN bus is susceptible to issues such as electromagnetic interference, poor termination, bad grounding and defective nodes. While such faults often produce recognizable errors, the cause of the issue is not discovered. Therefore, disturbances on the physical layer leads to faulty and unpredictable behavior on the CAN bus, which can be very difficult to identify and correct.

Detecting physical faults can be achieved by connecting an oscilloscope to a CAN bus and manually studying the signal waveform [5]. The downside with this type of diagnosis is the limited ability to provide real-time detection and the requirement for professional domain knowledge. Another approach is to automate fault detection using data-driven methods. Recent advances in machine learning (ML) has shown strong potential for anomaly detection and fault diagnosis in different mechatronic systems, detecting patterns that humans would need both time and expertise to recognize [6].

High-resolution signal analysis, synchronized with the CAN protocol, enables correlation of signal properties and bit-level data. This technology opens up new diagnostic

possibilities for identifying and classifying physical faults. Furthermore, small deviations in signal behavior can differentiate a perfect system from a good system, and a good system from a marginal system. Identifying the quality level of the network, even if it is currently working, enables proactive measures.

This thesis investigates the feasibility of applying machine learning techniques to detect and classify physical-layer faults on a CAN bus using high-resolution signal data. The goal was to determine whether, and to which extent, an ML-based diagnostic system could recognize physical faults on a CAN bus.

1.1 Problem Statement

Despite its robustness, the CAN bus is susceptible to many different physical errors that can be difficult to remedy since they are not necessarily detected by the protocol, but still disturb or disrupt communication. Detecting these errors is often done manually with the help of special equipment and domain knowledge.

The objectives of this thesis are the following.

- Configure a physical test environment and gather signal transition data from a variety of physical CAN bus configurations.
- Process and curate the data to extract the offsets from expected timings of signal transitions and filter out unwanted information.
- Apply machine learning techniques to identify which physical faults are possible to separate.
- Evaluate the success and limitations of the procedure.

1.2 Research Question

This thesis will attempt to answer the following research question.

Is it possible to detect physical faults on a CAN bus using machine learning?

The complexity of this task lies in drawing concrete conclusions of faulty bus states based on physical data which deviates under normal conditions. For example, it is required to distinguish faults from signal distortions that occur during normal circumstances.

1.3 Scope

The scope of the thesis was kept narrow to go far with one aspect of the problem, rather than touch lightly on several. First, this is an applied machine learning study, and the aim was not to develop a new state of the art model specialized for CAN buses. Instead, the thesis investigates the performance of certain machine learning architectures on excerpts of CAN bus data sampled under faulty configurations.

Second, one physical fault was isolated. Incorrect termination was chosen as it commonly occurs in real scenarios, and as it is often not detected by the CAN protocol. Furthermore, many physical aspects affecting the CAN bus, examples include electromagnetic interference (EMI), are much more difficult to test in isolation. Last, the CAN data was generated from a controlled environment, and not sampled from real vehicles.

1.4 Terminology

This section introduces key terms and concepts frequently used in the study to ensure clarity and consistency. In-depth descriptions are included in Chapter 2.

The *physical layer* of the CAN bus refers to the electrical and physical components, such as wires, controllers and ECU:s.

Termination refers to the use of resistors to reduce signal reflections on a transmission line such as the CAN bus. The CAN standard define how termination should be installed to ensure signal integrity.

The *stub lines*, also known as *drop lines*, refers to the wires that connect individual ECU:s to the main CAN bus line. The stub lines form branches of the CAN bus and can imply signal reflections interfering with the main bus line and are best kept as short as possible.

The CAN bus has two logical states – recessive and dominant. Due to the electrical properties of the CAN bus, recessive is represented by a logical 1, and dominant by a logical 0.

A *signal edge*, or *signal transition*, refers to the shift of one logical state to another. It is also referred to as a *bit transition*, but it is important to note that it does not refer to a transition from one bit to another of the same value. An edge from recessive to dominant is called a *falling edge*, and a transition from dominant to recessive is called a *rising edge*.

Ringing is a form of signal distortion characterized by oscillations following a signal transition. Ringing can result in signal transitions being sampled where they are not expected.

In the abstract representation of a CAN bus, the ECU:s are referred to as *nodes*. The outline of a CAN bus described in this thesis is therefore defined by the electrical wires, connection points, stub lines, nodes, and termination settings.

2

Theory

In this chapter, a background to the Controller Area Network and machine learning is presented. Following the background, a section on relevant studies to CAN analysis with machine learning is provided.

2.1 Controller Area Network

CAN was developed in the late 1980's by Bosch to fill the increasing need for high traffic communication within vehicles [7]. In 1991, it was first used in a production vehicle, and a few years later it was standardized by the International Organization for Standardization (ISO). Today, CAN is defined by the ISO 11898 standards which, among other things, define how the bus should be constructed and how reliable communication is accomplished [4]. It is designed to allow ECU:s throughout the vehicle to communicate with each other concurrently to accommodate the growing amount of smart systems and computers inside modern cars.

The CAN bus protocol handles communication through transmission and reception of "CAN frames". There are four types of frames: the Data frame, the Error frame, the Remote frame, and the Overload frame. It is with the Data frame normal data communication is performed. An overview of the Data frame is found in Table 2.1, and a visual representation is presented in Figure 2.1.

Messages transmitted onto the bus are received by all active nodes on the network, and depending on which bits are transmitted and at which point, listening nodes responds to the bus by transmitting bits of their own [7]. This reactive behavior can happen under many circumstances, normal as well as faulty, and can result in unpredictable states on the bus. For example, on an idle bus, several nodes can start communicating simultaneously.

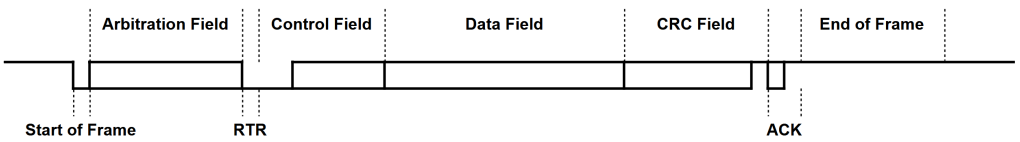

Figure 2.1: Graphical representation of the standard CAN Data frame.

Field	Description
Start of frame (SoF)	One dominant bit denoting the start of the transmission.
Arbitration	Contains the identifier of the message which determines the priority if multiple messages are transmitted simultaneously.
RTR	A dominant bit to indicate that the frame is not a Remote frame.
Version Control	Specifies the CAN version of the message. Part of the Control Field.
DLC	Describes the number of data bytes in the frame. Part of the Control Field.
Data	Includes the data of the frame, consisting of 0-8 bytes.
CRC	Ensures data integrity by detecting transmission errors using a checksum of the transmitted data.
ACK	A confirmation that the transmission was successful. All active nodes acknowledges that the message has been received. In case of an error, the sender transmits the message again.
End of frame (EoF)	Seven recessive bits indicating the end of the message.

Table 2.1: An overview of the CAN Data frame.

The CAN protocol handles such simultaneous transmissions through arbitration on the identifier of the CAN frame [7]. There are two specifications on identifier formats in standard CAN – CAN 2.0A and CAN 2.0B [7]. In CAN 2.0A, the identifier consists of 11 bits, and in CAN 2.0B, the CAN frame has a 29 bit identifier. Through bitwise operations of the identifier, the CAN protocol gives priority to messages with lower identifiers, described in detail below.

Nodes are connected to the bus in a wired-and fashion, meaning that any node transmitting a logical 0 will yield a logical 0 on the bus, regardless of whether any other nodes transmits a logical 1 [7]. If a node transmits a logical 1 and reads a logical 0 from the bus, it will interpret it as a bit error. Should this occur during the arbitration phase, the sender interprets the bit error as losing arbitration.

For example, if node A transmits a frame with ID = $14D_{16} = 000101001101_2$ and node B transmits a frame with ID = $14A_{16} = 000101001010_2$ simultaneously, both nodes will transmit and receive the same logical number up until the ninth bit, where node A transmits a logical 1 and receives a logical 0. This results in node A losing the arbitration, which stops it from transmitting and instead listens to the frame sent by node B. Fig. 2.2 illustrates this example.

One important note is that ID:s must be unique to nodes [4]. If a CAN bus is not implemented with unique identifiers, two nodes starting transmission simultaneously with similar ID:s will result in both nodes winning arbitration, leading to faults. This means that, for a correctly configured CAN network, a transmitting node can only be sure that it reads its own frame from the CAN bus after the arbitration phase.

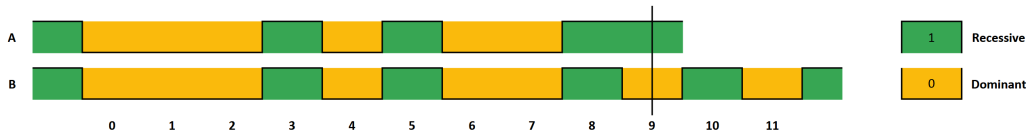

Figure 2.2: An example of arbitration. Node A transmits a frame with ID = $14D_{16}$ which is greater than node B's ID = $14A_{16}$, therefore losing arbitration.

It has been shown that unavoidable small manufacturing inconsistencies results in unique signal behavior to a sending node [8] [9]. It is expected that a bit transmitted from a node is of similar length during similar conditions, and dissimilarities of bit lengths transmitted from the same node indicates that one or more conditions have changed. In the context of fault detection, a change in bit length would indicate that a fault has occurred. As one or many nodes can transmit simultaneously during the arbitration phase, it is not possible to know that bits sampled during arbitration was transmitted from a certain node. Hence, it is not possible to draw valuable conclusions regarding the bit length from analyzing bits transmitted during the arbitration phase, as changes in bit length could either indicate that a fault has occurred, or that arbitration has occurred.

2.1.1 Error Handling

The CAN protocol includes extensive error detection and error handling mechanisms to ensure data integrity and fault confinement [7]. CAN error handling operates on several levels, including bit-level monitoring and frame-level consistency checks. When an error is detected, error frames are transmitted to all other nodes on the network, the faulty message is invalidated and a retransmission is triggered.

Detecting errors is done through a number of different mechanisms, for example, cyclic redundancy check (CRC), frame check and ACK check [4]. During CRC, the transmitting node calculates a check sequence of the Data frame that is compared to the check sequence calculated by the receiving node. If they do not match, a data transfer error is detected and an error frame is transmitted. The frame check involves all senders and receivers checking if the fixed-format fields, such as SoF and EoF, are present and have the right value. Last, the ACK check occurs when a receiving node acknowledges that a message has been received correctly by sending back a dominant bit in the ACK slot.

A bit error occurs when a transmitting node reads a different logical value on the bus than the value it transmitted [7]. Nodes continuously monitor the bus during transmission which allows for immediate detection of inconsistencies. Bit errors are expected during the arbitration phase, where losing arbitration is intentionally detected as a bit error and handled without triggering an error frame. Outside the arbitration field, however, bit errors indicate a fault condition and results in transmission of error frames.

As mentioned, Error frames are structurally different from Data frames. Error frames violates the rules of a CAN message and consist of six consecutive domi-

nant bits and of eight consecutive recessive bits. This error delimiter provides space for the other nodes to detect the error currently on the bus [7].

For physical layer errors, the exact cause is not easy to detect and locate [10]. Common physical layer errors include incorrect bus termination, incorrect voltage level and bad grounding. The result is either error frames being transmitted and handled by the protocol or different timing values from the norm, but the origin of the problem remains and will continue to affect the system.

2.1.2 Physical Layer

The data on a CAN bus is represented as a stream of bits encoded according to the Non Return to Zero (NRZ) method to maintain synchronization between nodes. NRZ encoding is a form of digital data transmission where the data bits are represented by two distinct, constant voltages [7]. This has the benefit of only requiring a minimum bandwidth for transmission, but the bit stream contains no information of the bit clock. Since NRZ does not provide inherent clock recovery, CAN enforces bit-stuffing rules to guarantee sufficient signal transitions. After five consecutive bits of the same logical value, the transmitter automatically inserts a complementary bit. The receiving nodes remove this stuff bit during decoding.

The CAN bus line is composed of two intertwined wires: CAN High (CAN_H) and CAN Low (CAN_L). Every node has a CAN transceiver. During transmission, it converts bits to a differential voltage on the CAN_L and CAN_H wires, and during reception, it converts the differential voltage to bits. Due to the lower production cost, there are CAN versions using a single wire. However, this increases the risk of signal interference and can only be paired with a low data transfer rate [4].

Rising edges, i.e., the transitions from dominant to recessive, are achieved by disconnecting the energy source driving the differential signal of the system [7]. The bus state returns to recessive when the energy degrades over the passive components. The falling edges, i.e., the recessive to dominant transitions, are in contrast driven by the transceivers of the nodes by creating the differential voltage between CAN_H and CAN_L [11].

The two signal transitions affects the CAN bus in different ways, and rising edges is of particular importance for evaluation of the signal integrity of a system [7]. During rising edges, capacitive and inductive energy discharge and influence each other. Recharging between inductive and capacitive energy may occur, which results in oscillations of the bus signals. This concept is known as ringing, and can imply signal interference on the main bus line.

2.1.3 Termination

When a signal reaches the physical end of a transmission line, the electromagnetic energy is reflected back toward the source, interferes with ongoing communication and distorts the signal waveforms [7]. To prevent such reflections, the CAN standard specifies that each end of the bus shall be terminated with a 120Ω resistor. This

practice, known as termination, ensures that the energy of a signal is absorbed at the ends of the bus.

The behavior of signal reflections on the CAN bus is governed by the characteristic impedance of the transmission line [7]. Characteristic impedance, separate from the physical quantity impedance, is an inherent electrical property of a transmission line which describes the relationship between the current and voltage, and the propagation speed along the line. The predetermined termination value of 120Ω is selected to match the characteristic impedance of the transmission line [7].

In a classical two-ended CAN topology, the two 120Ω termination resistors are connected in parallel between CAN_H and CAN_L [7]. For a transmission line with characteristic impedance $Z_0 = 120\,\Omega$, correct termination requires that the load impedance Z_L at each physical end of the line equals Z_0. If this is correctly configured, the voltage reflection coefficient Γ (Eq. 2.1) is zero, and no reflections occur [12]. The traveling wave is fully absorbed by the termination resistor, resulting in clean signal edges and minimal electromagnetic interference.

$$\Gamma = \frac{Z_L - Z_0}{Z_L + Z_0} \tag{2.1}$$

If this termination, for whatever reason, is not at the intended value, the signal integrity is affected and reflections will occur [12]. Furthermore, when the load impedance at the end of the bus is lower than the characteristic impedance, the system is over-terminated, which reduces signal amplitude and could lead to misinterpreted bits as edge timings are moved. If the load impedance is much higher or missing in one end, the system is under-terminated, causing reflections that amplify existing signals.

It is important to note that the two termination resistors at opposite ends of the bus do not combine into a single equivalent resistance for traveling waves. Each termination must locally match the characteristic impedance at the moment the wave reaches that end. Therefore, correct termination must be present at both physical ends of the CAN bus to ensure stable, reflection-free communication.

One can illustrate the effects of termination on the CAN bus by changing the termination and investigating the signals on an oscilloscope. Fig 2.3 shows the first dominant data bit of a CAN frame. CAN_H is represented by the blue line and CAN_L by the red line. The upper figure illustrates the bit measured with 120Ω termination in both ends, and the lower figure shows the same data bit on a CAN bus with 60Ω termination in one end. The differential voltage of the bit in the upper figure is slightly greater compared to the bit transmitted on the bus with a 60Ω termination in one end. This differential increase will result in a longer absorption time for the bus, leading to a longer bit length.

The bit measured on the bus with 120Ω termination in both ends had a bit length of $2.0078125\mu s$, and the bit which was measured on a bus with a 60Ω termination in one end had a bit length of $2.034375\mu s$, an increase of $2.034375\mu s - 2.0078125\mu s = 0.0265625\mu s$. While $0.027\mu s = 27$ ns is a very short time period, it is an increase of

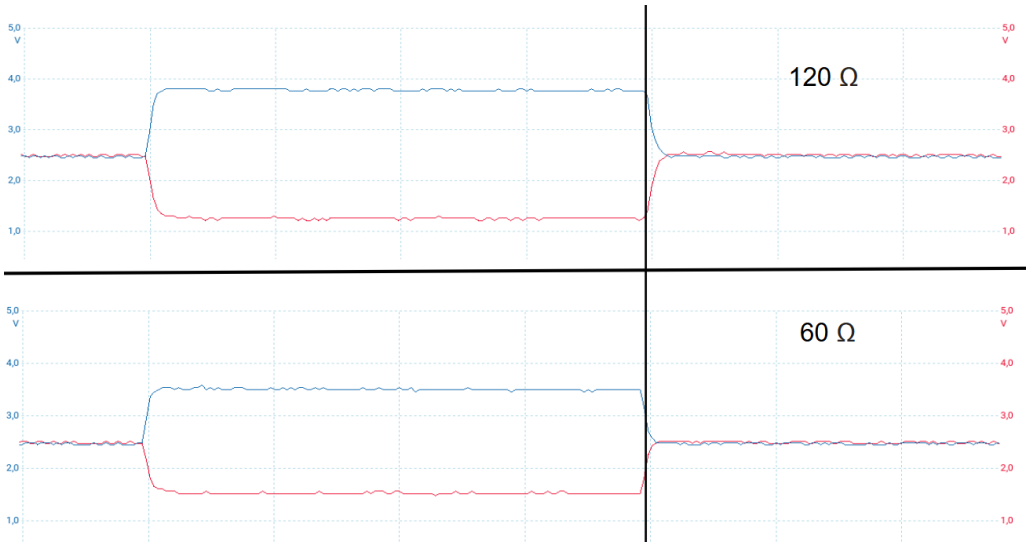

Figure 2.3: One dominant bit visualized in an oscilloscope, measured with 120Ω termination in both ends (top) and 60Ω termination in one end (bottom). The vertical line is drawn where the 60Ω system transitions to recessive state.

$\frac{0.0265625}{2.034375} \approx 1.3\%$. The procedure of measuring the bit length is described in detail in Section 3.1.

2.1.4 Network Architecture

The size and topology of a CAN bus has a large impact on the signal properties [7]. For example, reflected signals intensifies in networks with symmetric topology, since the signals come together at junction points and amplify. In contrast, in systems with low symmetry, the amplitude of reflected signals may erase each other. In one case, a sixth node was added to an asymmetric network with five nodes. As the sixth node created a symmetric topology, the propagation delay was more than doubled [7].

The maximum CAN network size is constrained by two interrelated parameters – the bit rate and the cable length [7]. Moreover, the bit rate determines the duration of each bit, and as the bit rate increases, the time between bits decreases, leaving less margin for signal propagation across the bus. For arbitration to work correctly, all nodes must be able to sample each transmitted bit before the next one is transmitted. Therefore, the round-trip propagation delay of the signal must be shorter than the defined timing segments. In other words, the signal must be able to reach the end of the bus line and come back before the next bit is transmitted. If the network cable is too long, and the bit rate is too high, the signal may not travel to the farthest node and back in time for proper synchronization.

Conversely, increasing the cable length increases propagation delay since the speed of the signal through the bus line is approximately 5 ns/m [7]. To maintain reliable communication over long distances, the bit rate would have to be reduced to allow enough time for synchronization. Stub lines, i.e., the cables connecting the nodes

to the CAN bus, are also a part of the total cable length of the bus and therefore increases the transmission delay [7]. Moreover, since stub lines branches the bus line, they cause reflections. This can cause ringing and delays, and in turn, bit errors, and the effect is increased the longer the stub lines are [7].

Signal quality is not only governed by the bus layout and can, in addition, be heavily affected by the measurement point in the system as every node, cable and connection has the possibility of affecting the signal through, for example, reflections [7]. Therefore, for an accurate analysis of the CAN bus, it is often not sufficient to measure the system from a single point.

2.1.5 External Factors

External factors such as temperature and time are known to impact electrical communication i different ways, which applies to the CAN_H and CAN_L wires in CAN buses [7]. The equation for the propagation velocity V_p of an electromagnetic wave can be seen in Equation 2.2, where c is the speed of light in vacuum and ϵ_t the dielectric constant or relative permittivity of a material.

$$V_p = \frac{c}{\sqrt{\epsilon_t}} \tag{2.2}$$

The relative permittivity changes with temperature and therefore affects the propagation velocity, resulting in a change in propagation delay. Since electrical resistance of conductors depends on the temperature (Eq. 2.3), higher temperatures would also cause cables to have a higher resistance.

$$R_T = R_{T_0}(1 + \alpha\Delta T) \tag{2.3}$$

Temperature changes causes faults such as drift of the oscillators and longer signal delays. This does not necessarily disrupt the communication, since they are relatively small, but these deviations affect the timing of bit transitions and reduce the systems resilience towards physical faults [7]. The exact effect depends on, for example, the materials of the communication cable. However, small components such as integrated circuits have shown a linear increase in propagation delay as temperature increases [13]. Different cable types including a type of coaxial cable showed an increased delay from -40 to -100 $ppm/°C$ when going from 18°C to 33°C [14].

2.2 Machine Learning

Machine learning is the procedure of teaching a computer to recognize patterns in data with the intention of using that learned behavior to make predictions [15]. Furthermore, machine learning enables computer programs to automatically adapt to experience, giving it the possibility of taking action without explicit instruction [16].

