Universiteit Leiden

Building Trust in Automotive Intrusion Detection Systems Through XAI Methods

Jingya Fu (s3446778)

Supervisors:
Nele Mentens & Wouter Hellemans

Bachelor Thesis– Data Science and Artificial Intelligence

Leiden Institute of Advanced Computer Science (LIACS)
liacs.leidenuniv.nl

August 20, 2026

Abstract

Deep-learning-based Intrusion Detection Systems (IDS) have achieved strong performance in detecting attacks on in-vehicle networks. However, their predictions are often difficult to interpret, which limits trust in these systems and their further application. This thesis investigates whether explainable artificial intelligence methods can reveal the reasons behind the predictions of an automotive intrusion detection system. A dilated one-dimensional convolutional neural network is trained to classify windows of CAN traffic from the CarDS dataset. The trained model is analysed using SHAP for local feature-level explanations and TRUSTEE for a global decision-tree approximation of its behaviour. The results show that the model mainly relies on timing patterns. These patterns are effective for detecting attacks that alter traffic timing or message frequency, but are less effective for stealthier attacks that preserve normal timing behaviour. The combined SHAP–TRUSTEE analysis further reveals an interface-specific shortcut for UDS attacks, suggesting that the model also exploits dataset-specific correlations. The explanations can guide the design of feature representations that improve model performance, while also helping to identify potential inductive biases in the model's learned decision strategies.

Contents

1 Introduction

1.1 Motivation

Modern vehicles contain numerous Electronic Control Units (ECUs) that exchange information over in-vehicle networks. Among the existing networking protocols, the Controller Area Network (CAN) has long served as the de-facto standard for in-vehicle communication [11]. However, the CAN protocol lacks authentication and encryption by design, leaving vehicles exposed to attacks such as message injection, replay, and masquerading [32]. Such attacks can disable ECUs or main functions, putting the safety of drivers and passengers at risk.

To counter this, automotive intrusion detection systems (IDSs) have been proposed. Early signature-based IDSs match data to known attack signatures in databases, and are unable to detect zero-day attacks [2]. By contrast, anomaly-based IDSs built on unsupervised learning models are better at detecting previously unseen attacks. Deep-learning models in particular, such as convolutional neural networks (CNNs) and temporal convolutional networks (TCNs), have shown strong performance on in-vehicle traffic [16].

However, due to the complexity of deep-learning models, it is often unclear why they produce certain predictions. In automotive security, decisions that cannot be explained reduce users' trust in the system and limit its real-world deployment. Moreover, the reported excellent performance of models may rely on shortcut learning or dataset artifacts instead of real attack patterns [12].

Explainable artificial intelligence (XAI) methods provide tools to open this black box. Local methods such as SHAP quantify how much each input feature contributes to an individual prediction, while global methods such as TRUSTEE approximate the overall behaviour of a model with an interpretable decision tree. Such explanations can reveal which features a model actually relies on when making certain predictions, which not only build users' trust in model predictions, but can in turn guide the design of better feature representations.

1.2 Research Question

Our research questions can be summarised as:

1. How does the trained model perform on the dataset, specifically across the different attack categories?

2. Which features do the local SHAP explanations identify as driving the model's predictions?

3. Are the global explanations of TRUSTEE consistent with the local SHAP analysis?

To answer these questions, a dilated one-dimensional CNN is trained to classify the CAN traffic from the CarDS dataset [8]. The trained model is then analysed with SHAP for individual sample explanations and with TRUSTEE for a global decision-tree approximation. The results show that the model mainly relies on timing patterns. It detects attacks that alter the timing or frequency of messages, but performs a lot worse on stealthier attacks that preserve normal timing behaviour. The XAI analysis further identifies an interface-specific decision pattern for UDS attacks, which is consistently reflected by both SHAP and TRUSTEE.

1.3 Thesis Overview

The thesis is structured as follows: Section 2 provides the background knowledge on automotive network security, intrusion detection, and XAI; Section 3 reviews related work in this field; Section 4 describes the dataset and its preprocessing; Section 5 presents the experimental setup, building the context for the results; Section 6 presents and interprets the results; Section 7 summarizes the findings and suggests directions for future work.

2 Background

In this thesis, since we delve into explaining the classification results of CNN-based automotive IDS, it is essential to briefly explore the data field we aim to explain, the baseline model we make use of, and the explanation methods used.

2.1 In-Vehicle Network Security

2.1.1 In-Vehicle Networks and CAN

Modern vehicles contain numerous electronic control units (ECUs), such as engine control units, braking control units, airbag control units, and body control modules. ECUs, as embedded devices designed to monitor and control the state of vehicles [31], have the need to exchange information to coordinate vehicle functions, and are thus interconnected by multiple in-vehicle networking protocols, such as Controller Area Network (CAN), Local Interconnect Network (LIN), FlexRay, Media Oriented System Transport (MOST), and Ethernet [11]. Among these protocols, the CAN has for a long time served as the de-facto standard for in-vehicle communication.

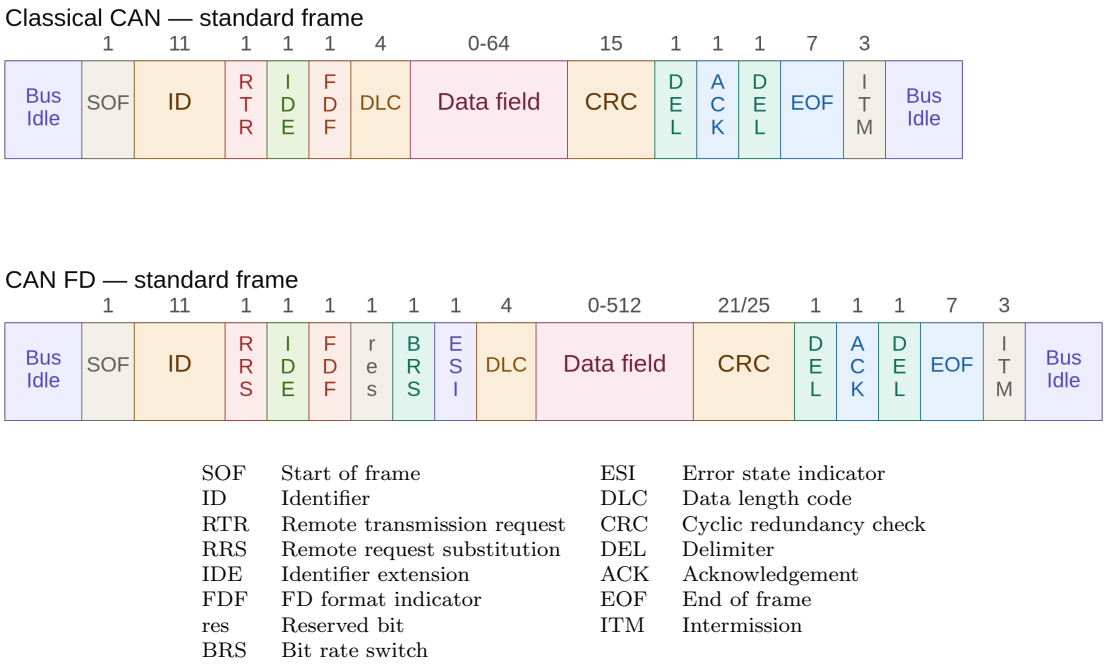
Each CAN message contains an identifier and a data field. CAN identifiers can be of base format (11-bit long) or extended format (29-bit long). Currently there are two widely used forms of CAN data frames: Classical CAN (CAN CC) and CAN with Flexible Data-Rate (CAN FD). CAN CC supports data payload of up to 8 bytes and uses a fixed bit rate (up to 1 Mbps) in transmission; CAN FD extends the frame format to support max payload length of 64 bytes and has a transmission speed for the header at up to 1 Mbps, but boosts speed to 2-8 Mbps for actual data transmission. Figure 1 shows the structure of a CAN CC frame and CAN FD frame with standard format identifiers.

CAN is designed to be lightweight and robust, and demonstrates many benefits in network traffic communication such as low cost and high reliability. However, some CAN features also lead to security vulnerabilities. For example, CAN is unable to prevent unauthorized devices from joining the bus; Moreover, CAN uses a broadcast communication model, where unencrypted messages transmitted on the shared bus can be received by all connected ECUs [32]. These limitations enable several attacks against in-vehicle networks, including message injection, spoofing, replay, fuzzing, and denial-of-service attacks, which are discussed in the following section.

2.1.2 Automotive Attacks

An adversary that has obtained access to an IVN may possess different levels of capability. A basic adversary may be able to observe network traffic and inject malicious messages, whereas a more powerful adversary may read, modify, delay, or even block legitimate communication before they reach the gateway ECU (GWE) [8]. These capabilities enable attacks with different objectives, ranging from traffic disruption and reconnaissance to the manipulation or suppression of legitimate communication. The following provides an overview of the CAN attacks considered in this thesis.

Denial of Service. A CAN DoS attack aims to reduce the availability of the system by repeatedly sending high-priority messages. Since CAN uses priority-based arbitration, these messages may occupy the shared bus and delay or prevent lower-priority messages from being transmitted.



SOF	Start of frame	ESI	Error state indicator
ID	Identifier	DLC	Data length code
RTR	Remote transmission request	CRC	Cyclic redundancy check
RRS	Remote request substitution	DEL	Delimiter
IDE	Identifier extension	ACK	Acknowledgement
FDF	FD format indicator	EOF	End of frame
res	Reserved bit	ITM	Intermission
BRS	Bit rate switch		

Figure 1: Structure of a CAN CC frame and a CAN FD frame

Fuzzing. In a CAN fuzzing attack, an adversary injects messages with randomized payload values or identifiers. It is often used during reconnaissance to identify which parts of a CAN message influence a particular vehicle function, but it may also cause unintended vehicle behaviour.