This opens up many possibilities of smart applications and automation such as content and product recommendation, anomaly detection, and predicting a patient's risk of having a certain disease, to mention a few.

Machine learning can be roughly categorized into two classes – supervised and unsupervised learning – where supervised learning regards training a model with data with predefined target labels, and unsupervised learning lacks such labels. This distinguishes machine learning into a set where training is based on known ground truth values, and one set which does not. The goal of supervised machine learning is to model a function by mapping input feature vectors to output target vectors [16]. This can prove to be accurate for well-defined problems, but it is not trivial to create useful labeled datasets. On the other hand, in unsupervised machine learning, the goal may be to group data of similar shape, or to determine the distribution of the input data [15]. One can distinguish these ML paradigms as supervised machine learning being task-driven and unsupervised machine learning being data-driven [16].

There are many potential machine learning applications and most models are often specialized in one area such as regression, classification or clustering [16]. Regression is the task of fitting a line through historical data, thereby giving an idea of the trend and predicting future data points. Classification is the task of assigning data to classes or categories. Clustering is the process of separating data points into clusters based on a similarity score.

Deep learning refers to a class of machine learning methods that employ multi-layer neural networks to learn complex data patterns and representations given enough training [17]. There are two key aspects – models consisting of several layers or stages of nonlinear data processing, and methods for supervised- or unsupervised learning of feature representation. Traditional machine learning algorithms are good at finding patterns in categorical and low dimensional data, but real-world data, such as human speech or images, is much more complex. Therefore, high-dimensional data suggests the need for architectures able to capture those complexities, such as those found within the deep learning paradigm [17].

2.2.1 Preprocessing

Preprocessing is a crucial step in the machine learning pipeline, involving the transformation of raw data into a format suitable for model training [18]. Real-world data is often noisy or inconsistent, with dimensions not correlating to other data points of similar type, which can negatively impact model performance if not addressed properly. Therefore, preprocessing aims to clean and standardize the data before it is applied to ML algorithms.

One common example is the scaling of numerical features. Since many machine learning algorithms are sensitive to the magnitude of input values, features with large numerical ranges may dominate. Techniques such as normalization or standardization ensure that all features contribute more equally during training [18]. Another important aspect of preprocessing is handling categorical features, which represent discrete categories rather than continuous numerical values [19]. These fea-

tures are often encoded into numerical form to be compatible with machine learning models. Common encoding techniques include one-hot encoding and label encoding, depending on whether the categorical values have an inherent order.

Feature engineering is the process of transforming raw data into more informative representations that improve a model's ability to learn patterns [16]. This can involve creating new features, modifying existing ones, or selecting a subset of relevant features. Extracting statistical features from data is an example of feature engineering that reduces the dimensionality of the data. However, complex and non-linear patterns could be lost in such transformation.

In addition to feature transformation, preprocessing may involve restructuring the data into appropriate formats [18]. Sparse datasets, i.e., datasets with entries of varying size, often require reformatting of the data into consistent input dimensions, e.g. by padding or truncating the data.

2.2.2 Evaluation

There are many ways of evaluating a machine learning model. A common first step is to partition the dataset into one training set and one test set [15]. The training set is used to tune the parameters of the model during the training phase, and the test set is used to evaluate the model's generalization capabilities. Without a test set, it would not be possible to determine whether the model has learned to identify abstractions, i.e., patterns in the data, or just to recognize the data in the training set. The goal of a machine learning model is to make valuable predictions on unseen data, and not to "remember" the dataset it was trained on. Therefore, it is evaluated on the test set, which it has not seen during training. A model seemingly learning patterns in the training set but performing poorly on the test set is referred to as an overfit model.

Evaluation metrics are used to assess the model's performance on the test set. For a binary classification task, the outcome is referred to as positive or negative [20]. It originates from diagnosing patients, where the result of a screening is referred to as positive or negative. True positive (TP) refers to a correct positive prediction, and true negative (TN) is a correctly predicted negative sample. Similarly, false positive (FP) and false negative (FN) represent the incorrectly predicted cases. These four classes are often presented in a 2×2 matrix called a confusion matrix (Table 2.2). The confusion matrix give valuable insights into which predictions it is failing to do. For example, a high degree of false negatives is very bad in the disease prediction task.

Some of the most common metrics for classification include accuracy, precision, recall and F1-score [20]. Accuracy indicates the fraction of correct predictions out of all predictions (Eq. 2.4). Precision measures the fraction of correct positive predictions out of all positive predictions (Eq. 2.5). Inversely, recall measures the fraction of correct positive predictions out of all positive samples (2.6). F1-score measures the harmonic mean of precision and recall (Eq. 2.7). These measures are often used together to highlight potential biases in the model, which is especially common in

		Predicted	
		Positive	Negative
Actual	Positive	TP	FN
	Negative	FP	TN

Table 2.2: Overview of a binary Confusion Matrix.

datasets with class imbalance.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \in [0, 1] \tag{2.4}$$

$$Precision = \frac{TP}{TP + FP} \in [0, 1] \tag{2.5}$$

$$Recall = \frac{TP}{TP + FN} \in [0, 1] \tag{2.6}$$

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \in [0, 1] \tag{2.7}$$

For multiclass classification the basics are the same, but there are no positive or negative definitions. With $n \geq 3$ classes, results are presented in a $n \times n$ confusion matrix where x_{ij} denotes how many times class i was predicted when the true class was j $(0 \leq i \leq n, 0 \leq j \leq n)$ [20]. The true predictions are presented in the diagonal $x_{ii}, 0 \leq i \leq n$, similar to binary classification.

The evaluation metrics are calculated by creating a 2×2 confusion matrix for each class where the class is the positive outcome and the other classes are negative, and averaging over the derived metrics.

To attempt to give a better estimate of the generalization of the model, k-fold cross validation (CV) can be performed [20]. The training set X is split into k folds, or partitions: $X_1, X_2, ..., X_k$. For each split i $(1 \leq i \leq k)$, the model is trained on $X \setminus X_i$ and evaluated on X_i. Each fold i returns a score, e.g. the accuracy (Eq. 2.4), of the predictions made on the validation set X_i. Cross validation thus returns a set of k scores $\{s_1, s_2, \ldots, s_k\}$. Computing the mean and standard deviation of the scores indicate the stability of the model and its hyper-parameters. If there is a high variance of the scores, the model was sensitive to the data it was trained on.

Cross-validation does not aim to replace evaluation of the model on the test set. Instead, it is used to indicate the generalizability of the model based on the training set, before the test set evaluation is performed. For example, it can be used to fine-tune the hyperparameters of the model.

Baseline comparison is a fundamental component of empirical evaluation in machine learning [21] [22] [23]. A baseline model provide a reference point against which the performance of a proposed model can be interpreted, helping to determine whether observed improvements are meaningful or merely reflect the presence of trivial structure in the data, or trivial prediction making. Furthermore, baselines are commonly included to assess whether learning is justified and whether increased model complexity yields tangible benefits.

Feature importance is a diagnosis of model behavior rather than a direct measure of how important a feature is in the dataset for a particular model [24]. Moreover, it quantifies how a trained model uses input variables to optimize a given objective, and thus reflects both the inductive biases of the learning algorithm and the structure of the loss function.

Formally, feature importance analysis aims to answer the question: *how is the model's performance affected by altering or removing a feature?* [24]. Since feature importance is defined with respect to a trained model, it is inherently model dependent. Two models trained on the same dataset may evaluate features' importance very differently due to differences in the architecture.

2.2.3 Models

Logistic Regression is a well known method used for classification that models the probability that a given data point belongs to a certain class [15]. It uses the logistic sigmoid function to map a linear function output to a probability. Moreover, the logistic regression model is trained using the maximum likelihood estimation method to find the best coefficients that minimize the logarithmic loss.

Decision trees offer a straightforward and interpretable approach to machine learning. They are modeled as branches of sequential questions about the data, starting from a root node that represents the entire dataset and ending in a leaf node that indicates the final prediction [25]. Furthermore, the internal nodes represents decisions regarding the value of a specific feature, splitting the tree into two or more branches. The decisions, consisting of a feature and a threshold, comes from a chosen criterion, which commonly aims to maximize the separation of the target variable such as gini impurity and information gain [25].

Random Forest is an ensemble model consisting of a multitude of decision trees countering the poor generalization observed from traditional decision trees [26]. The model accomplishes this by constructing trees in randomly selected parts of the data, effectively training on separate subsets of the feature set. This feature subsampling decorrelates the individual trees by preventing dominant predictors from repeatedly driving the same decision boundaries, thereby encouraging diverse and complementary models. During inference, each tree independently produces a prediction, and the final output of the forest is obtained through aggregation, an example being majority voting.

Gradient Boosting is another iteration of the decision tree ensemble concept which starts with a single, weak decision tree, and sequentially adds new trees that correct

errors made by previous trees. This continues until a set number of trees are met or a certain performance criterion is reached [27]. The training of Gradient Boosting uses gradient descent as a method for minimizing the loss of the ensemble model.

Linear Support Vector Machine (Linear SVM) is a machine learning algorithm specialized for linearly separable data that classifies input data by finding a hyperplane such that the distance between each class is maximized [28]. The algorithm maps input vectors to a much higher dimension space to construct a linear decision surface and draws adjacent lines along the data to define the margin of largest separation between the classes.

K-Nearest Neighbor (KNN) is a nonparametric learning method that makes predictions based on similarity of data points [15]. The algorithm identifies the k closest points in the feature space and assigns a label according to the majority class among these neighbors. This approach relies on the nearest neighbor classification principle originally described in [29], making it particularly effective for tasks such as clustering and anomaly detection where local structure in the data is informative.

Multi-Layer Perceptron (MLP) is one of the foundational architectures in deep learning, consisting of multiple layers of interconnected neurons arranged in a feedforward manner [30]. Input data is fed through the network sequentially, passed to one or more hidden layers that apply a learned linear transformation and a nonlinear activation function before reaching the output layer. This layered structure enables the network to learn complex patterns. It relies on the backpropagation algorithm to update the weights based on how far from the desired output the model is, which is measured by a selected loss function.

The Convolutional Neural Network (CNN) has achieved major success within deep learning, especially for computer vision and natural language processing [17]. A CNN is a feed forward neural network that automatically learns hierarchal representations from data by extracting local features through convolutions. Furthermore, its architecture is composed of several different layers: convolutional layers to extract local patterns, pooling layers to reduce dimensionality and activation layers to introduce non-linearity.

While traditional CNN:s are often associated with two-dimensional data such as images, one-dimensional CNN:s (1D-CNN) are well suited for sequential data similar to time series [31]. In these cases, convolutions are applied to a single temporal or spatial dimension to capture local relationships and sequential patterns. For this reason, they are commonly used for sequential feature extraction.

2.2.4 Hyperparameters

A model's hyperparameters define how it learns and how predictions are made [32]. Adjusting them can therefore drastically improve or worsen the performance of a model. Examples of common hyperparameters are model size, loss function or criterion and learning rate.

The loss function is a mathematical equation that aims to measure how much was

lost, or how bad an outcome was, to act as a guiding hand for the model [15]. Because of this, the goal of training (for models relying on loss functions) is to minimize the loss from a prediction or set of predictions.

Learning rate refers to the size of the steps taken when attempting to minimize the loss. It determines how fast a model learn from the training data, but a larger number will not necessarily yield better results as the model might overshoot loss minima if the learning rate is too high.

2.3 Related Work

CAN anomaly detection is widely researched, mainly focusing on detecting intrusion attacks or physical faults of a system. The first area categorizes under cybersecurity and protecting vehicles against attacks, and the second electrical engineering and hardware.

Hanselmann et al. [33] proposed an unsupervised intrusion detection model for CAN frame timestamp data called *CANet*, a novel neural network capable of simultaneously working on messages with different ID:s. It used several smaller models, one for each ID, and concatenated the output into an Autoencoder Neural Network. The motivation of using an unsupervised model was to be able to detect unknown attacks as well as known attacks.

Autoencoders have been a popular choice because of their ability to handle the high dimensionality that is found in CAN data with both information and voltage data included. Battaglia et al. [34] explored CAN anomaly detection using an Autoencoder which was able to learn efficient representations of the input CAN data without supervision. They used several one dimensional convolutional layers both in the encoder and decoder parts of the Autoencoder to separate normal CAN voltage data from anomalous. The model achieved high accuracy while keeping the inference latency low, but has only been evaluated in a controlled testing environment.

Levy et al. proposed *CAN-LOC*, a deep learning model based on Autoencoders and CNN:s [35]. The model was able to detect and localize intrusions using CAN voltage data without relying on an adversary's transmission data, increasing its effectiveness against silent intruders. The model was installed on the bus as an ECU and analyzed the CAN bus voltage signals as soon as the car was started. It was also able to adapt to environmental changes, something that affects the CAN bus signals significantly.

Unavoidable small inconsistencies in manufacturing of transceivers results in signal behavior unique to the sending node, which makes it possible to identify the source of a CAN message [8] [9]. Based on this assumption, Liu et al. proposed *vProfile* [36], a model trained to detect anomalies from edge sets, i.e., voltage traces of rising and falling edges. By collecting only the first edge set from the data field of a CAN frame, they managed to identify which ECU transmitted the message, suggesting that signals transmitted from an ECU contains unique electrical properties.

Long-short term memory (LSTM) models have previously been applied to fault detection problems because of their natural fit with sequential data, which comes

from their ability to learn both short- and long-range patterns without suffering from the vanishing gradient problem [37] [38]. This makes them excellent in handling tasks such as time series classification and natural language processing, but also for detecting faults in a running system. Ngo et al. used an LSTM network to decode CAN signal data from one vehicle using labeled CAN signals from another vehicle [39]. The network was able to, only using the CAN frame ID, timestamp and data bytes, correctly decode radar signals of another vehicle.

Chandralekha et al. compared two unsupervised models' abilities to detect anomalies in recorded CAN logs, a DBSCAN model and an LSTM Autoencoder [40]. Both models performed well on major pattern changes in the signal data, but only the LSTM Autoencoder detected the small pattern changes caused by anomalies.

Hossain et al. used an LSTM to create a Vehicle Intrusion Detection System based on CAN timestamp and message data [41]. They successfully managed to detect suspicious signals, such as spoofing or denial of service attacks, with high accuracy.

LSTM:s have also been used in combination with other architectures to create performative hybrid models or transformers. One such case is [42] where Zhang et al. used a combination of a CNN and a LSTM to produce a state of the art model for driver identification. Much of a driver's behavior in a car, for example acceleration and breaking habits, corresponds to CAN communication. The authors proposed an end to end model which used the feature extraction ability of a CNN in combination with an LSTM's ability to detect patters over longer time periods.

Another way researchers have been utilizing machine learning together with CAN is trying to reverse engineering CAN frames. This involves figuring out the semantic meaning and format of the signals sent from nodes in the network, and reconstructs them using only the data itself. In [43] Lin et al. proposed *ByCAN*, an automatic reconstruction tool using DBSCAN clustering and Template matching to decode CAN frames without any prior knowledge about them. It accomplished greater accuracy than the existing reverse engineering methods at the time by introducing many byte- and bit-level features. Buscemi et al. [44] showcased a supervised learning model that utilized the fact that different vehicle models reuse frame ID:s to achieve CAN bus reverse engineering. They also showcased an improved iteration of the READ algorithm (Reverse Engineering of Automotive Data frames) for better CAN frame tokenization by considering signal endianness.

There has already been at least one previous study by Peng et al. where physical fault detection on the CAN bus using machine learning was attempted using the voltage data of the CAN bus [45]. While they were able to successfully classify faults from their input using a 1D-CNN-LSTM hybrid model, the data and types of faults were not described in detail. Furthermore, they do not describe the relation between the physical setup, the faults and electrical waveforms. Since the voltage on a CAN bus deviates from expected values under normal conditions, such as during arbitration, an analysis must be performed on the correlation of physical faults, signal waveforms, and the part of the CAN message you can draw conclusions from.

We propose another solution that uses only a fraction of the data that others have

previously relied on, making it more suitable to be used in real time. Our method aims to predict physical faults on a CAN bus only using the offset of recessive transitions. Furthermore, this methodology accounts for arbitration, and only collect data points from the data field if arbitration is detected, which ensures that the recorded offsets are collected from only one node.

3

Methods

A test bench was configured to the physical configurations that this study aimed to investigate, faulty as well as normal. An interface was installed on one end of the bus which sampled the CAN bus and wrote the data to a log file. CAN traffic was generated with a script configured to run a similar number of times per physical configuration and measure point. This way, equal amounts of data was generated per fault configuration, which addresses the class imbalance problem.

This process was repeated for each test bench configuration investigated. The files were then preprocessed, labeled and merged into a common CSV dataset. On this dataset, a set of ML models was trained and evaluated.

This chapter includes in depth descriptions of each of these steps. First, necessary tools and measurements are described.

3.1 Tools

CANlib is a Software Development Kit (SDK) developed by Kvaser [46]. It is used to interact with Kvaser's CAN interfaces, enabling transmission and reception of CAN messages. Transmission of a CAN message requires connection to a CAN channel, an identifier (ID), and a payload of 0-8 data bytes. Reception of a CAN message requires an open CAN channel.

CANtegrity is a signal integrity hardware developed by Kvaser which samples the CAN bus with a frequency of 640 MHz [47]. Newer versions of the hardware has a frequency of 800 MHz, resulting in 1.25 ns between each sample, which was used in this study. CANtegrity is integrated in the CAN controller, which enables real-time analysis of signal properties in correlation to bit-level data defined by the CAN protocol. For example, phase shift of transitions, or unexpected transitions, can be investigated in relation to a certain CAN frame, and to the position within the frame. Such an analysis can identify which ECU a frame was transmitted from and physical faults related to phase shift [47]. In contrast, an oscilloscope-based analysis provides information of the signal characteristics on the CAN bus, such as deviations of the differential voltage over time. In isolation, such an analysis does not relate the signal behavior to CAN specific data.

CANtegrity is utilized with a compatible CAN interface, which outputs a payload from each frame it receives. In addition to the values of a standard CAN Frame

(Table 2.1), CANtegrity outputs a list of sampled signal transitions ("samples"), and a list of detailed bit data ("bits"). An excerpt of a CANtegrity frame payload is found in Listing 1.

Items in "samples" (S) contains the entries "value" (v), "start" (s), and "length" (l). "Value" refers to whether the edge is recessive to dominant (0) or dominant to recessive (1), "start" refers to the timestamp of the sampled edge, and "length" refers to the time passed until the next edge was sampled. As an edge occurs when the bus transitions from dominant to recessive or vice versa, every second sample is of value 0 and the other is of value 1. Furthermore, the items in "samples" forms the incremental relationships shown in Equation 3.1 and Equation 3.2.

$$s_n = s_{n-1} + l_{n-1}, n \in [2, |S|] \tag{3.1}$$

$$l_n = s_{n+1} - s_n, n \in [1, |S| - 1] \tag{3.2}$$

The bits in "bits" (B) contains detailed information of synchronization ranges, sample positions, and much more. Regarding edges of a CAN frame, the most important additions to "samples" are the fields "name", "value" (v) and "sync_pos". "Name" is the field of the CAN Frame, such as ID, DLC, and DATA, or STUFF if the bit is a stuff bit. "Value" refers to the binary value of the bit. "Sync_pos" contains a non-zero timestamp if the bit contains a dominant transition, and is zero if the bit is recessive, or the previous bit was dominant (Eq. 3.3).

$$sync\_pos_n = \begin{cases} t & \text{if } v_n = 0, v_{n-1} = 1 \\ 0 & \text{otherwise} \end{cases}, t \in \mathbb{N}, n \in [2, |B|] \tag{3.3}$$

With $sync\_pos_n = t$, t is the start of the dominant edge bit b_n, which is also the start value of a dominant edge in "samples" (Eq. 3.4). This design defines dominant edges as reference points for recessive edges. The relationship between $sync\_pos_m = t$ and $s_n = t$ for bit b_m and edge s_n is illustrated in Listing 1, where the second edge in "samples" with $start = 46276$ is related to the second bit in "bits" with $sync\_pos = 46276$.

$$sync\_pos_n = t, t \neq 0 \implies \exists i. S_i \in S, s_i = t, n \in [1, |B|], i \in [1, |S|] \tag{3.4}$$

All the edges in "samples" corresponds to bits in "bits", but all bits does not contain edges, so most bits in "bits" do not have a reference to "samples". The definition of a dominant transition bit is that the previous bit was recessive (Eq. 3.5). Similarly, the definition of a recessive transition bit is that the previous bit was dominant (Eq. 3.6). The definitions of the sets D and R are exemplified in Listing 2.

$$D = \{b_n | v_n = 0, v_{n-1} = 1\}, n \in [2, |B|] \tag{3.5}$$

```
{
    "payload": {
        "id": 329,
        ...,
        "samples": [
        ...
            {
                "value": 1,
                "start": 44704,
                "length": 1572
            },
            {
                "value": 0,
                "start": 46276,
                "length": 6428
            },
        ...
        ]
        "bits": [
            ...,
            {
                "name": "DATA",
                "value": 1,
                "sync_pos": 0,
                ...
            },
            {
                "name": "DATA",
                "value": 0,
                "sync_pos": 46276,
                ...
            },
            ...
        ]
    }
}
```

Listing 1: Excerpt of a CAN Frame payload in CANtegrity, showing two edges and two data bits. The entries in "bits" have been heavily abbreviated due to space limits.