Replay. A replay attack involves recording an authentic CAN message and retransmitting it at a later phase. The retransmitted message, although having valid payload and identifiers, may appear at an inappropriate time in the sequence and potentially cause vehicle malfunctions.

Spoofing. In a CAN spoofing attack, the adversary constructs messages imitating valid communication. Unlike a replay attack, which retransmits previously recorded messages, spoofing attacks use legitimate identifiers but maliciously crafted payload values, in order to convey a false vehicle state or trigger a target function.

UDS. Unified Diagnostic Services (UDS) is a diagnostic communication protocol used to communicate with ECUs for functions such as diagnostics, configuration, and software updates. A UDS attack maliciously abuses this protocol through, for example, unauthorized access or sensitive data extraction. Attackers may manipulate critical systems like braking or steering through UDS attack.

Masquerading. A masquerading attack is a stealthier form of message manipulation, in which the adversary suppresses the communication of legitimate ECUs and continues communication using their identifiers. Unlike ordinary spoofing, where legitimate and malicious messages may appear simultaneously and create conflicts, masquerading prevents the receiver from observing the original messages.

2.2 Automotive Intrusion Detection Systems

Automotive IDSs can generally be divided into host-based IDSs (HIDSs), which monitor the behaviour of individual ECUs or other in-vehicle components, and network-based IDSs (NIDSs), which analyse communication traffic exchanged over the in-vehicle network. This thesis focuses on NIDSs, specifically on the analysis of CAN traffic to distinguish malicious activities from benign messages.

An automotive NIDS protects in-vehicle network communication by monitoring traffic and raising alarms when detecting malicious traffic. It generally does not actively intervene in the flow of network traffic, but reports suspicious events to network administrators or automated protection mechanisms, which are responsible for taking actions to secure the system [16].

2.2.1 Automotive IDS Approaches

In [16], Lampe and Meng proposed a categorization of IDSs for the automotive CAN bus, including non-learning, traditional machine learning and deep-learning paradigms. Non-learning IDSs include sequence-based IDSs, rules-based IDSs, specification-based IDSs, etc. These IDSs generally rely heavily on pre-defined patterns, rules or specifications, and are lightweight but less accurate compared with well-trained machine learning approaches. Traditional machine learning IDSs make use of machine learning algorithms such as clustering, K-nearest neighbours, support vector machine, decision tree, random forest, etc. to classify IVN traffic. They are usually slower but more accurate than non-learning IDSs. Deep-learning IDSs, based on the concepts of artificial neural networks (ANNs), bear different architectures including feed-forward neural networks, convolutional neural networks, recurrent neural networks, long short-term memory, and so on. The existence of one or more hidden layers in deep-learning IDSs marks the difference between them and traditional machine learning IDSs. Deep-learning IDSs are resource intensive, requiring ample computational power, time and memory [16]. They usually achieve better performance than traditional machine-learning IDSs, but are also slower when it comes to reaction time.

2.2.2 Deep-Learning Automotive IDS

For deep-learning automotive IDSs, the input information may include CAN identifiers, payload bytes, payload length, network interfaces, time intervals, etc. [27]. Individual CAN messages are often naturally considered as the inputs of the IDS model. For attacks that result from invalid identifiers or payload bytes, such inputs may be valid. However, when it comes to attacks that can only be recognized from their contexts (e.g. replay attack), inputting an individual CAN message is not enough. To help automotive IDSs to detect such attacks, multiple consecutive CAN messages are often grouped into fixed size windows to form contexts [3] [9]. For a binary model, the outputs can either be benign or malicious; while a multi-class model will have outputs being assigned to different attack categories. In supervised intrusion detection, the training data will have labels such as benign or malicious [22]. The classifier then predicts the labels of some unseen traffic. In unsupervised approaches, the model is trained on benign traffic and detects attacks that deviate from the learned benign patterns [4]. Although deep learning models, if well-trained, can outperform other automotive IDS models such as non-learning IDSs and traditional ML-based IDSs [16], their outputs are hard to interpret because of their features in design. This also motivates the explainable AI methods introduced in 2.3.

2.2.3 One-Dimensional Convolutional Neural Networks

Convolutional neural networks (CNNs) are a widely used deep learning architecture to learn local patterns from structured input data. Conventionally a CNN consists of one or more convolutional layers, followed by nonlinear activation functions, pooling layers, and fully connected layers. A convolutional layer uses sliding filters across the input to find specific local patterns. An activation layer maps neuron inputs non-linearly to the outputs. A pooling layer downsamples the input feature map by generating only one value from the input feature map. A fully connected layer, usually at the end of a CNN architecture, combines the extracted features to produce the final prediction.

Although CNNs are particularly well known for their application to two-dimensional images, convolution can also be applied to one-dimensional sequential data. In fact, one-dimensional CNNs (1D-CNNs) have achieved exceptional performance levels in fields such as personalized biomedical classification and early diagnosis, structural health monitoring, etc.[14]. In automotive network intrusion detection, where the network traffic is formed by a time series of network packets, recent research has also shown 1D-CNNs are becoming exceedingly capable [16].

2.3 Explainable Artificial Intelligence

Machine learning models often show remarkable predictive performance under experimental conditions, yet their behaviour may become less reliable in real-world deployment. Sometimes they may even be used in actual or potential harmful ways. As a result, calls for "trustworthy AI" and "responsible AI" have intensified in recent years [17]. To achieve this, an essential requirement is the interpretability or explainability of model decisions.

For deep neural networks, explainability is particularly important, because their complex internal process makes it difficult for humans to understand how a particular output is produced. In automotive IDSs, where incorrect or unexplained decisions may affect the security of vehicle states and the drivers, this issue has become especially concerned. In addition to increasing trust and transparency, XAI can support model validation and debugging and provide insights that may help with model improvement.

2.3.1 XAI Concepts

XAI models provide local and global explanations for target models. A local explanation focuses on a specific sample or a small group of samples, and identify which features lead to model predictions. A global explanation aims to describe how the model works across the whole dataset. It reveals features that are generally important as well as common decision patterns [20].

Another categorization of XAI methods is model-intrinsic or -agnostic: model-intrinsic explanations rely on the specific structure of the model itself, such as the coefficients of a linear regression, or the splitting rules of a decision tree; model-agnostic methods do not require knowledge of model structures, and derive explanation merely based on model inputs and outputs [7].

2.3.2 SHAP

SHAP is a post-hoc XAI method. It uses coalitional game theory (Shapley Values) to estimate the importance of each feature. In game theory there are game result and players, which map to model

outputs and different features. Like each player in a game yields gains or costs, SHAP assigns each feature a value representing their contribution for a particular prediction, which can be either positive or negative [23]. The SHAP value ϕ for feature i is defined as:

$$\phi_i = \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!(|F| - |S| - 1)!}{|F|!} \left[f_{S \cup \{i\}}(x_{S \cup \{i\}}) - f_S(x_S) \right]. \tag{1}$$

where F is the set of all input features and S is a subset that does not contain feature i. $f_{S \cup \{i\}}(x_{S \cup \{i\}}) - f_S(x_S)$ represents how much the model output changes when feature i is added to the subset S. The Shapley value ϕ_i is the weighted average of this contribution over all possible subsets $S \subseteq F \setminus \{i\}$ of the remaining features [20].

We use a simple additive feature attribution method[20] in this thesis, where an individual prediction $f(x)$ is represented as:

$$f(x) = E[f(X)] + \sum_{i=1}^{M} \phi_i. \tag{2}$$

where $E[f(X)]$ is the expected model output over the background samples, M is the number of simplified input features, and ϕ_i is the effect the feature i has on the output $f(x)$. Thus, the SHAP values explain how the prediction moves from the baseline output to the output for the current sample [20].

SHAP provides different explainers depending on the type of model being explained. In this thesis, we use GradientExplainer, which is designed for differentiable models such as neural networks and estimates feature attributions using expected gradients [6]. Gradient can be seen as a local coefficient that describes how sensitive the model output is to a feature for a particular input. Expected Gradients evaluates gradients at points between background samples and the sample being explained, and averages the resulting contributions over the background distribution [6].

A positive SHAP value increases the explained model output, while a negative value decreases it. SHAP can produce both local and global explanations that are consistent with each other, where global explanations result from the aggregation of local explanations. In this thesis, SHAP is used for both local explanations of individual predictions and global aggregated explanations of feature importance across attack windows.

2.3.3 TRUSTEE

TRUSTEE [12] is a state-of-the-art XAI tool generating interpretable decision trees for ML models. It is model-agnostic and post-hoc, which means it is applicable to any given black box models. The decision tree explanation generated by TRUSTEE should satisfy the following requirements: first, the predictions of the tree have to be similar to those of the black box model; second, the decision trees have to be of low-complexity so that the selected parts of the tree are comprehensible and can accurately reproduce most predictions made by the black box model; third, the tree should be insensitive to trivial changes in the sampled data, which means repeated training runs will produce trees with similar structures. TRUSTEE gives global explanations summarizing the broader decision behaviour of the model. It aims to enable users to recognize potential inductive biases of the black box model, such as shortcut learning, spurious correlations, and problems with out-of-distribution samples.

2.3.4 Systematic Comparison Approach

SHAP helps gain insight into the feature attribution of individual samples, and its global aggregated feature importance only reveals the behaviour of the majority of data samples, where important features of minority subsets can be averaged away. TRUSTEE gives explicit decision rules of black box models, but it is only an approximation of the models, where the splitting rules can be real behaviour of the black box model, or just the tree's convenient partitions for imitating the model outputs [12]. In [15], a systematic comparison approach is proposed to combine the two XAI methods, where SHAP is applied to subsets corresponding to TRUSTEE root-to-leaf paths, so that the decision rules applied to the subsets by the surrogate tree can be verified by SHAP to show if they are authentic rules relied upon by the black box model. In this thesis we use this approach to verify the reliability of TRUSTEE decision rules.