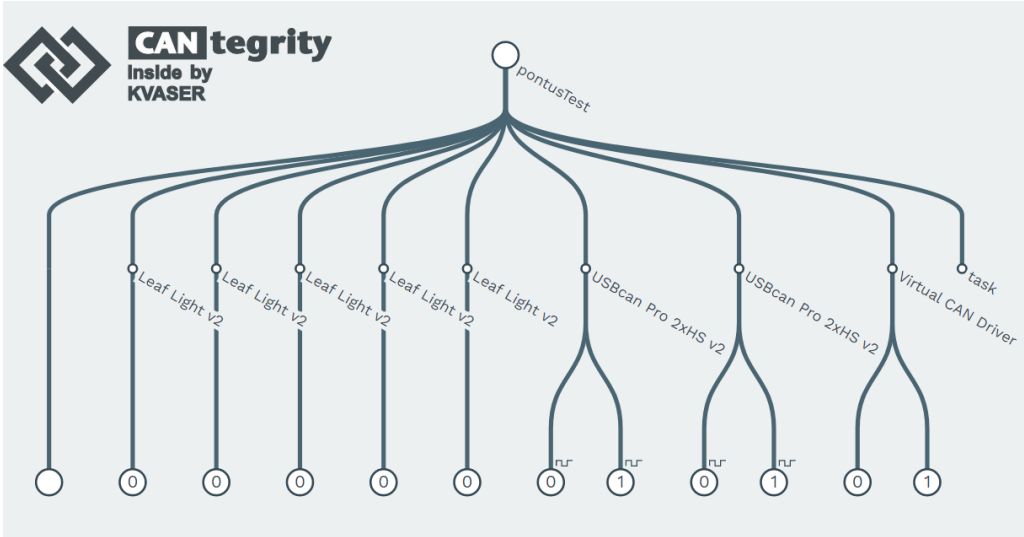

Figure 3.1: Overview of the CANtegrity desktop application with five Leaf Light v2 interfaces and two CANtegrity compatible USBcan Pro 2xHS v2 interfaces connected.

$$R = \{b_n | v_n = 1, v_{n-1} = 0\}, n \in [2, |B|] \tag{3.6}$$

The "samples" list and "bits" list thus forms a complementary relationship, as "bits" is required to find D and R, and "samples" is required to find bit lengths. Furthermore, items in "samples" are triggered on sampled edges, but contains no information of which bit the edge refers to, or how many bits have passed since the last edge (p). Edge lengths widely exceeding the expected bit length is found in "samples", because the edge contains several bits of the same value. To extract the actual bit length from "samples", p must be calculated from "bits" and divide the edge length.

Furthermore, only dominant edges can be found from "bits", as $sync\_pos_m = t$ in "bits" relates to an edge start $s_n = t$ in "samples" only if $b_m \in D$. Therefore, recessive edge bits are found in "samples" by applying Equation 3.1 to t of the previous dominant edge bit's $sync\_pos$.

The CANtegrity desktop application enables interactive setup of a CANtegrity measurement and logging [47]. By configuring logging from an interface, a JSONL file is incrementally created while the CANtegrity interface receives CAN messages. Each row in the JSONL file contains one frame payload. An overview of the CANtegrity desktop application is found in Fig. 3.1.

3.2 Measurements

The average bit length $\bar{L}(m, n)$ of bits $\{b_m, b_{m+1}, ..., b_{n-1}, b_n\} \in B$ in a frame is calculated by subtracting the start of b_m from the start of b_n and dividing by the number of bits between the two. Equation 3.7 describes the calculation of the average bit length, where t_i refers to the start of bit b_i.

```
{
    "bits": [
        ...,
        {
            "name": "BASE_ID",
            "value": 1,
            "sync_pos": 0,
            ...
        },
        {
            "name": "BASE_ID",
            "value": 0,
            "sync_pos": 9698,
            ...
        },
        ...,
        {
            "name": "BASE_ID",
            "value": 0,
            "sync_pos": 0,
            ...
        },
        {
            "name": "BASE_ID",
            "value": 1,
            "sync_pos": 0,
            ...
        },
    ]
}
```

Listing 2: Four bits $b_{m-1}, b_m, b_{n-1}, b_n \in B$ illustrating the definition of dominant and recessive transition bit sets D, R. Here, $b_m \in D, b_n \in R$ since the previous value of respective bit is of opposite value.

$$\bar{L}(m,n) = \frac{t_n - t_m}{n-m}, 1 \leq m < n \leq |B| \tag{3.7}$$

The bit length L is, trivially, the time passed from the start of the bit to the end of the bit (which is the start of the next bit). An issue with this definition of L is that many bits does not contain a starting edge, since consecutive bits often have the same value in a binary message. Specifically, a bit b_n has an edge if b_{n-1} has the opposite value of b_n ($b_{n-1} = 0, b_n = 1$ or $b_{n-1} = 1, b_n = 0$). For example, in 100011_2, the third, fourth and sixth bits do not contain edges.

Therefore, to calculate L_n for any bit b_n, it is required to find the previous bit b_i and the next bit b_j with a starting edge. L_n is then defined as the average bit length of those edges (Eq. 3.8).

$$L_n = \bar{L}(i,j) \tag{3.8}$$

3.2.1 Units

Bit length (L_s, measured in seconds), can be calculated from the bit rate R_b (Eq. 3.9).

$$L_s = \frac{1}{R_b} \tag{3.9}$$

For example, a CAN bus configured with bit rate $R_b = 500$ kb/s yields a bit length of $L_s = \frac{1}{500\ 000} = 0.000002\ s = 2\mu s$.

Bit length can also be measured in terms of clock cycles of the measuring unit (3.10).

$$L_{cc} = \frac{f}{R_b} \tag{3.10}$$

Following the example above and sampling with an interface with frequency $f = 800$ MHz yields a bit length of $L_{cc} = \frac{800\ 000\ 000}{500\ 000} = 1600$ clock cycles. A benefit of measuring in clock cycles is that that small deviations in bit lengths are noticeable to the human eye, and in Section 3.2.2 it is explained that L_{cc} is guaranteed to be a natural number.

The relation between the two measurements is also useful, particularly describing L_s in terms of L_{cc}, which is achieved by substituting R_b for $\frac{f}{L_{cc}}$ in Eq. 3.9.

$$L_s = \frac{L_{cc}}{f} \tag{3.11}$$

This simple equation allows for translation from L_{cc} to L_s, which is the more interpretable unit. For example, one clock cycle ($L_{cc} = 1$) measured with $f = 800$ MHz equals $\frac{1}{800\ 000\ 000}$ s $= 1.25$ ns.

Description	Input	Accepted values	Default values
ID Interval	min, max	$0 \leq min \leq max$	7, 7
ID Size	$size$	11, 29	11
DLC Interval	min, max	$0 \leq min \leq max \leq 8$	8, 8
Delay Interval	min, max	$0 \leq min \leq max$	50, 100
Iterations	it	$-1 \leq it$	1000

Table 3.1: CAN data traffic generator script input.

3.2.2 Quantization

Electrical signals, continuous variables by nature, can not be processed by computers without transformation, since computers perform operations on discrete (binary) values. Therefore, during sampling of analog signals, quantization is performed. Quantization is the process of converting data from high, continuous, precision to lower, discrete, precision [48].

Measuring timestamps t, the true $\tilde{t} \in \mathbb{R}$ is transformed into $t \in \mathbb{N}$, adding an uncertainty u (Eq. 3.12).

$$\tilde{t} \in [t-u, t+u), 0 \leq u \leq 0.5 \tag{3.12}$$

Measuring bit length (Eq. 3.7), both t_m, t_n have been quantized, which results in rounding errors in L. With $t \in \mathbb{N}$, an interval for the true bit length $\tilde{L}$ can be calculated.

$$\tilde{L}(m,n) = \frac{\tilde{t}_n - \tilde{t}_m}{n-m} \in \left(\frac{(t_n - u) - (t_m + u)}{n-m}, \frac{(t_n + u) - (t_m - u)}{n-m} \right) =$$

$$= \left(\frac{t_n - t_m - 2u}{n-m}, \frac{t_n - t_m + 2u}{n-m} \right) \subseteq \left(\frac{t_n - t_m - 1}{n-m}, \frac{t_n - t_m + 1}{n-m} \right)$$

The interval for $\tilde{L}$ thus indicate an uncertainty in L of $U \in \left[0, \frac{2}{n-m}\right)$.

3.3 Traffic Generation

The traffic was generated by transmission of CAN frames with a Python script connected to CANlib. To disable dependency of the CAN frame data, the ID and data was randomized for each frame. Randomizing values was done in Python's random package [49]. The user input an interval of numbers of ID:s, ID sizes, a DLC interval, a delay interval, and number of iterations. Table 3.1 describes the user input in detail.

A dictionary mapping available channels to a list of IDs was generated from the number of ID:s interval and ID sizes. For each channel, a random number of ID:s was generated from $[id\_min, id\_max]$, and each ID was generated from $[0, 2^{ID\_size})$.

```
{
    "ch_0": [922,742,607,1071,608,12],
    "ch_1": [1348,1523,1092,1580,1986,1178],
    "ch_2": [1667,1335,886,1873],
    "ch_3": [974,439,924,1780,585]
}
```

Listing 3: Example of generated ID dictionary with four channels, ID interval = [4, 6] and ID size = 11 bits.

Since frame identifiers must be unique to nodes in CAN, the script made sure that the generated ID was unique to the channel. Furthermore, the dictionary was saved and loaded throughout all measurements with similar number of nodes. An example output from the ID generation script is found in Listing 3.

The script listed the connected channels on the CAN bus from CANlib and let the user choose which channels to generate traffic on. A Python thread [50] ran the main loop for each selected channel, which generated and transmitted CAN messages onto the bus. The main loop ran *iterations* number of times, where $iterations = -1$ represented continuous traffic. Through threading, unpredictable transmission of frames on the CAN bus was achieved, which in contrast to sequential operation could generate simultaneous transmissions and enable arbitration. The pseudo-code of the traffic generation is presented in Algorithm 1.

Algorithm 1: Main loop of traffic generation script.

Input: it, dlc_min, dlc_max, delay_min, delay_max, id_list
$n \leftarrow 0$;
while $n \leq it$ ***or*** $it = -1$ **do**
    $n \leftarrow n + 1$;
    $delay \leftarrow random\_int(delay\_min, delay\_max)$;
    do sleep $delay$ ms;
    $dlc \leftarrow random\_int(dlc\_min, dlc\_max)$;
    $id \leftarrow random\_choice(id\_list)$;
    $data \leftarrow random\_bytes(dlc)$;
    do transmit $Frame(id, dlc, data)$;
end

3.4 Data Preprocessing

Initial filtering included removing potential error frames from the payloads, as error frames behaves unpredictably and was out of scope for this thesis. Furthermore, the last sampled transition was removed, as it corresponds to the ACK bit and is transmitted from the other active nodes of the network. See Algorithm 2 for an overview of the preprocessing procedure.

Algorithm 2: Main preprocessing script.

Input: log.jsonl
$payloads \leftarrow read\_file(log.jsonl)$;
$rows \leftarrow [\,]$;
foreach *payload* ***in*** *payloads* **do**
  if *payload is Error frame* **then**
    skip;
  end
  $samples, bits, id \leftarrow payload$;
  $node \leftarrow get\_measurement\_point()$;
  $bits \leftarrow remove\_ACK\_bit(bits)$;
  $samples \leftarrow process\_noisy\_edges(samples)$;
  $arbitration \leftarrow check\_arbitration(bits)$;
  if *arbitration* **then**
    $bits \leftarrow remove\_arbitration\_field(bits)$;
  end
  $L_E \leftarrow calculate\_expected\_bitlength(bits)$;
  $offsets \leftarrow calculate\_offsets(samples, bits, L_E)$;
  $metrics \leftarrow calculate\_metrics(offsets)$;
  $rows \leftarrow rows + [id, node, L_E, arbitration, metrics]$;
end
$make\_csv(rows)$;

Due to arbitration, edges sampled during the arbitration phase can originate from several nodes. If arbitration occurs, edges in the arbitration field are not representative of the sending node and should be excluded from the frame. However, to include as much data as possible from each CAN frame, an investigation of whether arbitration had taken place or not was performed.

The average dominant bit length of the data field $\bar{L}_{DATA}$ was compared to the average dominant bit length of the full frame $\bar{L}_{FULL}$. This was done by applying equation 3.7 on $\bar{L}(i,k)$ and $\bar{L}(j,k)$, where b_i indicates the first bit of the frame s.t. $b_i \in D$, b_j the first bit in the data field s.t. $b_j \in D$, and b_k the last data bit s.t. $b_k \in D$.

If arbitration did not occur, the average dominant bit length was close to equal in both fields. If the average bit length of the two fields differed, arbitration could be assumed to have happened. Following the argument of quantization error (Section 3.2.2), arbitration was considered to have occurred if $|L_E(i,k) - L_E(j,k)| \geq U$, where U is the uncertainty of $L_E(i,k)$ and $L_E(j,k)$, $U = \frac{2}{k-i} + \frac{2}{j-i}$. If this inequality evaluated to true, only edges from the data field were used.

Ringing can imply registered edge transitions if the ringing occurs around the bit value threshold. The CAN protocol is robust to ringing because of bit value sample points, meaning that if ringing occurs shortly after a transition, it is not interpreted as an actual transition because it is not expected at that time period.

With the high sampling rate of $f = 800\text{MHz}$, if ringing occurred, edges were added to the edges list "samples". Dominant transitions of length $L_{cc} = 1$, followed by a recessive transition of the same length, were found in "samples", without referencing a bit in "bits". Listing 4 shows an example of sampled ringing in CANtegrity.

As a counter measure, dominant transitions with start values similar to dominant bits' synchronization positions were considered to be the true edges, and only those edges were used as reference points (Eq. 3.13).

$$B_T : \{b_n | (b_n \in D, (\exists i | S_i \in S, s_i = sync\_pos_n))\}, n \in [2, |B|] \tag{3.13}$$

Recessive transitions with $1 < L_{cc} \leq 100$ were found in "samples", splitting a dominant edge in two. The ringing counter measure could not be applied, since the length of that edge was cut in half. Therefore, if a recessive edge $S_i \in S$ occurred s.t. $1 < l_i \leq 100$, S_i and S_{i+1} were removed from S, and the bit length of the previous dominant edge was updated (Eq. 3.14).

$$l_{i-1} = l_{i-1} + l_i + l_{i+1} \tag{3.14}$$

An expected dominant bit length L_E was calculated as $L_E = \bar{L}(1, |B_T|)$ (Eq. 3.7 on B_T). As dominant edges were reference points, bits with recessive edges ($b_n \in R$) were extracted from "bits", together with $sync\_pos_m$, the start of the previous dominant edge bit $b_m \in D \cap B_T$, and $p_i = n - m$. From $sync\_pos_m$, an edge length l_i s.t. $s_i = sync\_pos_m$, was extracted. L_E, p_i, and l_i were then used to calculate an edge offset O_i, i.e., the difference of the sampled edge and the expected edge (Eq. 3.15). Algorithm 3 illustrates the pseudo-code of the offset calculation.

$$O_i = l_i - L_E \cdot p_i \tag{3.15}$$

An offset was calculated for each recessive edge bit $b \in R$ throughout the frame, resulting in a set of offsets $O \subset \mathbb{R}$. By calculating the offset of the recessive transitions, the output was standardized and independent of the number of bits p it occurred after.

In addition to the offsets, the CAN ID, measuring point, L_E, and the boolean value of the arbitration check was added as preprocessing features. Table 3.2 presents the features extracted from the preprocessing procedure.

3.5 Feature Extraction

As the number of edges in a CAN frame depend on the binary message and identifier of the frame, the size of the offset array $|O|$ returned by the preprocessing script differed from frame to frame. Statistical features of O, such as the mean, median and standard deviation, were calculated and used as features. Table 3.3 presents two example outputs from the feature extraction, where the statistical features have

```
{
    "samples": [
        ...,
        {
            "value": 1,
            "start": 78526,
            "length": 1572
        },
        {
            "value": 0,
            "start": 80098,
            "length": 1
        },
        {
            "value": 1,
            "start": 80099,
            "length": 1
        },
        {
            "value": 0,
            "start": 80100,
            "length": 8026
        },
        ...
    ],
    ...,
    "bits": [
        ...,
        {
            "name": "DATA",
            "index": 3,
            "value": 0,
            "field": 13,
            "sync_pos": 80100,
            ...
        },
        ...
    ]
}
```

Listing 4: Sampled ringing in CANtegrity. Two edges of length $L_{cc} = 1$ is found in "samples".

Algorithm 3: Offset calculation script.

Input: samples, bits, L_E
Output: offsets
$offsets \leftarrow [\,]$;
$p,\ last\_edge,\ last\_bit\_value \leftarrow 0$;
foreach *bit* ***in*** *bits* **do**
    $p \leftarrow p + 1$;
    if $bit \in D$ **then**
        $p \leftarrow 0$;
        $last\_edge \leftarrow bit.sync\_pos$;
        $last\_bit\_value \leftarrow bit.value$;
    end
    else if $bit \in R$ **then**
        $expected\_edge \leftarrow last\_edge\ + L_E \cdot p$;
        $sampled\_edge \leftarrow retrieve\_recessive\_edge(samples, last\_edge)$;
        $offset \leftarrow round(sampled\_edge - expected\_edge)$;
        $offsets \leftarrow offsets + [offset]$;
        $p \leftarrow 0$;
        $last\_edge \leftarrow sampled\_edge$;
        $last\_bit\_value \leftarrow bit.value$;
    end
    else
        skip;
    end
end
return *offsets*

Feature	Type	Description
id	Categorical	Identifier of CAN message
node	Categorical	Measurement point, one end of the CAN bus
L_E	Continuous	Expected bit length
arbitration	Boolean	Occurrence of arbitration

Table 3.2: Features calculated during preprocessing.

id	node	L_E	arbitration	mean	median	std
1756	1	1600	False	-7.462	-8.0	0.905
1106	1	1599.89	False	9.273	9.915	0.96

Table 3.3: Example output of two processed frames, the first with normal termination and the second with missing termination in one end. Node value 1 indicates that they were both sampled at the left end.

been added to the set of preprocessing features. The first row was sampled from a bus with normal termination (120Ω at each end) and the second where one end had a missing termination.

3.6 Test Bench

A test bench was assembled to enable controlled injections of physical faults on the CAN bus. The test bench was constructed from configurable T-connectors, interfaces, and normal as well as configured cables. The interfaces were connected to the T-connectors, forming the nodes of the network. Between the interfaces and the T-connectors, a stub line was added. The T-connectors were connected to each other with cables, forming a CAN bus of linear topology. An interface was connected to a normal T-connector to enable injections of physical faults. Two CANtegrity interfaces were connected to normal T-connectors on each end of the bus. Those interfaces were not transmitting messages, but instead functioning as sampling points that sent data back to the CANtegrity software for real-time analysis and logging. See Figure 3.2 for a schematic overview of the test bench.

Faults were injected by manipulating the T-connectors through CAN messages with configuration instructions. A separate interface was connected to the bus line, only involved in module configuration communication. This way, it was ensured that fault injection instructions did not interfere with data bus communication.

Two buses running on the same cables was achieved by configuring the cables connecting the T-connectors with two separate CAN buses running on different wires. The configuration bus ran on wires 2 and 7, and the test bench bus ran on wires 1 and 8.

The main component of the test bench, the configurable T-connector, consisted of in-, out-, and T-connections (Fig. 3.3). The in- and out connections were the module's connection to the CAN bus, and the T-connection was the module's connection to an interface.

3.6.1 Faults

Faults were injected by manipulating the termination and connections of the T-connector. The termination of a T-connector could be set to 120Ω, 60Ω, or 10 000Ω. 10 000Ω termination was used to simulate no termination, and 60Ω was used to

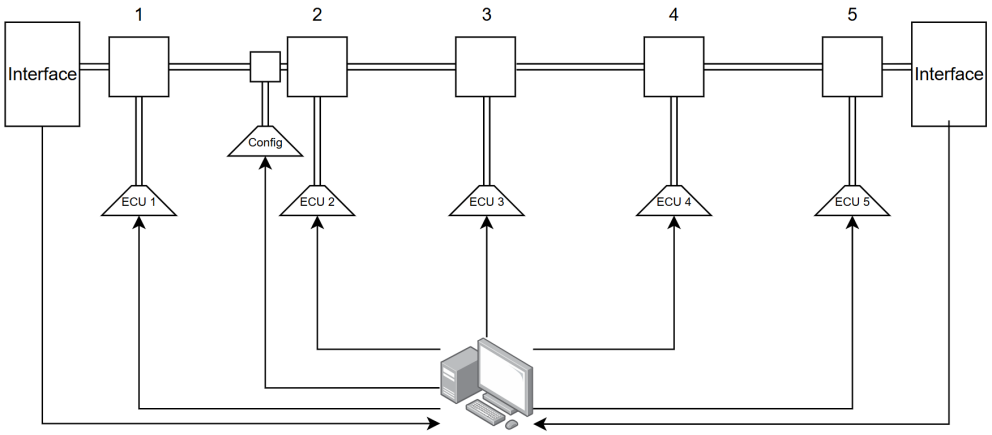

Figure 3.2: Schematic overview of a test bench with five nodes. The squares (denoted 1-5) represents the configurable T-connectors. The cable between the T-connectors makes up the bus line and the cables from the T-connectors to the ECU:s are the stub lines.