3 Related Work

This section reviews research related to the two main components of this thesis: ML-based automotive IDSs and the application of XAI for network intrusion detection. The first part focuses on convolutional models for automotive intrusion detection, including approaches based on CAN ID, payload data, and complete CAN frames. The second part discusses XAI methods used to investigate automotive IDSs, emphasizing on feature attribution and surrogate model explanations. These related works provide the foundation for using SHAP and TRUSTEE to analyze the predictions of a CNN-based automotive IDS.

3.1 CNN-based Automotive IDS

A wide range of machine-learning models have been applied to automotive intrusion detection, including conventional classifiers [28], long short-term memory (LSTM) [29], autoencoders [30] as well as convolutional models. Since the baseline used in this thesis is based on one-dimensional dilated CNNs, this section focuses on CNN- and TCN-based approaches.

Early approaches often relied on simple features such as CAN identifiers. Song et al.'s work [27] is one example of applying CNN to automotive network intrusion detection. It makes use of the fact that many attacks involve invalid identifiers of CAN messages, and detects anomalies in CAN identifiers. However, for attacks that use legitimate identifiers but forge malicious payload, this kind of approach is not a suitable solution.

To capture richer attack patterns, apart from identifiers, later approaches also made use of payload information. Cheng et al. [3] proposed a temporal-convolutional model that analyses both the sequence of CAN messages and their content. Each CAN message was represented using CAN identifiers as well as its data field, and sequences of raw messages were transformed into two-dimensional message matrices, which were processed by residual TCN blocks with dilated causal convolutions.

This richer feature representation helps the model detect attacks that mainly alter message content while leaving the timing pattern relatively unchanged. Chiscop et al. [4] investigated the use of TCN for detecting CAN message modification attacks. These attacks modify the payload without changing the timing patterns of CAN traffic, which makes them harder to detect. The proposed method was trained in an unsupervised manner to reconstruct the normal behaviour of CAN signals.

TCN-based approaches further developed this idea by modelling dependencies across sequences of CAN messages. Mei et al. [22] proposed a bidirectional TCN for supervised intrusion detection in in-vehicle CAN traffic. Each CAN message was represented using the timing gap between messages, identifier, and eight-byte payload data. The input windows were then processed by TCN with dilated convolutions. The authors found that fuzzing was more difficult to learn because of its random injection patterns.

The CNN baseline used in this thesis is based on the MR-TCN model proposed by Hellemans et al. [9]. Both models process sequences of CAN frames using dilated one-dimensional convolutions and use information from the CAN identifier, payload, and message timing as input. This allows the models to learn from both the content of CAN frames and the patterns across consecutive messages. Unlike MR-TCN, this thesis focuses on analysing and explaining the CNN's predictions rather than optimizing the model for hardware deployment.

3.2 XAI for Network Intrusion Detection

XAI has also been applied to general network intrusion detection to explain model decisions. Mane and Rao [21], for example, used several post-hoc XAI methods, including SHAP and LIME, to interpret a deep-learning-based NIDS. Their work focuses on conventional network traffic rather than in-vehicle communication.

Later work applied XAI specifically to in-vehicle IDSs. Lundberg et al. [18] used SHAP-based explanations for an in-vehicle IDS and investigated whether these explanations could increase users' trust in the system. This shows that XAI can be used not only to explain model predictions, but also to assess the trustworthiness of automotive IDSs.

Further work introduced automotive domain knowledge into the explanations. Jeong et al. [13] proposed a X-CANIDS framework that converts raw CAN payloads into human-understandable signals using a CAN database, allowing the explanations to indicate which signals or ECUs are related to an attack. This provides more meaningful explanations than directly interpreting raw payload bytes.

More recently, explainability has been integrated into practical in-vehicle IDS frameworks. Ding et al. [5] proposed DeepSecDrive, which combines a lightweight deep-learning detector with explainability for real-time in-vehicle intrusion detection. Similarly, Aljabri et al. [1] applied SHAP to an in-vehicle IDS with entropy- and time-based features to verify how these engineered features contribute to its predictions.

Besides explaining individual IDSs, XAI methods can also be compared to examine whether they reveal consistent model behaviour. Kumar and Thing [15] applied TRUSTEE and Kernel SHAP to several state-of-the-art ML-based NIDSs and conducted systematic comparison of the explanations of both methods. For selected root-to-leaf paths in the TRUSTEE tree, SHAP was applied to the samples following those paths, and the tree features were compared with the features ranked important by SHAP. This comparison strategy is adapted in this thesis to verify that the global classification rules of TRUSTEE surrogate trees are real model behaviour, instead of artefacts of the trees.

4 Dataset and Preprocessing

4.1 Dataset

This thesis uses the Controller Area Network and Automotive Ethernet Realistic Data Set (CarDS) developed by Hellemans et al. [8]. It is a novel dataset captured from a modern commercial vehicle manufactured in 2020. The vehicle's IVN consists of 10 internal CAN buses and 6 Automotive Ethernet (AE) connections. Although CarDS contains both CAN and Automotive Ethernet traffic, this thesis only uses the CAN portion of the curated dataset. Figure 2 presents the CAN network architecture of the vehicle used to collect CarDS. The 10 main CAN buses are connected through a Gateway ECU (GWE), which forwards selected messages between CAN buses [8].

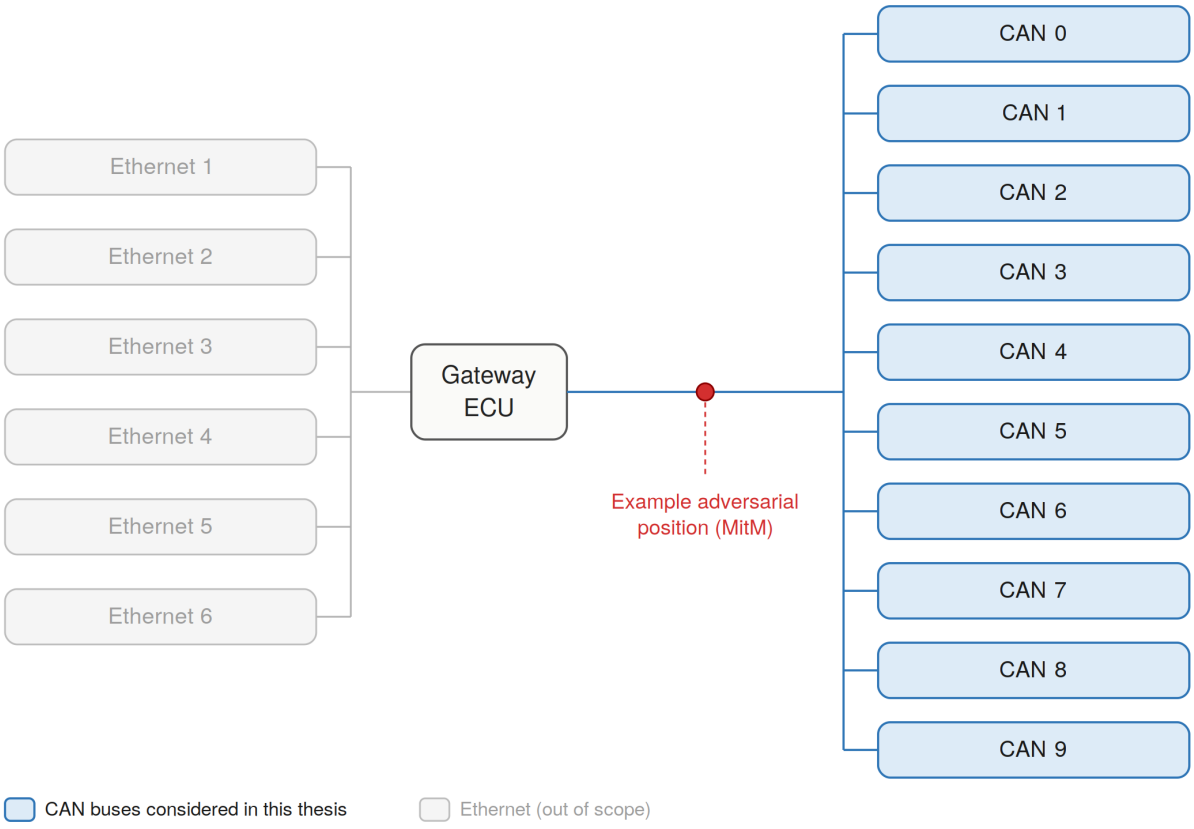


Figure 2: Overview of CAN network architecture, adapted from Hellemans et al. [8]

There are 9 different attack scenarios in the dataset: benign, replay, spoofing, fuzzing, advanced, DoS attacks, V-mode attacks, UDS attacks, and Automotive Ethernet Pentesting. Apart from the general attacks introduced in 2.1.2, advanced attacks and V-mode attacks are specially curated in the dataset and are discussed here.

V-mode attack in the CarDS dataset is the first publicly available real masquerading attack. It physically separates the CAN bus and assumes a man-in-the-middle position between a CAN

bus and the gateway ECU [8]. It has stronger capabilities than normal message injection attacks, because apart from reading and injecting messages, the attackers can also block, delay or modify the messages they intercepted.