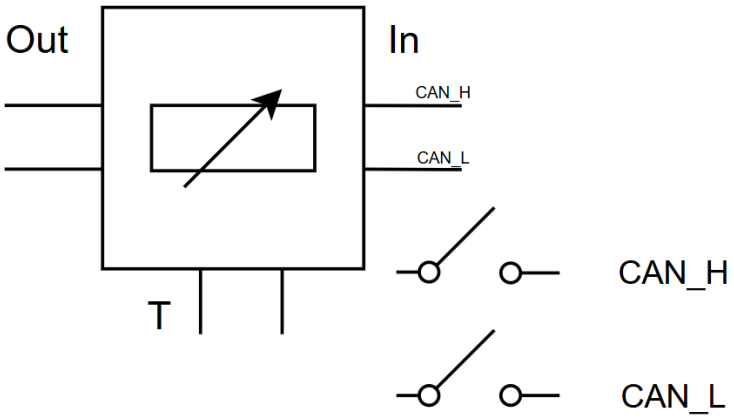

Figure 3.3: Schematic of the T-connectors used in the test bench.

Nodes	Cable length	Stub length
3	2 m	2 m
5	35 m	4 m

Table 3.4: Overview of the different parameters used to create distinct variants of the test bench.

indicate double termination. To disconnect a wire, CAN_H or CAN_L was turned off to a certain connection.

3.6.2 Physical Layouts

The cable lengths and stub lengths affects the electrical properties of the system. With longer cables, the timing of the edges are affected, and longer stub lines increases the amount of reflections. Different layouts of the test bench were evaluated to investigate whether the fault detection was affected by the layout of the CAN bus. Table 3.4 presents the physical variables and their available values during the study.

3.7 Data Exploration

As an initial analysis of the shape and behavior of the data, data exploration was performed. By visualizing two dimensions in the dataset, patterns can be identified, and the results one can expect to be feasible can be indicated.

Scatter plots were created to visualize such relations. For example, the ID versus the statistical mean and the mean versus the standard deviation was plotted to highlight the distribution of O.

Additionally, an in-depth data analysis was performed to highlight influential aspects of the physical setup. By extracting data transmitted from one node and measured from one point, the correlation of fault classes, measuring point and the physical CAN bus layout was illustrated.

3.8 Sequential Analysis

To examine whether the sequential order of the offsets were of importance, two experiments with a 1D-CNN were conducted. The performance of the model was evaluated on the unaltered offset array versus a shuffled and sorted version. Furthermore, it was evaluated whether a 1D-CNN could be used to extract valuable features from the offset array.

Model Name
Logistic Regression
Decision Tree
Random Forest
Gradient Boosting
Linear Support Vector Machine
K-Nearest Neighbors
Multi-Layer Perceptron

Table 3.5: Overview of classifiers evaluated.

3.9 Machine Learning Procedure

The performance of a machine learning model, architecture and paradigm depends on the task and several properties of the dataset, such as the dimensionality and correlation of features. Since data could be generated with ground truths (the fault configuration of the test bench), supervised learning was performed.

To cover several architectures within machine learning, and in turn, to exhaust the possibilities of achieving as good results as possible, a variety of ML classifiers were evaluated. Linear models were used to utilize potential linear patterns, and Decision Tree-based models were used as they excel at finding non-linear patterns in tabular data. Furthermore, K-Nearest Neighbors was included to evaluate whether the data formed separable clusters, and Multi-Layer Perceptron was used to investigate whether deep learning would be able to outperform the traditional machine learning models by identifying complex non-linear patterns. The models used the study are presented in Table 3.5, and were implemented with Scikit-learn's [51] state-of-the-art algorithms.

After processing the data from CANtegrity and constructing labeled datasets, the datasets were prepared to be compatible with the ML models. Depending on the model, a StandardScaler [52] was used to scale numerical features using z-score normalization. One-hot encoding was used on the categorical features *id* and *node*.

The dataset was split into training and test sets, where 80% of the dataset was put into the training set. Equal amounts of each class was put in each set to avoid biases due to class imbalance. A benchmark was performed, meaning that all models were trained and evaluated on equal splits of the dataset, which enabled a fair comparison of the results.

After training, the remaining 20% of the dataset was used to test the performance of the models on unseen data. Testing was performed by having the models making a prediction on each sample of the test set. The set of predictions was then compared to the true labels not known to the models.

To evaluate the performance of the classifiers, the evaluation metrics in Section 2.2.2 were applied to the predictions made by the models and the true labels. In addition to the metrics, confusion matrices were created and a feature importance analysis

was performed for compatible models to provide insights in which features were most critical during training of the model.

Baseline comparison was performed by adding Scikit-learn's Dummy Classifier [53] to the set of models evaluated in the benchmark. The Dummy Classifier uses simple prediction rules rather than learning patterns in the features of the dataset.

The physical setup of the test bench, i.e., the topology, number of nodes and cable lengths, alters the signal behavior of the CAN bus, which increases or decreases the edge offset. Therefore, to evaluate the limitations of the study due to physical properties of the CAN bus, measurements were made on different test bench layouts. One dataset was created per physical configuration, and the performance of the models on the datasets was compared to each other.

To decrease the dependency of the models' chosen hyperparameters, a parameter search was performed with Scikit-learn's GridSearchCV [54]. Grid Search evaluates the best scoring set of parameters by performing an exhaustive search over a specified parameter space. Hyperparameter optimization was performed after first evaluating the performance of the models with Scikit-learn's default parameters.

4

Results

With the large range of potential physical faults to investigate, termination faults were chosen to be the focus. The common occurrence of termination faults motivated this choice. Furthermore, due to the robustness of CAN, they often do not generate error frames, which means that they are not detected, and if they do generate error frames, the source of the error is not possible to determine. Therefore, detection of faulty termination on the physical layer provides valuable insights.

To identify termination is a general problem, and was narrowed down to a measurable definition. The termination settings of the test bench, i.e., the configurable T-connector, included 120Ω, 60Ω and 10 000Ω. The recommended configuration of a CAN bus is to use one 120Ω resistor in each end of the bus (Section 2.1.2). Therefore, configuring the test bench with 120Ω termination in each end of the bus was used to represent a correctly terminated bus. 10 000Ω termination was applied to one end to emulate a missing termination, and 60Ω was injected to indicate a double terminated end of the bus.

As a first step towards investigating whether, and to which extent, poor termination could be identified, the problem was divided into iterations, where each iteration built upon the previous, with the aim of increasing the capabilities of the models. Specifically, each iteration was required to be able to achieve the results of all the previous iterations.

The first iteration was defined as identifying a missing termination. Iteration 2 included identifying a system with double termination. Iteration 3 was defined as localizing the previous faults, i.e., to determine at which end the fault had occurred.

4.1 Iteration 1

A test bench with five nodes, two meter drop lines and a total cable length of ten meters was used. Three test bench configurations were created – 120Ω termination in both ends, 120Ω termination in the left end and 10 000Ω termination in the right end, and 10 000Ω in the left end and 120Ω in the right end. The first setup represented correct termination, and the second and third represented missing termination in one end. The second and third configurations were grouped to require the model to not only differentiate correct termination from missing termination, but also to group missing termination in the left end from missing termination in the right

Left end	Right end	Target label
120Ω	120Ω	0
10 000Ω	120Ω	1
120Ω	10 000Ω	1

Table 4.1: Iteration 1 test bench configurations with corresponding labels.

end. As the target was a binary classification, both of the incorrect termination configurations were included in the incorrect termination class. An overview of the test bench configurations and target labels of Iteration 1 is found in Table 4.1.

For each configuration of the test bench, a measurement was made from each end of the bus. Traffic was generated with the traffic generation script in Algorithm 1 with the default parameters. With 1000 frames transmitted from each node, five nodes and two measuring points, a total of 10 000 data points were generated per fault configuration. Furthermore, with a delay interval of 50 – 100 ms, one measurement took approximately 75 seconds.

To ensure class balance, half of the data points from the faulty configurations were dropped. As the ID and data was randomly generated per each frame, and external factors affecting the CAN bus such as temperature and aging of components were stable during measurements, the data points were considered to be independent and identically distributed (i.i.d.). Therefore, a data point was not considered to be related to its position in the dataset. Furthermore, the size of the dataset, and to avoid using synthetic or duplicated data, motivated undersampling instead of oversampling. If the dataset size would be considered to be a problem, more data could easily be generated with the test bench.

Preprocessing was performed according to the description in Section 3.4. From the feature extraction, the statistical mean, median and standard deviation of the offset array were added to the features calculated during preprocessing. One data point thus consisted of the features id, node, expected bit length, arbitration, mean, median and standard deviation.

From initial data exploration, it became obvious that Iteration 1 was a trivial problem. Fig. 4.1 illustrates the ID vs. the offset mean, where the labels refer to the physical configurations described in Table 4.1. The correctly terminated bus had offset mean values in approximately $(-15, 0)$ and the second class in approximately $(3, 12)$. Based on these ranges, the offset mean was by itself enough to separate the classes.

The models in Table 3.5 were trained and evaluated on the dataset. All models were able to differentiate the two classes to a degree of 100% across all evaluation metrics (Table 4.2). Fig. 4.2 shows a feature importance analysis of the Decision Tree, illustrating that the offset mean was sufficient to differentiate the classes, as indicated by the data exploration.

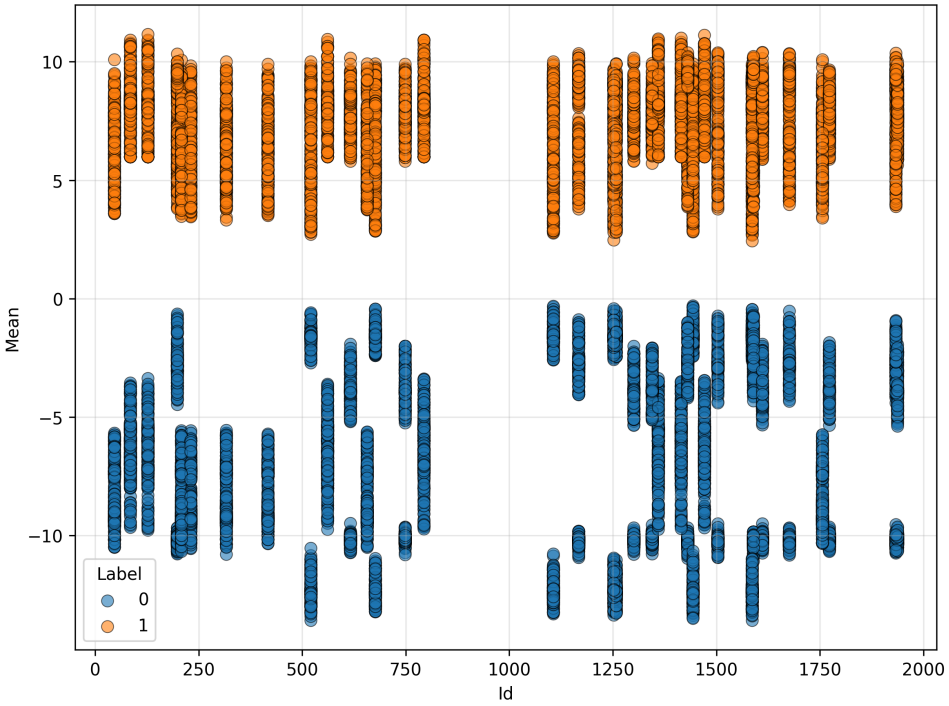

Figure 4.1: The frame ID vs. the offset mean of the Iteration 1 dataset. The correctly terminated system is illustrated in blue, and the group of missing terminations in one end of the CAN bus is illustrated in orange.

Model	Accuracy	Precision	Recall	F1-score
Logistic Regression	100	100	100	100
Decision Tree	100	100	100	100
Random Forest	100	100	100	100
Gradient Boosting	100	100	100	100
Linear SVM	100	100	100	100
K-Nearest Neighbors	100	100	100	100
Multi-Layer Perceptron	100	100	100	100
Baseline Classifier	50	25	50	33.33

Table 4.2: Iteration 1 benchmark results, measured in percentage.

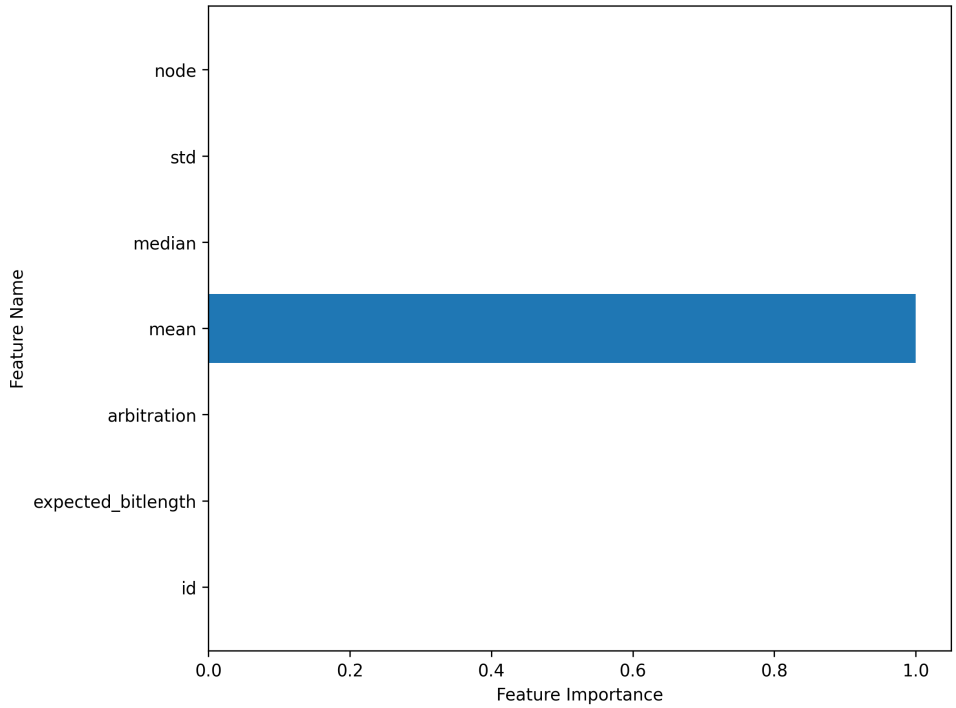

Figure 4.2: Feature Importance analysis of the Decision Tree evaluated on the Iteration 1 dataset.

Left end	Right end	Target label
120Ω	120Ω	0
10 000Ω	120Ω	1
120Ω	10 000Ω	1
60Ω	120Ω	2
120Ω	60Ω	2

Table 4.3: Iteration 2 test bench configurations with corresponding labels.

4.2 Iteration 2

The aim of Iteration 2 was to identify a CAN bus with double termination in one of the ends, in addition to differentiating a normally terminated CAN bus and a missing termination in one end, which was achieved in Iteration 1. Two new measurements were made, one with 60Ω in the left end and 120Ω in the right end, and one with 120Ω in the left end and 60Ω in the right end. The test bench configurations and their corresponding labels are found in Table 4.3. The number of nodes, cable lengths and drop line lengths were identical to the setup in Iteration 1. The measurement was performed similar to the procedure in Iteration 1, resulting in 10 000 data points per fault configuration. To achieve class balance, undersampling was performed for both faulty classes. During data exploration, an overlap of the offset mean values was indicated (Fig. 4.3).

Models were trained and evaluated with the benchmark (Table 4.4). The best per-

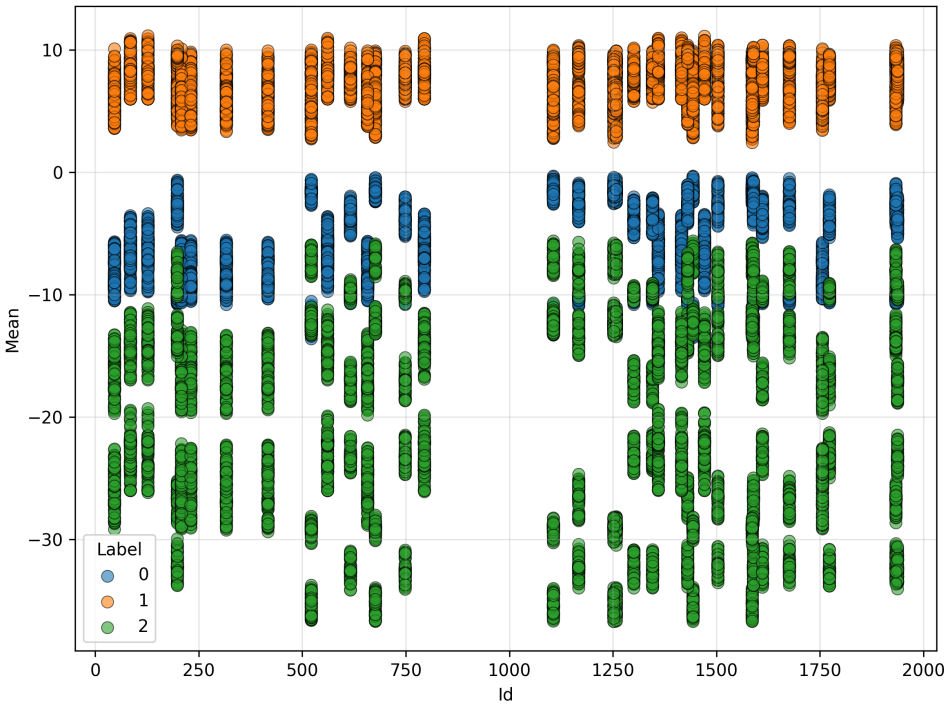

Figure 4.3: The frame ID vs. the offset mean of the Iteration 2 dataset. In addition to the Iteration 1 dataset, a third class (illustrated in green) has been added.

Model	Accuracy	Precision	Recall	F1-score
Logistic Regression	91.97	92.09	91.97	91.96
Decision Tree	99.30	99.30	99.30	99.30
Random Forest	99.47	99.47	99.47	99.47
Gradient Boosting	99.98	99.98	99.98	99.98
Linear SVM	88.72	89.03	88.72	88.68
K-Nearest Neighbors	97.55	97.56	97.55	97.55
Multi-Layer Perceptron	98.0	98.0	98.0	98.0
Baseline Classifier	33.33	11.11	33.33	16.67

Table 4.4: Iteration 2 benchmark results, measured in percentage.

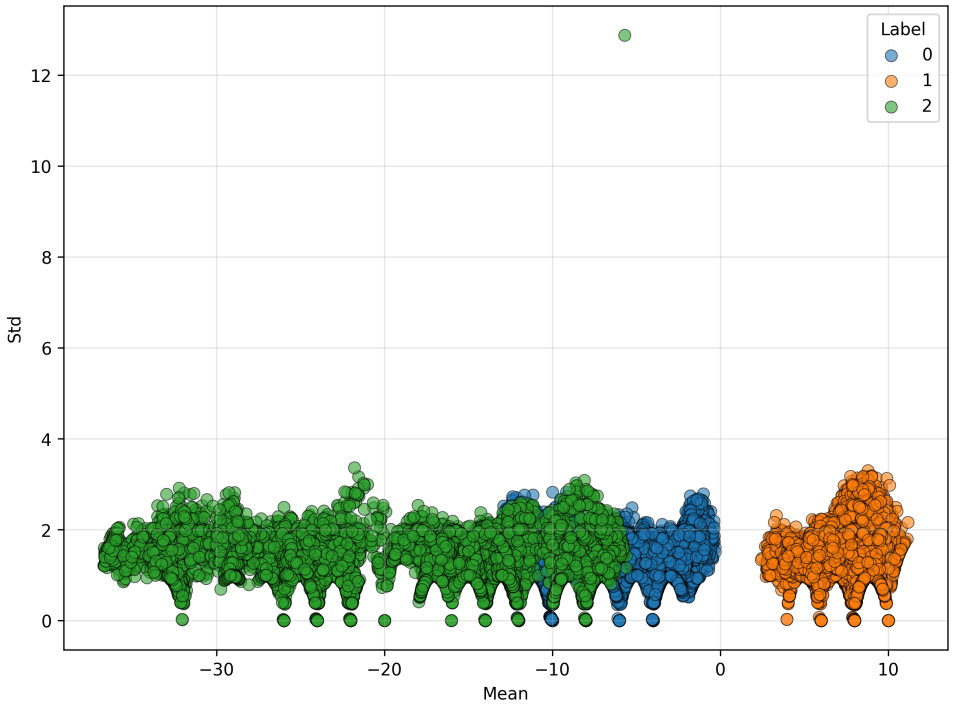

Figure 4.4: The offset mean vs. the offset standard deviation of the Iteration 2 dataset.

forming model, Gradient Boosting, achieved 99.98% across all evaluation metrics, and Random Forest achieved 99.47% across all evaluation metrics. The confusion matrix for Gradient Boosting (Fig. 4.5) illustrates that there is one false prediction made by the model. Specifically, the model predicted 0 when the true label was 2. In Fig. 4.4, an outlier of class 2 is observed. If this data point was part of the test set, it is likely this prediction the model failed to do accurately.