Advanced attack category contains targeted message-injection attacks that account for message counters and cyclic redundancy checks (CRC). Instead of merely injecting a selected identifier and payload, the attacker constructs the counter and CRC values consistently with the messages currently observed on the bus. This makes the malicious frames more similar to legitimate communication and increases the probability that they will be accepted by receiving ECUs [8].

4.2 Feature Engineering

Raw CAN logs cannot be directly processed by the CNN. Therefore, we need to perform feature engineering.

CAN data in the CarDS dataset contain CAN CC and CAN FD messages. Figure 3 presents two examples of CAN CC and CAN FD message from the dataset. As we can see, each CAN message contains a timestamp, the interface on which the message was observed, a CAN identifier, a payload, and a label. For example, the CAN CC message in Figure 3 represents a message observed on interface `can8` at the specified timestamp. Its hexadecimal CAN identifier is `1A4`, and its payload consists of the eight bytes `CA 37 CA 30 08 04 00 00`. The final label R denotes a benign message and T denotes a malicious message.

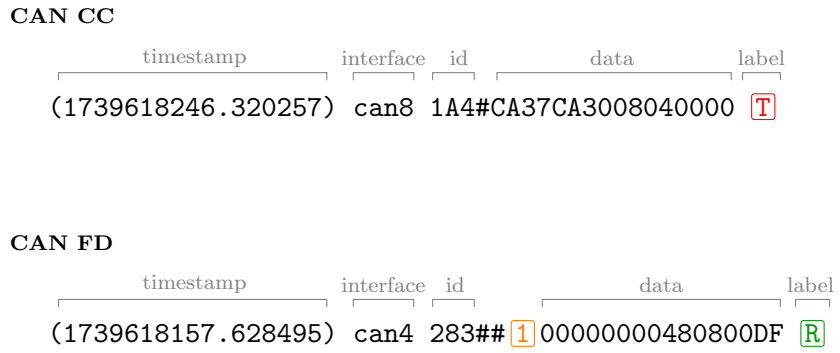


Figure 3: Example CAN CC and CAN FD data traces

We construct input features for the CNN model. First we make use of identifiers and payload bytes, which are raw bytes from the original CAN message. Since the CAN identifier is of either 11 or 29 bits long, we apply zero-padding and represent it as 4 bytes. Additionally, since CAN CC can have payload length of up to 8 bytes, while CAN FD can have 64 bytes, to make sure input features have equal length, we zero-pad both CAN message payload to 64 bytes. Then, since the interface number ranges from 1 to 10 but has no ordinal meaning, we apply one-hot encoding, resulting in 10 binary interface features. The protocol of CAN data trace is also represented, where CAN CC is represented as 0 and CAN FD is represented as 1. The length of the payload data is also added as a feature. Finally, we compute two temporal features using timestamps: the time difference with the previous message on the same interface, as well as the time difference with the previous message of the same ID on the same interface.

To keep a consistent scale of each features, we perform min-max normalization on ID, payload bytes and payload length features; for delta time features, we first do logarithmic conversion and

then standardize them using mean and standard deviation.

Table 1: Feature vector layout and encoding

Feature indices	Features	Dimensions	Encoding
0–3	CAN identifier	4	Four bytes, divided by 255
4–67	Payload	64	Zero-padded bytes, divided by 255
68–77	CAN interface	10	One-hot encoding
78	Protocol	1	CAN CC = 0, CAN FD = 1
79	Payload length	1	Normalized length
80	Delta time on same interface	1	Log transform and standardization
81	Delta time for same interface and identifier	1	Log transform and standardization

After parsing, each message is converted into an 82-dimensional feature vector. The vector contains 4 identifier features, 64 payload features, 10 interface features, 1 protocol indicator, 1 payload length feature, and 2 temporal features.

4.3 Window Construction and Class Distribution

Individual CAN messages provide only a limited view of the current network. Although some attacks may introduce clearly abnormal identifiers or payload values, other attacks cannot necessarily be identified from a single message. For example, replay attacks use legitimate identifiers and payloads, but their occurrence in a series of messages may be out of context. Therefore, we group consecutive CAN messages into fixed-size windows, allowing the model to analyse both the content of individual messages and the communication context in which they occur.

In this thesis, we set the window size to be 64. Thus, each input sample consists of 64 consecutive CAN messages. Since every message is represented by an 82-dimensional feature vector, a window is represented as

$$\mathbf{X} \in \mathbb{R}^{82\times 64}.$$

The feature dimension forms the input channels of the one-dimensional CNN, while the window dimension preserves the chronological order of the messages. A window is labelled as malicious if it

contains at least one malicious message; otherwise, it is labelled as benign.

Since the curated CarDS dataset only provides training and test sets but no separate validation set, we divide each log file in the curated training set into training and validation data. The first 80% of the messages in each file are assigned to the training set, while the remaining 20% are assigned to the validation set. This way of split preserves the original message order and reduces the risk of closely related traffic appearing in both training and validation sets. Feature engineering and window construction are then performed separately for the training, validation, and test data. The resulting dataset composition after preprocessing is presented in Table 2. The resulting class distributions in the three subsets are presented in Table 3. From Table 3 we can see all three subsets are imbalanced, with the number of benign windows being considerably larger than those of attack windows. Besides, the ratio of attack windows in the test set is roughly two times of those in the training and validation sets.