		Predicted class		
		0	**1**	**2**
Actual class	**0**	2000	0	0
	1	0	2000	0
	2	1	0	1999

Table 4.5: Iteration 2 Confusion Matrix for Gradient Boosting.

4.3 Iteration 3

The aim of the third iteration was to localize the faults of Iteration 2. The first two iterations had data generated for all the configurations of the test bench the third iteration aimed to investigate. Therefore, the same datasets were used in Iteration 3, but the labels were assigned differently (Table 4.6).

Left end	Right end	Target label
120Ω	120Ω	0
10 000Ω	120Ω	1
120Ω	10 000Ω	2
60Ω	120Ω	3
120Ω	60Ω	4

Table 4.6: Iteration 3 test bench configurations with corresponding labels.

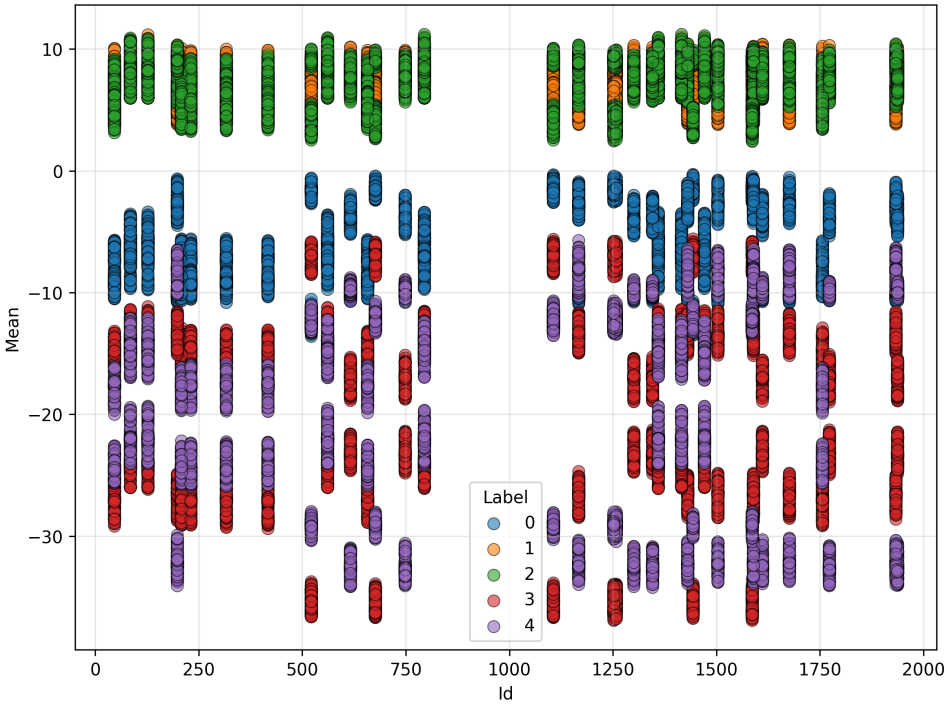

Figure 4.5: The frame ID vs. the offset mean of the Iteration 3 dataset. It contains the same data as in Iteration 2, but with one label for each fault configuration.

As each configuration represented one class in Iteration 3, the generated data was balanced for each class. With the exception of labeling, preprocessing was performed similarly to the procedure in Iteration 1 and Iteration 2.

Visualizing the frame ID vs. the offset mean, one could see that there were big overlaps, indicating that the data was no longer trivial to separate (Fig. 4.5).

Models were trained and evaluated similar to previous iterations. The results of the benchmark is presented in Table 4.7. The evaluation metrics dropped several percentage points for all models compared to their performance in Iteration 2, and for Logistic Regression and Linear SVM, the accuracy dropped significantly.

The best performing model was Gradient Boosting with an accuracy of 91.87%. Table 4.8 illustrates that it was target 1 and 2, i.e., identifying the end of missing termination, the model failed to predict accurately. As seen in Fig. 4.5, the overlap of offset means in class one and two are almost perfect, indicating difficult decision

Model	Accuracy	Precision	Recall	F1-score
Logistic Regression	58.70	57.65	58.7	57.95
Decision Tree	89.02	89.04	89.02	89.01
Random Forest	90.88	90.89	90.88	90.88
Gradient Boosting	91.87	91.89	91.87	91.87
Linear SVM	53.91	56.67	53.91	54.33
K-Nearest Neighbors	84.90	84.95	84.90	84.89
Multi-Layer Perceptron	80.3	80.47	80.3	80.2
Baseline Classifier	20.0	4.0	20.0	6.67

Table 4.7: Iteration 3 benchmark results, measured in percentage.

making based on those features. Random Forest achieved an accuracy of 90.88%. Fig. 4.6 showcases a split between all features except arbitration. In particular, the most important features included both the mean and the median, indicating that simple decisions were not sufficient in distinguishing the classes.

		Predicted class				
		0	**1**	**2**	**3**	**4**
Actual class	**0**	2000	0	0	0	0
	1	0	1557	443	0	0
	2	0	367	1633	0	0
	3	0	0	0	2000	0
	4	3	0	0	0	1997

Table 4.8: Iteration 3 Confusion Matrix for Gradient Boosting.

To explore why no model were able to achieve optimal, or close to optimal, accuracy on the third iteration problem, the following sections are presented. First, the models' hyperparameters were investigated by performing a parameter search over a parameter space potentially affecting the results. Second, the feature extraction process was evaluated to assess whether the offset array was too heavily aggregated. Third, an analysis of potential sequential patterns in the offset array was performed by developing and analyzing a 1D-CNN. Fourth, in-depth data exploration was performed to identify whether class 1 and class 2 were of too similar shape to distinguish to a high degree. Last, how the size of the CAN bus, meaning the cable lengths, the stub lengths, and the number of nodes, affected the results were investigated.

4.4 Hyperparameter Optimization

A hyperparameter optimization was performed on each model with a 5-fold cross validation of the training set. As explained in Section 2.2.2, this split the training set X into five equally sized partitions $X_1, ..., X_5$, and for each fold i ($1 \leq i \leq 5$),

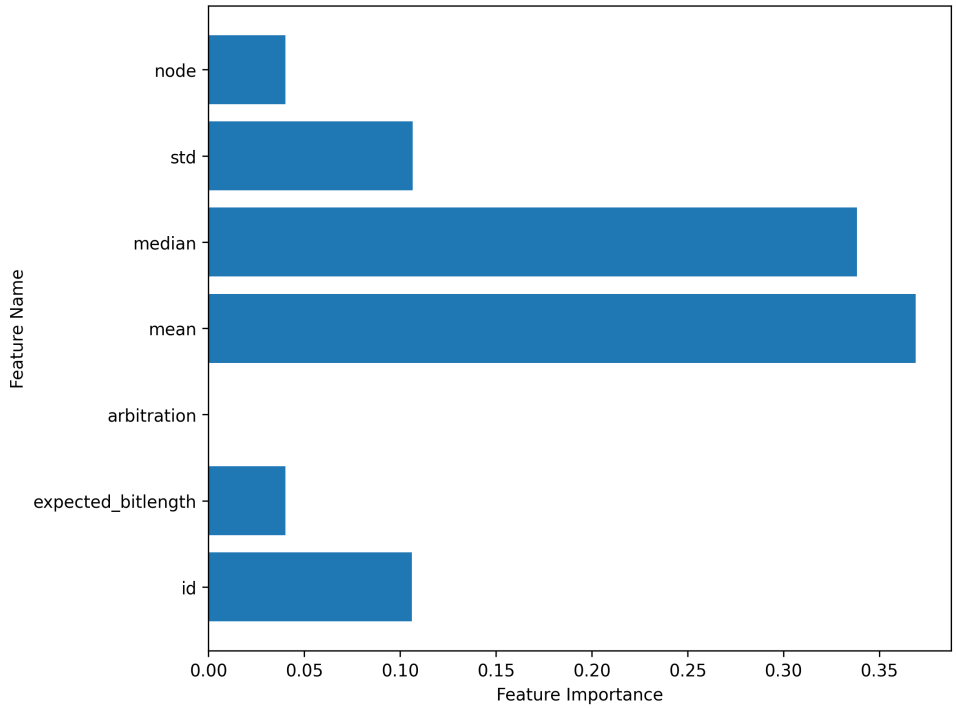

Figure 4.6: Feature Importance analysis of Random Forest evaluated on the Iteration 3 dataset.

the model was trained on $X \setminus X_i$ and evaluated on X_i. For each split and evaluation, an accuracy a_i was returned. The result of the cross validation was thus the mean value of $\{a_1, ..., a_5\}$, henceforth referred to as the *mean test score*.

A parameter space, i.e., a map from a model parameter to a set of inputs, was defined for chosen hyperparameters of the models. For parameters not present in the parameter space, the model's default values were used, unless stated otherwise.

The hyperparameter optimization was performed with Scikit-learn's Grid Search over the parameter space. Grid Search works by training and evaluating a model on each combination of the Cartesian product of the parameter space. The tuple of parameters achieving the highest mean test score was applied to the model, which was trained on the complete training set and evaluated on the test set.

For several models, randomness determines the learning procedure and potentially the outcome of the trained parameters. Therefore, randomness disables true comparison of the hyperparameter optimization, as random elements can benefit the model and its relation to the dataset. For a pseudo-random number generator, a seed, or a "random state", can be defined to enable deterministic behavior of random features. Therefore, to enable reproducibility and minimize random elements affecting the cross validation results, all models with random elements were initialized with a similar random state.

Due to the combinatorial growth of parameter tuples in the parameter space, a small set of parameters must be chosen for each hyperparameter, and for models with

Parameter	Parameter Space	Default
C	0.01, 0.05, 0.1, 0.5, 1, 5, 10	1
l1_ratio	0, 0.25, 0.5, 0.75, 1	0
solver	lbfgs, saga	lbfgs

Table 4.9: Parameter space for Logistic Regression.

Parameter	Parameter space	Default
criterion	gini, entropy, log_loss	gini
splitter	best, random	best
min_samples_split	2, 5, 10, 20	2
min_samples_leaf	1, 2, 5, 10	1

Table 4.10: Parameter space for Decision Tree.

many input parameters, most parameters must be initialized with default values. The choice of hyperparameters to evaluate was based on properties of the model architecture potentially affecting the outcome of its performance, such as learning procedure, the model's relation to the dimensionality in the data, and inference. For numerical values, a range was defined around the default value to infer which direction the model benefited from, either increasing or decreasing the default value. For certain models and hyperparameters, a hyperparameter can restrict the learning capabilities. In those cases, the parameter space was defined above the default value. Furthermore, in all parameter spaces, the default parameters were included to investigate whether the optimized parameters outperformed the default parameters.

4.4.1 Logistic Regression

In Table 4.9, the parameter space for Logistic Regression is presented. The models were evaluated with a maximum number of iterations of 1000 to limit early stopping. The mean test score varied in $(57.15\%, 58.06\%)$. The best performing parameters were C=5.0, l1_ratio=0, solver=lbfgs, meaning that C was the only parameter that changed from the default parameters. Evaluating the model on the test set with C=5.0 achieved the exact same results as in the benchmark (Table 4.7), indicating that the hyperparameters were not the primary reason for its poor results.

4.4.2 Decision Tree

A parameter search was performed over the parameters in Table 4.10. Mean test score varied in the range $(87.12\%, 89.05\%)$, indicating a small gain of optimizing the hyperparameters. The best parameters included min_samples_leaf = 1, min_samples_split = 10, splitter = random, criterion = gini, with similar results for min_samples_leaf = 2 and the other two criteria.

Running the model with updated parameters according to the parameter search, an accuracy of 88.93% was achieved. The result was a negligible decrease compared to the benchmark results, which indicated that the model was not restricted by the

Parameter	Parameter space	Default
n_estimators	10, 50, 100, 200, 500, 1000	100
criterion	gini, entropy, log_loss	gini
min_samples_split	2, 5, 10, 20	2
min_samples_leaf	1, 2, 5, 10	1

Table 4.11: Parameter space for Random Forest.

Parameter	Parameter Space	Default
learning_rate	0.01, 0.05, 0.1, 0.5, 1	0.1
min_samples_leaf	10, 20, 40, 60, 100	20
l2_regularization	0, 0.5, 1	0

Table 4.12: Parameter space for Gradient Boosting.

default hyperparameters.

4.4.3 Random Forest

The hyperparameters of Random Forest were evaluated with the parameter space in Table 4.11. Mean test scores were found in the range (88.76 %, 91.03 %). The best parameters were min_samples_leaf = 1, min_samples_split = 5, n_estimators = 500. The model was indifferent to the criterion.

Evaluating the model on the test set, an accuracy of 90.98% was achieved. Compared to an accuracy of 90.88% with default parameters, it was a negligible gain, and the hyperparameters were not considered to be decisive.

4.4.4 Gradient Boosting

The hyperparameters of Gradient Boosting were optimized over the space described in Table 4.12, achieving accuracy scores in the range $(44.51\%, 91.95\%)$. Such a large range indicates a strong dependency of the hyperparameters, and in particular, there were sets of hyperparameters performing very poorly. The best parameters proved to be learning_rate = 0.05, min_samples_leaf = 40, l2_regularization = 0, and the worst performing parameters were learning_rate = 1, min_samples_leaf = 10, l2_regularization = 0.

Training and evaluating Gradient Boosting with the updated hyperparameters, an accuracy of 92.27% was achieved. Given that the mean score ranges from 44.51 to 91.95, the hyperparameters seemed to have a significant impact on Gradient Boosting's performance on this dataset. A smaller learning rate and a larger amount of samples per leaf seemed to work best, suggesting that the model suffered from biases. A larger learning rate likely led to an overshoot of optimal solutions and a too small number of samples per leaf led to an overfit model. However, as the performance of the model with optimized parameters only increased by 0.40% points in comparison to the default parameters, it could not be determined that the model's performance could be optimized with different parameters.

Parameter	Parameter Space	Default
penalty	l1, l2	l2
loss	hinge, squared_hinge	squared_hinge
C	0.01, 0.05, 0.1, 0.5, 1, 5, 10	1
fit_intercept	True, False	True

Table 4.13: Parameter space for Linear SVM.

Parameter	Parameter Space	Default
n_neighbors	3, 5, 7, 9	5
weights	uniform, distance	uniform
algorithm	auto, ball_tree, kd_tree, brute	auto
p	1, 2, 3	2
metric	euclidean, minkowski	euclidean

Table 4.14: Parameter space for K-Nearest Neighbors.

4.4.5 Linear Support Vector Machine

The Linear SVM model was evaluated with the parameter space in Table 4.13, achieving mean test scores in the range $(29.51\%, 53.88\%)$. The best performing parameters were penalty = l2, loss = squared_hinge, C = 10, and the results were indifferent to fit_intercept. Compared to the default parameters, C was the exception.

Evaluating the model with C=10 on the test set, an accuracy of 53.84% was achieved, a decrease of 0.07 % points, which is negligible. The fact that Linear SVM continued to perform poorly with optimized parameters suggested that the hyperparameters were not the cause for its poor performance.

4.4.6 K-Nearest Neighbors

The hyperparameter optimization was performed with the parameter space in Table 4.14, achieving mean test scores in the range $(90.94\%, 91.57\%)$. The best performing parameters included n_neighbors = 5, weights = distance, algorithm = brute, p = 1, and metric = minkowski. Compared to the default parameters weights = uniform, algorithm = auto, p = 2, a potential improvement was provided. Especially since weights and p were changed, which are decisive for the decision boundary of the model.

Running the model with the updated hyperparameters on the test set, an accuracy of 88.13% was achieved. A substantially lower accuracy compared to the highest mean score achieved during Grid Search could indicate that the model performed exceptionally on certain splits during the cross validation, or that the model was overfit on the training set.

However, the model achieved an increase of 3.23% points compared to the benchmark results. The change of the distance metric from uniform to inverse introduces a bias in the model of a preference to closer neighbors, which indicated that closer

Parameter	Parameter Space	Default
hidden_layer_sizes	(100,), (100,50,), (100,50,25)	(100,)
alpha	0.00001, 0.0001, 0.001	0.0001
learning_rate	constant, invscaling, adaptive	constant

Table 4.15: Parameter space for Multi-Layer Perceptron.

Parameter	Parameter Space	Default
strategy	most_frequent, prior, stratified, uniform	prior

Table 4.16: Parameter space for the Baseline Classifier.

neighbors were of greater importance compared to an equal vote among the k nearest neighbors.

4.4.7 Multi-Layer Perceptron

A parameter search was performed over the parameter space in Table 4.15. 78.08% accuracy, the highest mean test score, was achieved with hidden_layer_sizes = (100, 50, 25,), alpha = 0.0001. The results were indifferent to the learning_rate parameter. The lowest accuracy 51.32% was achieved with hidden_layer_sizes = (100,) and alpha = 0.00001, indicating an underfit model.

Evaluating the model with hidden_layer_sizes = (100, 50, 25,) and alpha = 0.0001 on the test set, an accuracy of 85.02% was achieved. Compared to the default parameters, which achieved 80.3% on the test set, this was an increase of 4.72% points, indicating that network complexity was an important factor. While this increase was the largest of all the models, Multi-Layer Perceptron was performing worse with optimized hyperparameters than the best performing models with default parameters. This could indicate that MLP was not an ideal model for this dataset, or that it could only outperform the tree-based models given enough complexity and training.

4.4.8 Baseline Classifier

A "parameter optimization" was performed on the Baseline Classifier to ensure that its poor result was not due to the prediction strategy. The strategies are presented in Table 4.16 and include all viable strategies of Scikit-learn's Dummy Classifier.

All strategies got close to 20% accuracy and 0% standard deviation on the cross validation scores, but an accuracy of 20.16% was achieved with strategy = stratified. Evaluating on the test set, the most frequent and prior strategy achieved a 20% accuracy, stratified strategy got 19.86% and uniform 20.43%.

4.4.9 Analysis

The deviation of accuracy in percentage points for each model is presented in Table 4.17. A marginal change was observed for all models except K-Nearest Neighbors

Model	Mean Score Range	Accuracy	Increase
Logistic Regression	(57.15, 58.06)	58.70	0
Decision Tree	(87.12, 89.05)	88.93	-0.90
Random Forest	(88.76, 91.03)	90.98	0.10
Gradient Boosting	(44.51, 91.95)	92.27	0.40
Linear SVM	(29.51, 53.88)	53.84	-0.07
K-Nearest Neighbors	(90.94, 91.57)	88.13	3.23
Multi-Layer Perceptron	(51.32, 78.08)	85.02	4.72
Baseline Classifier	(19.98, 20.16)	19.86	-0.14

Table 4.17: Results from the models evaluated on the test set with optimized hyperparameters. The results from the test set evaluation is presented in the Accuracy column. Increase refers to the Iteration 3 benchmark accuracy results (Table 4.7). Mean Score Range and Accuracy are measured in percentages while Increase is measured in percentage points.

(3.23 % points) and Multi-Layer Perceptron (4.72 % points), which was considered to be substantial. K-Nearest Neighbors updated the parameters weights, algorithm, p, and metric, and gained a significant accuracy increase on the test set. By changing its distance metric from uniform to inverse, and by using the Manhattan distance instead of the Euclidean distance, the model's decision boundary was changed. These changes indicated that the interpretation of the distance of offset values was of importance in training and inference, as closer neighbors were more important in classifying the faults. The largest accuracy increase was achieved with the optimized hyperparameters of Multi-Layer Perceptron. Such an increase was expected, as the default hyperparameters of MLP only has one hidden layer. The optimal hyperparameters included a three hidden layer network, and increasing the complexity of the neural network enables more complex learning. However, neither KNN nor MLP outperformed the tree-based models with default parameters.

Logistic Regression and Linear SVM, which performed poorly on Iteration 3, continued to perform poorly with optimized parameters. This result indicated an architectural problem of the models rather than a hyperparameter issue. Random Forest and Gradient Boosting gained a very small accuracy increase, a result that was considered to be negligible. However, they remained as the best performing models. Most importantly, none of the models achieved a close to perfect result on the test set evaluation, indicating that the hyperparameters were not the primary reason for the non-optimal results.

The baseline classifier remained at approximately 20% accuracy on the test set regardless of decision strategy. While this result was expected, it illustrated that the default decision strategy did not affect the accuracy of the baseline, indicating a balanced dataset.

The hyperparameter optimization was performed to infer whether Scikit-learn's default parameters were insufficient or inaccurate for this particular problem and dataset. However, the summarized results indicated that the default parameters were not the primary reason for the models' inability to localize a missing termina-

tion in one end.

However, the results of the hyperparameter optimization are not conclusive. As discussed in Section 4.4, a subset of hyperparameters and potential parameters to include in the parameter space must be chosen due to the combinatorial growth in the number of parameter tuples. To address this, the parameters and parameter spaces were chosen based on the hyperparameters' impact on training and inference. It is not concluded that there does not exist corner cases of parameter combinations performing very well on this problem.

4.5 Feature Extraction

To investigate whether the statistical features of the offset array excluded relevant patterns in the data, the feature extraction procedure was reconsidered. For every new feature extraction procedure, the features computed during preprocessing described in Table 3.2 were included.

4.5.1 Additional Statistical Features

A first approach of increasing the expressiveness of the data was to include additional statistical features of the offset array. For each data point, in addition to the mean, median, and standard deviation, statistical measures such as percentiles, modal value, and range were added. The full set of statistical measures used is presented in Table 4.18. A new dataset was computed with the additional features, based on the same test bench setup described in Section 4.3. The models were trained and evaluated on the dataset with default parameters.

Most models showed improved metrics with these additional features. Logistic Regression achieved an accuracy of 64.72%, Linear SVM 58.25%, and Multi-Layer Perceptron 85.76%, all several percentage points higher compared to the Iteration 3 benchmark results. The improved performance of these model made sense as the additional features provided the models more informative representations to learn. Since linear models and neural networks rely heavily on the expressiveness of the input space, features that further capture the shape of the data can enhance their ability to separate closely related classes.

The tree-based models, however, showed only marginal accuracy deviations from the extra statistical features. Decision Tree achieved an accuracy of 88.94%, Random Forest 90.50%, and Gradient Boosting 92.69%. For Decision Tree and Random Forest, this was a slight decrease, and for Gradient Boosting, it was a slight increase compared to the Iteration 3 results. Decision trees inherently perform feature selection through training and can construct non-linear splits of the data that could approximate many of the extended statistical features, as several of them were related to each other. As a result, the performance of these models remained essentially the same, and the new features did not seem to provide much more information beyond what the tree-based models derived from the original dataset.

K-Nearest Neighbors exhibited a small decrease in performance as it achieved 82.89%

Measure	Description
Mean	Offset Mean
Median	Offset Median
Standard Deviation	Offset Standard Deviation
Variance	Offset Variance
Minimum	Offset Minimum Value
Maximum	Offset Maximum Value
Range	Maximum - Minimum
Count	Number of Offsets
P5	5th Percentile Offset Value
P10	10th Percentile Offset Value
P25	25th Percentile Offset Value
P75	75th Percentile Offset Value
P90	90th Percentile Offset Value
P95	95th Percentile Offset Value
Inter-Quartile Range	P75 - P25
Mode Value	Offset Modal Value
Mode Count	Offset Modal Count
Mean Absolute Deviation	Average Distance from Mean
Median Absolute Deviation	Average Distance from Median

Table 4.18: The extended statistical features.

accuracy, indicating that it suffered from the higher dimensionality. Similar to the tree-based models, a probable explanation lies in the correlation of the added dimensions to the existing ones.

The hyperparameter optimization was performed again with similar parameter spaces described in Section 4.4. The mean accuracy range of the cross validation, test set results, and the comparison to the accuracy of the default parameters are presented in Table 4.19. The majority of the models did not perform significantly better or worse, and the results could be written off as statistical noise. The exceptions were again K-Nearest Neighbors and Multi-Layer Perceptron. For K-Nearest Neighbors, the optimal parameters were metric = minkowski, n_neighbors = 3, p = 1, the choice of algorithm did not matter. With updated hyperparameters and evaluated on the test set, K-Nearest Neighbors achieved an accuracy of 87.42%. The best performing parameters of Multi-Layer Perceptron were hidden_layer_sizes = (100, 50, 25) and alpha = 0.00001, achieving an accuracy of 88.44%.

Compared to the default parameters, KNN and MLP gained the largest increase, similarly to the results in Section 4.4. This, once again, showed that adjusting the default hyperparameters was of importance for these models. KNN showed a bigger increase and MLP a smaller increase compared to previous parameter optimizations which could be explained by the increased complexity of the input data. However, Gradient Boosting, the best performing model, gained a decrease in accuracy of 0.68 % points. Neither with or without parameter optimization, a model accurately predicted which end of the bus contained the missing termination fault to a 100%

Model	Mean Score Range	Accuracy	Increase
Logistic Regression	(62.85, 63.26)	64.59	-0.13
Decision Tree	(87.2, 88.96)	88.34	-0.60
Random Forest	(88.45, 90.37)	90.70	0.20
Gradient Boosting	(44.94, 92.73)	92.01	-0.68
Linear SVM	(38.05, 58.85)	58.35	0.10
K-Nearest Neighbors	(89.41, 90.96)	87.42	4.53
Multi-Layer Perceptron	(58.14, 82.68)	88.44	2.68
Baseline Classifier	(20.0, 20.16)	19.86	-0.14

Table 4.19: Hyperparameter optimization results on the extended features dataset. The results from the test set evaluation is presented in the Accuracy column. Increase refers to the model's change in accuracy with default parameters. Mean Score Range and Accuracy are measured in percentages while Increase is measured in percentage points.

accuracy.

4.5.2 Offset Array

To further investigate the loss of information from aggregating the offset array with statistical features, the statistical feature extraction procedure was reconsidered. Furthermore, some models performed worse on the extended statistical features, possibly since several of them were related to each other. Therefore, a dataset was created with the offset vector as features, in addition to the features computed during preprocessing (Table 3.2). As the offset arrays were of different length for each frame, the array was zero-padded to ensure similar shape. In particular, the length of the offset array varied in the range (17, 33).

The models were trained and evaluated with default parameters. The tree-based model continued to perform best on the dataset. However, for the first time, Random Forest achieved a higher accuracy (92.41%) than Gradient Boosting (92.16%). For Random Forest, this was an increase of 1.53 % points compared to the Iteration 3 benchmark and 1.91 % points compared to the extended features dataset, which showed that the offset array features were of slightly greater value for this model.

For Gradient Boosting, the result implied an accuracy increase of 0.29 % points compared to the Iteration 3 benchmark, but a decrease of 0.53 % points compared to the extended features dataset, which showed that Gradient Boosting was less affected by the choice of features. The linear models did not only perform poorly, but worse in comparison to Iteration 3. Logistic Regression achieved an accuracy of 54.89%, a decrease of 3.81 % points, and Linear SVM got 49.68% accuracy, a decrease of 4.23 % points compared to its results on the Iteration 3 benchmark. This result was likely due to the high dimensionality of the data. Without aggregation, the feature space might be too erratic and linear patterns too obscure.

KNN and MLP also worsened their performance in comparison to their results in Iteration 3. For KNN, an accuracy of 82.32% was achieved, a decrease of 2.58

Model	Mean Score Range	Accuracy	Increase
Logistic Regression	(54.95, 55.23)	55.09	0.20
Decision Tree	(85.82, 89.81)	90.23	1.52
Random Forest	(87.43, 92.67)	92.66	0.25
Gradient Boosting	(46.08, 93.51)	93.55	1.39
Linear SVM	(34.29, 49.98)	39.40	-10.28
K-Nearest Neighbors	(80.01, 86.70)	88.50	6.18
Multi-Layer Perceptron	(53.34, 78.19)	83.95	5.59
Baseline Classifier	(19.98, 20.16)	19.86	-0.14

Table 4.20: Hyperparameter optimization results on the offset array dataset. The results from the test set evaluation is presented in the Accuracy column. Increase refers to the offset array benchmark accuracy results. Mean Score Range and Accuracy are measured in percentages while Increase is measured in percentage points.

% points. This indicated that most offsets were similar to each other, making it difficult for KNN to separate the data. MLP got an accuracy of 78.36%, a decrease of 1.94 % points. As the offset array provided more complexity in the input data, this result was unexpected. However, considering the low complexity of the network with default parameters, the result indicated an underfit model.

The hyperparameters were optimized by running the parameter search over the parameter space defined in Section 4.4. In comparison to the previous optimizations, more drastic accuracy changes were observed. This made sense considering the offset array dataset contained more dimensions, and potentially more complex patterns to learn. The results of the hyperparameter optimization is presented in Table 4.20.

Linear SVM got a much lower accuracy on the test set with the parameters C=0.05, fit_intercept = False, loss = squared_hinge, penalty = l2, resulting in a drop of 10.28% points. This result indicated that the model overfit to the training data. Similar to previous results, the Multi-Layer Perceptron benefited from a higher complexity network, and scored an accuracy of 83.95% on the test set with hidden_layer_sizes = (100,50,25,) and alpha = 0.00005, a gain of 5.59% points compared to the default parameters. With l2_regularization = 0.5, learning_rate = 0.05, min_samples_leaf = 10, max_leaf_nodes = None, Gradient Boosting outperformed Random Forest with an accuracy of 93.55% to 92.66%, where Random Forest got its highest accuracy with min_samples_leaf = 1, min_samples_split = 2, n_estimators = 1000. K-Nearest Neighbors showed a drastic increase of accuracy with algorithm=kd_tree, metric=minkowsi, n_neighbots=3, p=1, weights=distance. In particular, it decreased the number of neighbors to three, and increased the importance of closer neighbors. This result indicated the importance of tighter decision boundaries for the model as the dimensionality was increased.

In conclusion, the offset array dataset enabled an accuracy increase for the best performing models. However, the problem of differentiating class 1 and 2 remained.

Model	Mean Score Range	Accuracy	Increase
Logistic Regression	(62.46, 63.04)	62.11	-2.24
Decision Tree	(88.05, 90.08)	90.15	0.14
Random Forest	(89.13, 92.22)	92.44	0.10
Gradient Boosting	(42.07, 94.35)	93.67	-0.07
Linear SVM	(30.80, 58.76)	50.17	-8.28
K-Nearest Neighbors	(80.02, 88.45)	89.28	5.53
Multi-Layer Perceptron	(61.03, 83.96)	89.55	4.41
Baseline Classifier	(20.0, 20.16)	19.86	-0.14

Table 4.21: Hyperparameter optimization results on the dataset with extended statistical features and the offset array combined. The results from the test set evaluation is presented in the Accuracy column. Increase refers to the extended features combined with offset array benchmark accuracy results. Mean Score Range and Accuracy are measured in percentages while Increase is measured in percentage points.

4.5.3 Additional Statistical Features and Offset Array

Seeing that the best performing models increased the accuracy of the test set evaluation in both Section 4.5.1 and Section 4.5.2, an attempt to use a combination of the additional statistical features and the offset array features was performed. The models were evaluated with default parameters.

Gradient Boosting achieved an accuracy of 93.74%, an increase of 1.87 % points compared to the Iteration 3 benchmark results. This increase can be considered to be substantial, and furthermore, it was the highest accuracy Gradient Boosting achieved across all datasets. This result indicated that the model was able to combine dimensions of the two datasets to enable differentiation of which end of the bus that had no termination. Random Forest achieved an accuracy of 92.34%, which was a negligible but slight decrease to its performance on the offset array dataset. Both Logistic Regression and Linear SVM performed about the same compared to the extended features dataset, which suggested that they disregarded the combination of the datasets and focused on the dataset where linear relationships were easier to find. KNN and MLP also showcased small changes with default parameters compared to the other two datasets, suggesting that the combination of features did not affect them.

A parameter search was performed over the parameter spaces defined in Section 4.4, and the results of the optimization is presented in Table 4.21. The linear models showcased a low accuracy and overfit to the training data, leading to a decrease in accuracy with optimized parameters. Similar to previous results, KNN improved due to having a more beneficial distance metric, and MLP improved due to having increasing the number of hidden layers, enabling more complex learning of the non-linear feature space. The tree-based models were still performing best, but were barely affected by the parameter optimization. Evaluating Gradient Boosting on the test set, a 93.67% accuracy was achieved with l2_regulatization = 0.5, learning_rate = 0.1, min_samples_leaf = 40, and max_leaf_nodes = None, which was a slight

decrease compared to the default parameters.

4.5.4 Analysis

The overall results from the revised feature extraction procedure showed that the initial statistical features aggregated the data too heavily. However, there were no set of features that enabled a model to distinguish the missing termination faults to a perfect degree.

The best accuracy for all models, feature sets and parameter optimizations was achieved by Gradient Boosting (93.74%) on the combination of the additional statistical features and the offset array, and by using the default parameters of the model. The confusion matrix of Gradient Boosting on this dataset with default parameters is presented in Table 4.22. The matrix illustrates that while the performance of the model was increased, the issue of localizing a missing termination persisted.

		Predicted class				
		0	**1**	**2**	**3**	**4**
Actual class	**0**	2000	0	0	0	0
	1	0	1673	327	0	0
	2	0	295	1705	0	0
	3	1	0	0	1998	1
	4	1	0	0	1	1998

Table 4.22: Confusion Matrix of Gradient Boosting on the additional statistical features and the offset array combined.

4.6 Sequential Patterns

Aggregating data into statistical features discards sequential and local patterns potentially present in the data. To investigate if the offset array carried important sequential information, a 1D-CNN classifier with three convolutional layers and two fully connected layers was developed.

The model was trained and evaluated on only the offset array, excluding the preprocessing features (Table 3.2), in three different configurations. The first contained the original order of the offsets. In the second, the offset array was shuffled, and in the third, it was sorted by ascending order. The 1D-CNN was trained for 50 epochs and achieved the following results. Original ordering of the offset array yielded 85% accuracy. Sorting the offset array by ascending order yielded 74% accuracy, and shuffling the offset array yielded a 74% accuracy.

Since the original order resulted in 11% points higher accuracy than the shuffled and sorted order, a sequential pattern in the offset array was indicated. However, if

the order was vital to classify a CAN frame, the shuffled and sorted ordering should have yielded drastically different results. In contrast, the results of the 1D-CNN on the manipulated orders of the offset array indicated that it was able to make accurate predictions based on the nominal values of the offsets.

To evaluate the performance of the 1D-CNN on only the offset data, a benchmark with the other models was performed on the same datasets. In comparison to the other models, the 1D-CNN performed slightly worse on the original order and slightly better on the random and shuffled order. With default parameters, Random Forest achieved the highest accuracy throughout the three datasets. On the ordered array, Random Forest achieved an accuracy of 85.98%. On the shuffled array, it scored a 71.65% accuracy, and on the sorted array 71.71%.

4.6.1 Sequential Feature Extraction

As the 1D-CNN performed better on the original order of the offsets, an investigation was performed whether sequential features could be extracted with the 1D-CNN and used by the best performing benchmark models.

Using the combined dataset of additional statistical features and the offset array (with preserved ordering), the 1D-CNN was trained on the offset array as described in Section 4.6. Thereafter, the embeddings of the CNN were extracted by forwarding test samples through the CNN and discarding the last fully connected layer (the classification layer). From the first fully connected layer, the dimensionality of the network was reduced to eight features. Finally, Random Forest and Gradient Boosting with default parameters were trained on the additional statistical features in combination with the extracted CNN features.

Random Forest achieved an accuracy of 90.40%, and Gradient Boosting an accuracy of 92.10%, an increase for both models compared to the results of the offset array described in Section 4.6. This was expected, as the preprocessing features were essential to determine the transmitting node and which end of the bus the measurement was made from. However, the accuracy was worse compared to the models' performance on the extended statistical features and offset array combined. This result indicated that the models benefited from training on the offset array, rather than the features the 1D-CNN extracted from it.

4.6.2 Analysis

The results from the investigation suggested that there were sequential patterns present in the offset array. However, the tree-based models outperformed the 1D-CNN and achieved better results on the offset array compared to the features the 1D-CNN extracted from it.

Investigating the dataset, the number of edges in a frame varied from 17 to 33, with a median of 25. Deep learning models excel at finding patterns in high-dimensional data, where traditional machine learning models suffers from computational cost and learning difficulties due to sparseness of data points. However, in lower dimensions,

traditional machine learning algorithms often outperforms deep learning architectures, as the feature space is not complex enough to exploit with a complex model. Having only 17 to 33 dimensions to extrapolate sequential patterns from was likely the explanation for the 1D-CNN:s inability to extrapolate patterns from the offset array which value exceeded that of the offset array itself. The fact that Random Forest and Gradient Boosting performed worse on the sequential features compared to the original offset array suggested that the dimensionality of the data was too low.

4.7 Data Analysis

Having examined hyperparameter optimization, the feature extraction process and sequential patterns in the offset array, no model performed perfectly. Specifically, localizing the missing termination remained the difficult case. To determine whether the two classes were too similar to distinguish, and to gain further knowledge of the behavior of the recessive edge offsets, an in-depth data analysis was performed.

Fig. 4.7 illustrates the offset mean vs. the offset standard deviation sampled from the right end of the CAN bus during normal termination, i.e., 120Ω on each end. The displayed dataset consists of 5000 CAN frames, and the colors represents data from each ECU on the network. The figure demonstrates how the physical distance of the measuring interface and the ECU affects the delay of the recessive edge. A negative offset value indicates an edge occurring earlier than the expected bit length. ECU:s physically closer to the interface, such as number 4 and 5, had offset mean values closer to the expected value, while the ones further away deviate from the expected. Furthermore, the visualization indicates a wide distribution within the class, which was also true for the fault configurations.

The data formed arch-like structures, which could, in part, be explained by quantization (Section 3.2.2). Quantization results in offsets falling into discrete segments two clock cycles apart. Therefore, if the edges are sampled consistently throughout a frame, the offset mean is an integer with no variance. If the offsets are sampled within several segments, the offset mean falls between different discrete categories, and the standard deviation is non-zero. If the samples are equally distributed within the discrete range, the mean falls in between two values and the standard deviation is at its highest.

Evaluating a visualization where fault, transmitting node and measuring point has been isolated enables analysis of the correlation of offset mean values and physical fault. Fig. 4.8 shows the offset mean vs. the offset standard deviation transmitted from one ECU for each fault configuration described in Table 4.6. The data exhibits consistent patterns within each group, and the groups are mostly separated. As the figure only presents data transmitted from one ECU, this suggests that the ID is important when classes are indistinguishable based on the statistical mean. Furthermore, the similarities between missing termination in one end of the bus (orange and green) demonstrate that the two classes have similar mean and standard deviation values, which could explain why no model could completely distinguish

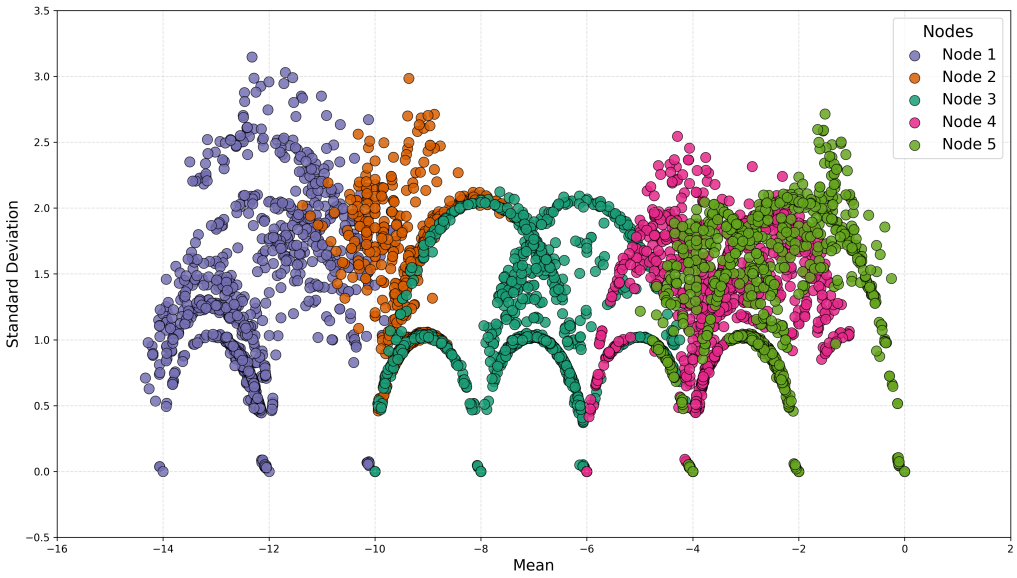

Figure 4.7: The offset mean vs. the offset standard deviation of CAN frames transmitted from different nodes, measured with 120Ω termination in both ends. The measurement was made from the right end of the bus.

those two configurations.

Similarly to Fig. 4.8, the colors in Fig. 4.9 refer to the same fault configurations. The image to the left displays CAN frames transmitted from the fifth ECU sampled from the left end, and the image to the right displays CAN frames transmitted from the first ECU sampled from the right end. The classes have essentially switched positions, showing that the offset mean of a frame depends on where the measurement is made. This indicates that the combination of the node and ID features is required to make the classification.

Fig. 4.10 visualizes the offset mean vs. the offset standard deviation for a correctly terminated system across different test bench layouts. The image to the left illustrates frames transmitted from all ECU:s. A visible overlap can be seen between all the layouts, but there is also a shift to the left for the larger systems. The image to the right showcases the same classes, but only contains frames transmitted from the third ECU. As all data points comes from a correctly terminated system, this illustrates the occurrence of domain shift, i.e., that the distribution of the offset mean values shift based on the network size. From this, one can conclude that different CAN bus sizes result in different recessive transition delays. Furthermore, as the size of the offset mean range is increased with the system size, one can conclude that the offsets are more easily distinguishable as the system is enlarged. Specifically, which ECU the data originates from should be easier to distinguish in a larger system, as the groups are drawn further apart. Furthermore, since fault classes in one system likely overlaps with normal termination in another system, a model trained on one system would perform poorly on another system, even if it accurately classifies the faults of the first system. Therefore, the model is inherently related to the exact CAN bus layout it was trained on.

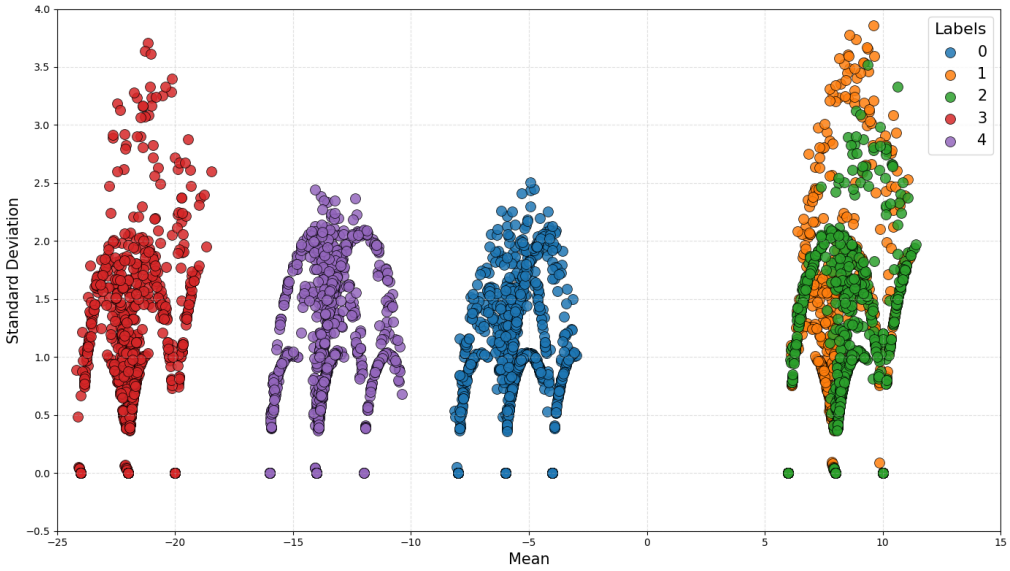

Figure 4.8: The offset mean vs. the offset standard deviation of the termination configurations described in Table 4.6. The CAN frames were transmitted from the third ECU, and the measurement was made from the left end.

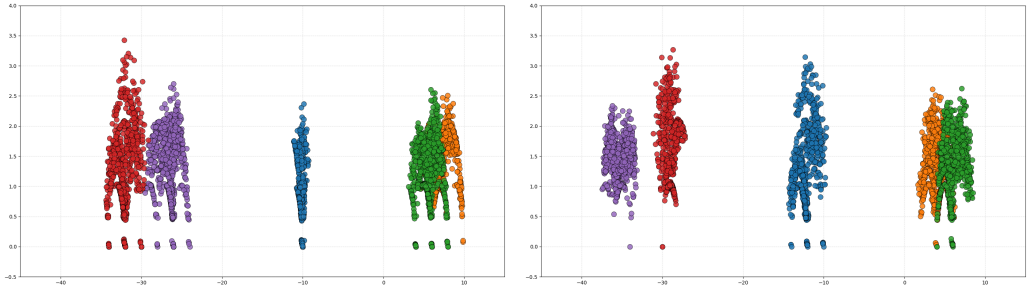

Figure 4.9: The offset mean vs. the offset standard deviation of the termination configurations described in Table 4.6. The colors refer to the same fault classes described in Fig. 4.8. The left figure showcases data transmitted from the fifth node, measured from the left end. The right figure illustrates transmissions from the first ECU, measured from the right end. The mean values of the fault classes have switched positions based on transmitting ECU and measuring end.

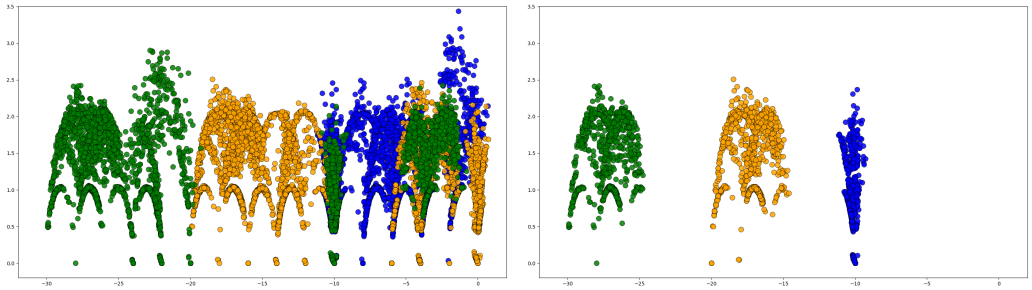

Figure 4.10: The offset mean vs. the offset standard deviation of three CAN bus layouts. The blue class refers to a CAN bus with a total of 10 m cable lengths and 2 m drop lines. The yellow class has increased the cable length with 35 m compared to the first class, and the green class has doubled the stub lengths compared to the yellow system. All measurements were made with five ECU:s and 120Ω termination in both ends.

Nodes	Bus length	Stub lines	Accuracy	Increase
5	10	2	93.74	-0.07
5	45	2	98.17	-0.05
5	45	4	99.90	-0.01
3	6	2	92.08	0.07
3	41	2	99.37	-0.10
3	41	4	99.78	0.02

Table 4.23: Results for Gradient Boosting across different test bench layouts. Nodes refer to the number of installed ECU:s on the network. Bus length (m) refers to the total cable length, and Stub lines (m) refers to the stub line length per node. Accuracy (%) refers to the accuracy of the model with default parameters. Increase (% points) refers to the accuracy increase with optimized parameters.

4.8 Test Bench Layout

As the distribution of the offset mean values where wider for larger system sizes, an investigation whether the models could distinguish fault classes to a better degree on larger systems was performed. The investigated layouts included some of the potential combinations presented in Table 3.4.

The models were trained and evaluated on the dataset with additional statistical features and the offset array combined, as it achieved the highest accuracy for the best performing model on the basic system (Section 4.5). In Table 4.23, the accuracy of the best performing model is presented for each test bench layout. The best performing model was Gradient Boosting throughout all layouts. The first row refers to the basic setup used in Section 4.3. Performing hyperparameter optimization, a negligible change to the accuracy occurred.

The results of the benchmarks on the different physical layouts indicated that the feasibility of localizing the faults was increased as the total cable length of the bus was increased. Fig. 4.11 displays the offset mean vs. the offset standard deviation

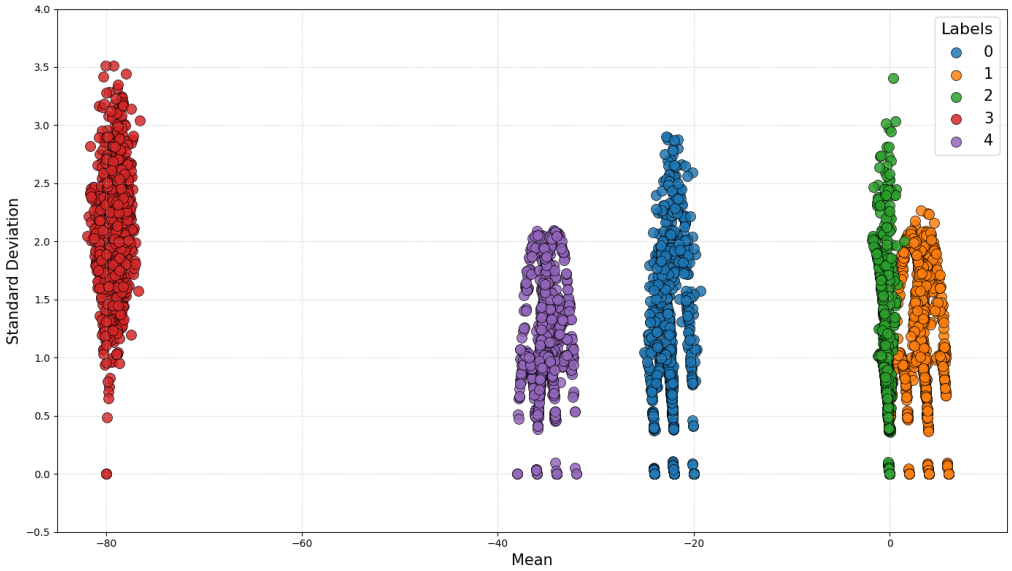

Figure 4.11: The offset mean vs. the offset standard deviation of the termination configurations described in Table 4.6. The measurement was made on a test bench with 45 meter total bus length, four meter stub lines and five nodes. CAN frames were transmitted from the third ECU, and the measurement was made from the left end.

for the fault classes described in Table 4.6 on the test bench layout of 45 m total cable length, 4 m stub lengths and 5 nodes. The visualization showcases that class 1 and 2 are almost completely separated, which explains the significant accuracy increase compared to the basic system. Additionally, the spread of all classes in the plot is greater, suggesting that the recessive edges drift further apart between different fault configurations on larger systems.

Similarly, increasing the stub length consistently increased the model's accuracy. This result makes sense considering the travel distance from the ECU to the bus line is increased with the increased stub length. However, the effect of increasing the stub length seemed to be larger per meter added compared to simply adding cable length to the bus line. This aligns well with the relation of stub lines and signal distortion described in Chapter 2.

Changing the amount of nodes on the CAN bus affects the system in several ways. Adding or removing electrical components, which contains unique electrical properties, affects the electrical properties of the system as a whole. For example, as the number of nodes directly correlates with the number of branches on the bus, the amount of reflection is affected. Furthermore, as the number of transmitting nodes correlates with the probability of several nodes transmitting simultaneously, the probability of arbitration occurring is impacted. Decreasing the amount of nodes also reduces the total cable length.

However, as seen in Table 4.23, Gradient Boosting showcased similar results evaluated on the system with three nodes compared to the CAN bus with five nodes. A

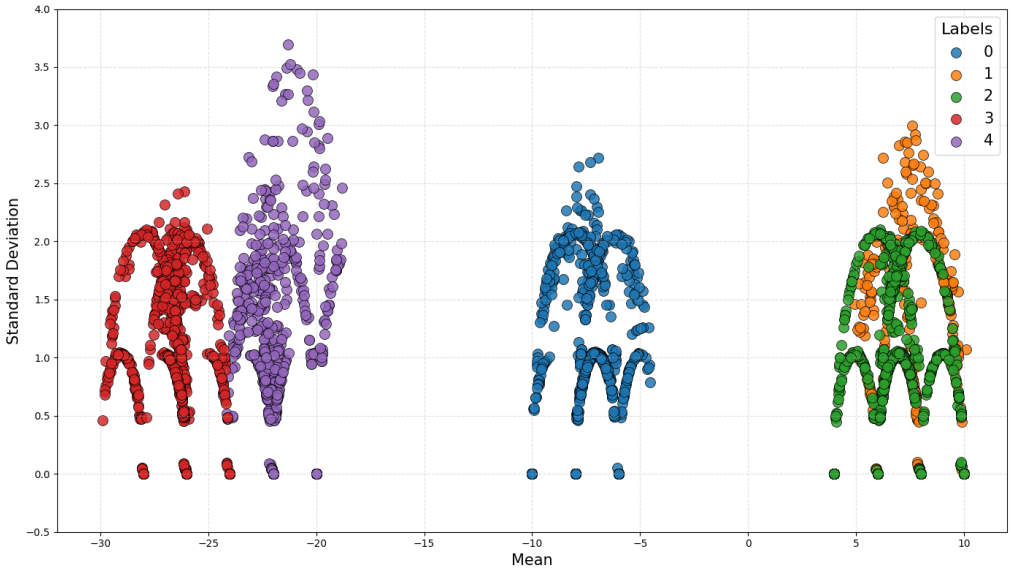

Figure 4.12: The offset mean vs. the offset standard deviation of the termination configurations described in Table 4.6. The measurement was made on a test bench with 6 meter total bus length, two meter stub lines and three nodes. CAN frames were transmitted from the third ECU, and the measurement was made from the left end.

slight decrease in accuracy was observed for the smaller layout, but as the number of nodes was decreased, so was the number of cables connecting the T-connectors, and in turn, the total cable length. Adding the 35 m cable, the opposite was observed, and Gradient Boosting achieved a higher accuracy on the three node network compared to the system with five nodes.

Fig. 4.12 illustrates the offset mean vs. the offset standard deviation for the five termination faults, measured on a test bench with three nodes, 6 m total cable length and 2 m stub line lengths. Similar to the system with five nodes (Fig. 4.8), classes 1 and 2 are overlapping to a high degree. However, there is also a slight overlap of class 3 and 4, and the values of the mean and standard deviation have changed slightly. Possibly, this change in signal behavior is negligible compared to the change in signal behavior due to the termination faults. However, as measurements only were made with three and five nodes, it can not be concluded to which degree this classification problem is affected by the number of nodes on the network.

4.8.1 Analysis

After performing measurements on different system sizes, it was concluded that a system with a shorter total cable length has a smaller difference in offsets between different classes. Increasing the size of the test bench resulted in larger separation of offset mean values and higher accuracy on the classification evaluation. This result was also indicated for longer stub lines, but such a layout was not tested isolated from a longer total cable length.

The number of nodes on the network affected the separation of the offset means of fault classes, but to a lower degree compared to the total cable length. Furthermore, decreasing the number of nodes on the network also decreases the total cable length of the network, as the cable length between the nodes was 2 m. Adding a 35 m cable between two nodes on the three node network increased the accuracy of the best performing model similarly to the five node network.

The overall results of the test bench layout analysis indicates that the total cable length was the most important factor affecting the classification, and it can not be concluded if, or to which degree, the number of ECU:s on the network affects the distribution of offset mean values, and in turn, the classification of physical faults.

5

Conclusion

In this chapter, a discussion of the results is presented. The Limitations section suggests the shortcomings of these results and the ML models' generalization capabilities. Furthermore, the section Future Works is presented, suggesting possible extensions of, and alternative approaches to, the study. Last, the study's main contributions are summarized in a concluding remark.

5.1 Discussion

The aim of the study was to investigate if machine learning could be used to reliably recognize physical faults on a CAN bus using only the timestamps of signal transitions. Several models achieved a prediction accuracy close to 100% for all fault distinctions investigated. Localizing the faults was best achieved with an accuracy of 93.74% with the Gradient Boosting model, and localizing a missing termination was the primary reason for the non-optimal results. Considering that the results were from models predicting one CAN frame at a time, meaning that outliers have a large impact, it would be unreasonable to hope for perfect accuracy separating the five classes. This is especially impactful when dealing with classes of similar shape, which missing termination on either side of the bus proved to be. However, as localizing the faults was an extension of the primary purpose of the study, it is considered to be feasible to use machine learning for CAN fault detection. Furthermore, it could be considered to be more valuable to detect that a CAN bus has a certain fault with close to certainty, than to determine that it has a fault at a certain point with a lower degree of certainty.

Investigating the CAN bus layout, the offset mean values of the fault configurations were drawn apart as the system was enlarged. Specifically, on a system with 45 m total cable length and 4 m stub lines, Gradient Boosting achieved an accuracy of 99.90% on localizing the faults. Compared to the initial system, which was of 10 m total bus length and 2 m stub line lengths, it was concluded that distinguishing faults was easier on larger systems.

The difference in performance between the evaluated models hint at characteristics of the underlying feature space. The tree-based models, Decision Tree, Random Forest and Gradient Boosting, achieved the highest accuracy scores overall, while Logistic Regression and Linear SVM performed significantly worse. This contrast suggest that although certain two dimensional projections, such as the mean ver-

sus the standard deviation of the edge offsets, appeared to show class separation except for the missing termination faults, the full feature space is unlikely to be linearly separable. The performance of K-Nearest Neighbors indicated that while local clusters exists in the feature space, class overlap and feature correlation limited its effectiveness. Multi-Layer Perceptron, despite its ability to model non-linear relationships, did not match the performance of the tree-based models, likely due to the limited dimensionality and complexity of the data. The performance of MLP could probably be increased with further hyperparameter tuning and training, but is unlikely to achieve a perfect accuracy due to the similarity of the two missing termination fault classes.

The methodology presented in this thesis transformed physical layer signal characteristics into a standardized set of offsets referencing the expected signal transition. While the offsets described the signal in nominal values, they were relative within the CAN frame. The mean of these offsets captured systematic shifts, while the standard deviation captured noise and stochastic variance. The first describing a global effect, meaning the position of the clusters, while the second describes the structure within these clusters. This explains the strong performance of simple machine learning models, as the resulting feature space is both low-dimensional and structured. Furthermore, the measure enabled to differentiate deviations of signal behavior related to to physical faults, the topology as well as specific ECU:s.

Supervised learning was used instead of unsupervised learning as it allowed for interpretable results that could be evaluated on the ground truth. It was especially helpful in inferring the difficulties in localizing a missing termination. Furthermore, priority was given to identifying certain faults, which was achieved to a high degree. An unsupervised model trained to detect anomalies would not be able to tell which kind of physical fault an anomaly relates to. On the other hand, if the goal was to detect anomalies rather than identifying certain faults, unsupervised learning would have been the better choice. Unsupervised models do not require labeled training data and generalize better to anomalies not present in the training set, which supervised models suffers to do. Therefore, approaching the problem with unsupervised models could improve generalization.

However, based on the data analysis laid out in Section 4.7, a model's inference was correlated to the system it was trained on. Therefore, an unsupervised model would suffer from the domain shift problem as well. This is, however, based on the results of the 1D-CNN:s inability to extract structure able to identify faults solely based on the inherent relation of recessive edge offsets within a CAN frame, and not the actual offset value, which proved to be related to the physical layout of the network. As the 1D-CNN seemed to be able to detect sequential patterns, the inability of accurately inferring class solely based on the sequential structure could also be due to the low dimensionality of the CAN frame offsets. If so, an unsupervised model such as an Autoencoder could be trained on e.g. the voltage data of the CAN bus instead, which would address the domain shift problem and further motivate the use of an unsupervised model.

Faulty termination can be difficult to detect using traditional diagnostic methods.

As illustrated in Fig. 2.3, the signal may appear disturbed but not entirely disrupted, making visual inspection with an oscilloscope both time consuming and inconclusive. The deviations introduced by such faults are often subtle, typically on the order of tens of nanoseconds, yet these deviations are significant enough to affect reliable CAN communication. Successfully integrating the methodology described in this thesis into existing workflows could therefore greatly increase diagnosis accuracy while reducing the time and effort needed to make such a prediction. If physical faults could be detected before functionality degrades, maintenance is improved and potential incidents are decreased. Additionally, this approach operates on observed traffic without requiring modifications to either the CAN protocol or the system hardware.

5.2 Limitations

The main limitations of this study regards the generalizability of the results in two regards. First, the data was generated under stable indoors conditions, which affects the distributions of the classes and minimizes the amount of outliers due to physical noise. Second, as the cable lengths, stub line lengths and number of ECU:s affects the electrical properties of the system, the recessive transition offset values were seen to shift with the system layout. Therefore, a model trained on a network of a certain layout is poorly generalized to a system of a different layout.

CAN traffic was generated under very stable conditions in an indoors climate, including factors such as temperature, noise, and the absence of other disturbances. In contrast, all of these conditions are varying in a real, outdoors setting. For example, in a car, components such as the engine create noise and EMI, and can potentially result in drastic temperature changes affecting the signal behavior of the CAN bus negatively. Furthermore, for a vehicle running on a road, there are potentially many external factors affecting the CAN bus communication, such as vibration from the wheels, and humidity, dust and salt affecting the quality of the electrical components over time. Further investigation of real-world data is required to make reliable conclusions.

The procedure described in the thesis comes with a weakness to domain shift. Different layouts of the CAN bus, such as cable lengths, stub lengths and number of nodes, lead to different timings of recessive edges for all investigated faults, indicating a high correlation of signal behavior and physical layout. The models required something to relate the offset values to, and would therefore need to be trained on a nominal system in order to predict faults on said system. Due to changes in the physical layout, a given system can look like a completely different one from the perspective of signal transition timestamps, which impacts the feasibility of a machine learning model identifying faults negatively. This implies that a model's generalizability from one layout to another is limited to a large extent, i.e., that a model's capabilities of making accurate predictions are tightly connected to the physical layout of the CAN bus it was trained on.

During preprocessing, the frame ID was output and used as a feature in model

training. As the ID:s are unique to ECU:s in the CAN standard, using the ID as a feature enabled the models to learn signal behavior unique to the transmitting node. However, ID:s do differ from one CAN configuration to another. A model trained on ID:s in one network is therefore limited to evaluate a network with identical ID configurations.

As seen in Section 4.7, the offset values depended on the measuring point. Therefore, to be able to distinguish frames, and to not limit the results to one measuring point, the measuring point ("node" in Section 3.4) was used as a feature to enable the model to distinguish signals depending on the measuring point. While this enabled fault analysis from both ends of the network, it created a dependency of the definitions of the ends of the network. Trouble shooting a network would require one to know which measuring point is used, and how it was defined during training.

These limitations highlight the practical difficulties this methodology would have if directly applied to a CAN bus as a trouble shooting instrument. It would be required to retrain the model for each unique system to evaluate, and it would require one to be able to simulate the faults to detect.

5.3 Future Work

After finishing this study, several potential avenues for continued research became clear. Some of them have already been introduced in this chapter, such as generalization and unsupervised learning approaches. Others include extensions such as exploring more faults, more types of systems and experiment with real data from vehicles.

If the study was extended by a few months, a natural next step would be to continue working with the existing data and a similar preprocessing pipeline, but deepen the analysis. This could involve refining the feature engineering process, testing alternative representations of the CAN data, or attempt to generalize the results. Short-term extensions could also include evaluating the robustness of the models under varying message lengths. Although that might reduce reproducibility, it more closely reflect real-world conditions, where message length is not perfectly uniform.

A longer extension would allow for in-depth exploration of other representations of CAN traffic such as voltage data. This would enable a thorough evaluation of other machine learning architectures potentially better at this task. Furthermore, it would motivate the use of unsupervised learning, potentially better capable of generalization between networks. Another key longterm goal would be to validate the approach on real vehicle data. Controlled laboratory datasets provide a clean foundation, but real CAN traffic is influenced by many external factors. An investigation into how the CAN bus data is affected by external factors and manufacturing differences would add value to the results of this study.

5.3.1 Generalization

From the observations in this study, the issue with training a model on one system and expecting it to perform well on another system was that the offset distribution shifted. For example, one CAN bus could exhibit recessive edges around 20 clock cycles earlier than expected for normal termination, and around 5 clock cycles earlier for a missing termination in one end. Meanwhile, another system could exhibit a recessive edge offset around 5 clock cycles for normal termination. Therefore, training a model on the nominal offset values will suffer from this domain shift problem.

To address this, a pattern within the data not tied to a specific CAN bus needs to be found. Alternatively, one could try to describe the system through the physical parameters such as total impedance, total cable length or number of ECU:s, and try to find a learnable relationship between the CAN system itself and the signal behavior. However, the challenge of such an approach is to find a faithful representation of a system that would align well with real CAN buses. Furthermore, as stated in Section 2.3, manufacturing of electrical components results in small deviations of signal behavior for identical components. This assumption could lead to poor generalization from one network to another, even if the physical layout was identical.

5.3.2 Additional Faults

Another natural continuation of this study is to research more types of faults and see if this methodology could be used for those as well. A short investigation into wire disconnection and short circuit faults was made on the test bench with five nodes, 10 m total bus length, and 2 m stub line lengths.

For wire disconnection, fault classes were defined by disconnecting the CAN_H out connector on nodes two to five, and normal termination in both ends. The out connector of the leftmost node was not detached, as this only affects the measuring device on the left end. Disconnecting the CAN_H wire divided the network into two systems, and no frames transmitted from the other side of the wire that had been cut off were received by the logging device. For example, cutting the out node of the second T-connector and sampling from the left end, frames transmitted from the first ECU were received. Measuring from the right end under similar conditions received frames from the second, third, fourth and fifth node.

The nodes were defined to transmit 1000 frames each, which resulted in a total of 5000 frames per fault, halving the total amount of frames per fault compared to the termination faults. Therefore, the normal termination dataset was undersampled (motivated in Section 4.1). Gradient Boosting scored the greatest accuracy on the test set with 99.88%, indicating an excellent ability to identify and localize wire disconnection. However, disconnecting a wire has the obvious effect of cutting off messages from the other side of the cut. Essentially, with knowledge of the origin of ID:s, this analysis could be performed by inspecting which ID:s the measuring unit receives.

Two short circuit faults were investigated, the first by connecting CAN_H to 12 V, and the second by connecting CAN_L to ground. In addition to the base case with 120Ω termination in both ends, this amounted to three fault classes. Gradient Boosting performed best with 99.92% accuracy with default parameters.

Cutting a wire and splitting the bus in two possibly affects the bus similarly to the missing termination fault. To investigate whether a model could distinguish between the three faults, a concatenation of all previous datasets was made. This amounted a total 11 classes – five termination cases, four wire disconnections faults and two short circuit faults. Gradient Boosting performed the best and achieved an accuracy of 96.75% with default parameters, indicating that the model was able to distinguish the physical faults correctly to a high degree. The confusion matrix of Gradient Boosting on the combined faults dataset is presented in Table 5.1. Evaluating the predictions of Gradient Boosting, it was only the missing termination in one end fault it did not classify with close to perfect accuracy.

		Predicted class										
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Actual class	**0**	998	0	0	2	0	0	0	0	0	0	0
	1	0	847	153	0	0	0	0	0	0	0	0
	2	0	192	807	0	0	0	0	0	0	0	1
	3	0	0	0	999	1	0	0	0	0	0	0
	4	0	0	0	1	999	0	0	0	0	0	0
	5	0	0	0	0	0	998	1	0	1	0	0
	6	0	0	0	0	0	0	1000	0	0	0	0
	7	0	0	1	0	0	1	0	997	0	0	1
	8	0	0	0	0	0	0	0	0	999	0	1
	9	0	0	0	0	0	0	0	1	0	998	1
	10	0	0	0	0	0	0	0	0	0	0	1000

Table 5.1: Confusion Matrix of Gradient Boosting on the termination, wire disconnection and short circuit faults combined. Class 0-4 refers to the termination faults, 5-8 refers to the wire disconnection faults, and 9-10 refers to the short circuit faults.

If the study was performed under a longer period of time, an in-depth analysis of the two additional physical faults would be performed. It is not clear to which extent the results in Table 5.1 can be generalized. For example, the procedure of evaluating the results on additional CAN bus layouts described in Section 4.8 should be performed.

Furthermore, if additional physical faults were added, the feasibility of one model distinguishing each fault becomes questionable. One approach of solving this issue

could be to develop one separate model per fault to investigate, similar to the procedure laid out in this thesis of predicting termination faults. However, such an approach would suffer from the practical use of trouble-shooting a CAN bus, as the fault to investigate is unknown.

5.3.3 Voltage Data

Since issues regarding the data arose during the study in a multitude of ways, an interesting continuation would be to explore more complex representations of the CAN traffic. Potentially, high-dimensional data could solve the issues regarding generalization and deep learning models that was illustrated by the experiments in this study. Several related studies used the voltage data on the CAN bus to train ML models, indicating a valuable sequential pattern.

Using the analog signal as input feature would allow for more complex architectures to learn significant patterns arising during faulty conditions. Since the 1D-CNN performed better on the original order of the offset array, a sequential pattern could be considered to exist in the data. Using high dimensional voltage data could potentially increase the predicting abilities of a sequential model. Furthermore, sequential patterns could prove to be system invariant, meaning that it persists between different CAN systems. If such a procedure would be successful, it would combat the domain shift problem as well.

5.3.4 CAN Bus Analysis Tool

Another relevant continuation of this study would be to investigate how the results could be adapted to continuous traffic on a real CAN bus. In classification, there is a tradeoff between false positives and false negatives, also known as the precision and recall tradeoff. In short, it describes that attempting to increase one usually decreases the other, and it can be used to analyze whether a model has a bias of predicting positive or negative. Compared to e.g. medical classification, where false negatives are highly undesirable, one could argue that the opposite holds for a CAN bus analysis tool. A high rate of false positives would alarm that the system has an issue when it in fact does not. Conversely, for a higher rate of false negatives, the analysis tool would stay silent unless certain that a fault had occurred. Measuring an active CAN bus, loads of CAN frames are sampled. Therefore, a model's prediction capabilities would not necessarily be dependent on one CAN frame. It is essentially only required that the model makes one conclusive prediction that an error has occurred. Such an approach, possibly done by introducing a bias in the model, was not examined in the study, but could be of higher value in a practical setting.

The precision and recall tradeoff could also be addressed by developing a model classifying the distribution of a set of frames rather than one individual frame, or by inferring the class of a set of frames by averaging over predictions. Applying the latter idea to this study, a model would be trained according to the procedure described in this thesis and evaluated on one physical configuration without known labels, representing the real CAN bus to investigate. Assume the model is Gradient

Boosting, trained on the Iteration 3 dataset, and that the predictions and true classes are given as in Table 4.8.

For a normally terminated bus, the model made $\frac{2000}{2000}$ predictions of class 0, a strong indication that the bus does not contain a fault. Localizing double termination, the model made $\frac{2000}{2000}$ predictions of class 3, and $\frac{1997}{2000}$ of class 4, both of which are strong suggestions that the bus contains a certain fault at a certain end. For one of the difficult cases, e.g. classifying a missing termination in the left end, Gradient Boosting made 1557 predictions of label 1, and 443 of label 2. A prediction frequency of $\frac{1557}{2000} = 77.85\%$ of class 1 and $\frac{443}{2000} = 22.15\%$ of class 2, the model indicates that there is a missing termination in the left end, but that the data is similar to the case where the missing termination is in the right end. A strong indication of one fault, and a weaker indication of another, tells that the model mostly indicates that the bus contains the first fault, but that it is similar to the second. This insight is a valuable first step during trouble shooting. Furthermore, and perhaps most importantly, the model made no predictions of class 0, 3, and 4, a sturdy suggestion that the bus is not well terminated or double terminated in one end.

5.4 Conclusion

It is considered to be feasible for an ML model to reliably predict termination faults on a CAN bus based on the offset of the recessive edge from the expected value. The electrical signals deviate enough from fault to fault to make this distinction. Integration of the methodology presented in this thesis into existing workflows could therefore improve diagnostic speed and reliability. To our knowledge, such an assessment, based on recessive edge offsets, has not been made previously, which makes this study a contribution to the field of CAN bus fault diagnosis.

Tree-based architectures, in particular Gradient Boosting, are considered to be the best approach of inferring fault classes based on these offset features. Particularly, they are considered to outperform deep learning architectures due to the low dimensionality of the feature space of signal transitions within one CAN frame. The results indicates that deep learning approaches would require higher dimensions of the input data, such as the analog signal of the CAN bus, to enable better classification performance than decision trees.

However, due to the many factors affecting the electrical properties of a CAN bus, the ML based analysis laid out in this thesis is closely connected to the particular network the models were trained on. Therefore, even for a model accurately predicting termination faults on one CAN bus, it is likely not possible to apply the model to another CAN bus. This conclusion could also apply to the same network at another point in time, as the electrical behavior of the signals on a CAN bus can change over time due to external factors and aging of components. Furthermore, the study was performed with high quality hardware in an indoors environment. Whether the results of the study is applicable to a real-world setting needs to be further investigated.

Bibliography

[1] H. M. Boland, M. I. Burgett, A. J. Etienne, and R. M. S. III, "An Overview of CAN-BUS Development, Utilization, and Future Potential in Serial Network Messaging for Off-Road Mobile Equipment," in *Technology in Agriculture*, F. Ahmad and M. Sultan, Eds., London: IntechOpen, 2021, ch. 25. DOI: 10.5772/intechopen.98444.

[2] C. Ebert and C. Jones, "Embedded Software: Facts, Figures, and Future," *Computer*, vol. 42, no. 4, pp. 42–52, 2009. DOI: 10.1109/MC.2009.118.

[3] Z. Guo, K. Koufos, M. Dianati, and R. Woodman, "State-of-the-art virtualisation technologies for the centralised automotive E/E architecture," *Frontiers in Future Transportation*, vol. 6, 2025. DOI: 10.3389/ffutr.2025.1519390.

[4] Robert Bosch GmbH, *Bosch Automotive Electrics and Automotive Electronics: Systems and Components, Networking and Hybrid Drive*. Springer Vieweg Wiesbaden, Jan. 2014. DOI: 10.1007/978-3-658-01784-2.

[5] Y. Chen, Q. Li, and Q. Luo, "Research on Fault Diagnosis of Vehicle-mounted Network Communication Based on CAN Bus," *IOP Conference Series: Materials Science and Engineering*, vol. 677, no. 4, Dec. 2019. DOI: 10.1088/1757-899X/677/4/042049.

[6] N. A. Lwahas, S. L. Lawal, L. S. Aduku, S. A. Oyèkólá, and A. Surajo, "Recent Advances in Machine Learning for Fault Diagnosis in Mechatronic Systems," *Global Journal of Research in Engineering and Computer Sciences*, vol. 5, no. 4, pp. 1–6, Jul. 2025. DOI: 10.5281/zenodo.15852048.

[7] W. Lawrenz, Ed., *CAN System Engineering: From Theory to practical Applications*, 2nd ed. London, England: Springer, 2013.

[8] P.-S. Murvay and B. Groza, "Source identification using signal characteristics in controller area networks," *IEEE Signal Processing Letters*, vol. 21, no. 4, pp. 395–399, Apr. 2014. DOI: 10.1109/LSP.2014.2304139.

[9] W. Choi, H. J. Jo, S. Woo, J. Y. Chun, J. Park, and D. H. Lee, "Identifying ECUs Using Inimitable Characteristics of Signals in Controller Area Networks," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 6, pp. 4757–4770, Jun. 2018. DOI: 10.1109/TVT.2018.2810232.

[10] Vector CANtech, Inc., "Common High Speed Physical Layer Problems," 2003. [Online]. Available: https://cdn.vector.com/cms/content/know-how/_application-notes/AN-ANI-1-115_HS_Physical_Layer_Problems.pdf.

[11] M. Hell, "CAN XL Physical Layer Network Design," in *Proceedings of the 18th International CAN Conference*, Infineon Technologies, Baden-Baden, Germany, May 2024.

[12] Texas Instruments. "RS-485 Basics: When Termination Is Necessary, and How to Do It Properly," Accessed: May 4, 2026. [Online]. Available: https://www.ti.com/document-viewer/lit/html/SSZTB23.

[13] A. Golda and A. Kos, "Temperature influence on power consumption and time delay," in *Euromicro Symposium on Digital System Design, 2003. Proceedings.*, 2003, pp. 378–382. DOI: 10.1109/DSD.2003.1231970.

[14] W. Lutes G. Diener, "Thermal coefficient of delay for various coaxial and fiber-optic cables," *The Telecommunications and Data Acquisition Report*, 1989.

[15] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer, 2006.

[16] I. H. Sarker, "Machine Learning: Algorithms, Real-World Applications and Research Directions," *SN Computer Science*, vol. 2, no. 3, 2021. DOI: 10.1007/s42979-021-00592-x.

[17] L. Deng and D. Yu, "Deep Learning: Methods and Applications," *Foundations and Trends in Signal Processing*, vol. 7, pp. 197–387, Jun. 2014. DOI: 10.1561/2000000039.

[18] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques* (The Morgan Kaufmann Series in Data Management Systems), 3rd ed. Waltham, MA, USA: Morgan Kaufmann, 2011.

[19] D. Micci-Barreca, "A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems," *SIGKDD Explorations Newsletter*, vol. 3, no. 1, pp. 27–32, Jul. 2001. DOI: 10.1145/507533.507538.

[20] O. Rainio, J. Teuho, and R. Klén, "Evaluation metrics and statistical tests for machine learning," *Scientific Reports*, vol. 14, Mar. 2024. DOI: 10.1038/s41598-024-56706-x.

[21] P. Domingos, "A few useful things to know about machine learning," *Communications of the ACM*, vol. 55, pp. 78–87, Oct. 2012. DOI: 10.1145/2347736.2347755.

[22] Z. Lipton and J. Steinhardt, "Troubling Trends in Machine Learning Scholarship: Some ML papers suffer from flaws that could mislead the public and stymie future research.," *Queue*, vol. 17, pp. 45–77, Feb. 2019. DOI: 10.1145/3317287.3328534.

[23] D. Wolpert and W. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997. DOI: 10.1109/4235.585893.

[24] C. Molnar, *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*, 3rd ed. 2025. [Online]. Available: https://christophm.github.io/interpretable-ml-book.

[25] L. Rokach and O. Maimon, "Decision trees," in *Data Mining and Knowledge Discovery Handbook*, O. Maimon and L. Rokach, Eds., Boston, MA, USA: Springer, Jan. 2005, pp. 165–192. DOI: 10.1007/0-387-25465-X_9.

[26] T. K. Ho, "Random decision forests," in *Proceedings of 3rd international conference on document analysis and recognition*, IEEE, vol. 1, Montreal, QC, Canada, 1995, pp. 278–282. DOI: 10.1109/ICDAR.1995.598994.

[27] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, Oct. 2001. DOI: 10.1214/aos/1013203451.

[28] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995. DOI: 10.1007/BF00994018.

[29] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, Jan. 1967. DOI: 10.1109/TIT.1967.1053964.

[30] M.-C. Popescu, V. E. Balas, L. Perescu-Popescu, and N. Mastorakis, "Multilayer perceptron and neural networks," *WSEAS Transactions on Circuits and Systems*, vol. 8, no. 7, pp. 579–588, Jul. 2009.

[31] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: Analysis, applications, and prospects," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 12, pp. 6999–7019, Dec. 2021. DOI: 10.1109/TNNLS.2021.3084827.

[32] P. Probst, A.-L. Boulesteix, and B. Bischl, "Tunability: Importance of hyperparameters of machine learning algorithms," 2019. DOI: 10.48550/arXiv.1802.09596.

[33] M. Hanselmann, T. Strauss, K. Dormann, and H. Ulmer, "CANet: An Unsupervised Intrusion Detection System for High Dimensional CAN Bus Data," *IEEE Access*, vol. 8, pp. 58 194–58 205, 2020. DOI: 10.1109/ACCESS.2020.2982544.

[34] A. Battaglia, N. Canino, P. Dini, G. Lombardo, F. Longo, and D. Rossi, "Autoencoder-Based Detection of Physical-Layer Anomalies in Automotive CAN Networks," in *2025 IEEE 31st International Symposium on On-Line Testing and Robust System Design (IOLTS)*, Ischia, Italy, 2025, pp. 1–4. DOI: 10.1109/IOLTS65288.2025.11116870.

[35] E. Levy, A. Shabtai, B. Groza, P.-S. Murvay, and Y. Elovici, "CAN-LOC: Spoofing Detection and Physical Intrusion Localization on an In-Vehicle CAN Bus Based on Deep Features of Voltage Signals," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 4800–4814, 2023. DOI: 10.1109/TIFS.2023.3297444.

[36] N. Liu, C. Moreno, M. Dunne, and S. Fischmeister, "vProfile: Voltage-Based Anomaly Detection in Controller Area Networks," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, 2021, pp. 1142–1147. DOI: 10.23919/DATE51398.2021.9474106.

[37] H. Zhao, S. Sun, and B. Jin, "Sequential Fault Diagnosis Based on LSTM Neural Network," *IEEE Access*, vol. 6, pp. 12 929–12 939, 2018. DOI: 10.1109/ACCESS.2018.2794765.

[38] R. DiPietro and G. D. Hager, "Deep learning: RNNs and LSTM," in *Handbook of Medical Image Computing and Computer Assisted Intervention*, Academic Press, 2020, pp. 503–519. DOI: 10.1016/B978-0-12-816176-0.00026-0.

[39] P. Ngo and J. Sprinkle, "Lightweight LSTM for CAN Signal Decoding," in *Proceedings of the Workshop on Data-Driven and Intelligent Cyber-Physical Systems*, Nashville, TN, USA: Association for Computing Machinery, 2021, pp. 27–31. DOI: 10.1145/3459609.3460528.

[40] Chandralekha, H. M. Chandrashekar, P. S. Nijesh, P. P. S. Sreejith, and M. K. Ghosh, "Anomaly detection in recorded CAN log using DBSCAN and LSTM Autoencoder," in *2022 IEEE 3rd Global Conference for Advancement in Technology (GCAT)*, Bangalore, India, 2022, pp. 1–7. DOI: 10.1109/GCAT55367.2022.9971885.

[41] 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. DOI: 10.1109/ACCESS.2020.3029307.

[42] J. Zhang et al., "A Deep Learning Framework for Driving Behavior Identification on In-Vehicle CAN-BUS Sensor Data," *Sensors*, vol. 19, no. 6, 2019. DOI: 10.3390/s19061356.

[43] X. Lin et al., "ByCAN: Reverse Engineering Controller Area Network (CAN) Messages From Bit to Byte Level," *IEEE Internet of Things Journal*, vol. 11, no. 21, pp. 35 477–35 491, Nov. 2024. DOI: 10.1109/JIOT.2024.3435833.

[44] A. Buscemi, I. Turcanu, G. Castignani, R. Crunelle, and T. Engel, "CAN-Match: A Fully Automated Tool for CAN Bus Reverse Engineering Based on Frame Matching," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 12, pp. 12 358–12 373, 2021. DOI: 10.1109/TVT.2021.3124550.

[45] X. Peng, J. Ge, D. He, and Z. Liu, "Deep Learning-Based Fault Diagnosis for CAN Bus," in *2024 6th International Conference on Electronic Engineering and Informatics (EEI)*, Chongqing, China, 2024, pp. 1–6. DOI: 10.1109/EEI63073.2024.10696010.

[46] Kvaser. "CAN bus API (CANlib)," Accessed: Jan. 27, 2026. [Online]. Available: https://kvaser.com/canlib-webhelp/page_canlib.htm.

[47] Kvaser. "Kvaser CANtegrity," Accessed: Jan. 27, 2026. [Online]. Available: https://kvaser.com/kvaser-cantegrity/.

[48] R. Ansari and L. Valbonesi, "Signals and systems," in *The Electrical Engineering Handbook*, W.-K. Chen, Ed., Burlington: Academic Press, 2005, pp. 813–837. DOI: 10.1016/B978-012170960-0/50061-X.

[49] Python Software Foundation. "random – Generate pseudo-random numbers," Accessed: Jan. 27, 2026. [Online]. Available: https://docs.python.org/3/library/random.html.

[50] Python Software Foundation. "threading – Thread-based parallelism," Accessed: Jan. 30, 2026. [Online]. Available: https://docs.python.org/3/library/threading.html.

[51] F. Pedregosa et al., "Scikit-learn: Machine learning in python," *The Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, Nov. 2011, ISSN: 1532-4435.

[52] The Scikit-learn community. "StandardScaler," Accessed: Apr. 1, 2026. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.

[53] The Scikit-learn community. "Dummy Classifier," Accessed: Mar. 31, 2026. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html.

[54] The Scikit-learn community. “GridSearchCV,” Accessed: Mar. 3, 2026. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.