Table 2: Composition of the dataset after splitting and processing with window size 64

Split Category	Train	Validation	Test
advanced	75554	18887	29781
benign	299385	74841	48402
DoS	45372	11342	24483
ethernet	95418	23852	93107
fuzzing	87177	21792	60355
replay	92711	23176	49178
spoofing	74967	18741	35230
UDS	220894	55223	153088
v-mode	98135	24532	51047
Total	1089613	272386	544671

Table 3: Overall window-level class distribution across the training, validation, and test sets

Split	Total windows	Benign windows	Attack windows	Benign ratio	Attack ratio
Train	1089613	1011535	78078	92.83%	7.17%
Validation	272386	254554	17832	93.45%	6.55%
Test	544671	473294	71377	86.90%	13.10%

Table 4: Layer-wise architecture of the dilated 1D CNN baseline. Tensor shapes are written as (batch, channels, length).

Layer	Input	Output	Kernel	Stride	Dilation	Padding
Conv1D	$(1, 82, 64)$	$(1, 64, 64)$	(3)	(1)	(1)	(1)
Dropout	$(1, 64, 64)$	$(1, 64, 64)$	-	-	-	-
Conv1D	$(1, 64, 64)$	$(1, 128, 64)$	(3)	(1)	(2)	(2)
Dropout	$(1, 128, 64)$	$(1, 128, 64)$	-	-	-	-
Conv1D	$(1, 128, 64)$	$(1, 128, 64)$	(3)	(1)	(4)	(4)
AdaptiveMaxPool1D	$(1, 128, 64)$	$(1, 128, 1)$	-	-	-	-
Flatten	$(1, 128, 1)$	$(1, 128)$	-	-	-	-
Linear	$(1, 128)$	$(1, 64)$	-	-	-	-
Dropout	$(1, 64)$	$(1, 64)$	-	-	-	-
Linear	$(1, 64)$	$(1, 1)$	-	-	-	-

5 Experimental Setup

5.1 CNN Baseline

We use a one-dimensional dilated CNN as the baseline model. A 1D-CNN applies convolutional filters along a single dimension. In this thesis, this dimension refers to a sequence of 64 CAN messages, and the channels refer to the extracted 82 features [10]. The model has three convolutional blocks with kernel size 3 and dilation rates of 1, 2 and 4. The dilation lets the receptive field grow from 3 to 7 to 15 messages, and zero padding of 1, 2 and 4 keeps the length fixed at 64 [9]. Because this padding is applied symmetrically on both sides, the convolutions are non-causal, with each output position aggregating messages both preceding and following it within the window. Each block applies batch normalization and a ReLU activation, with dropout (0.3) after them.

We then apply global max pooling over the temporal dimension, reducing the 128×64 feature map to a 128-dimensional vector by taking each channel's maximum output across the window. Finally, the classification is done by two fully connected layers with a ReLU and dropout between them, producing a single logit that is converted to an attack probability. Table 4 summarizes the architecture of the baseline model.

The model is trained with binary cross-entropy on the logit. To counter the class imbalance between benign and attack windows, the loss is weighted by the ratio of benign to attack samples in the training set. We use the Adam optimizer and the BCEWithLogits loss function. The model is trained using a batch size of 64 for up to 50 epochs, using early stopping with a patience of 10 epochs on the validation macro-averaged F1 score; the checkpoint with the best validation macro-F1 is saved for test evaluation.

5.2 SHAP Setup

To interpret the trained CNN, a post-hoc SHAP analysis is performed with the GradientExplainer [6], which approximates SHAP values through expected gradients. The explainer is applied to the raw attack logit rather than the sigmoid probability, since gradient-based attributions are generally

more stable before the bounded sigmoid transformation [24].

The trained model is run over the full test set using the threshold of 0.785, which is the best threshold selected on the validation set. Each window is labelled as TP (True Positive), FN (False Negative), FP (False Positive), or TN (True Negative). 500 random samples are drawn from the attack windows for explanation. As SHAP values are defined relative to a reference distribution, a background set of 200 windows is sampled from the benign data, helping to investigate which features distinguish the attack samples from benign samples.

For each explained window, SHAP assigns a contribution value to every input feature at each of the 64 message positions, resulting in a matrix of size $(82, 64)$. In this study, a positive SHAP value means the value of the feature pushes the prediction to be attack while a negative SHAP value does the contrary [15], and its magnitude indicates the strength of this effect. These values are then summed over the 64 message positions to obtain one contribution per feature for the global beeswarm analysis and the local waterfall plots.

5.3 TRUSTEE Setup

TRUSTEE is used to extract a global surrogate decision-tree of the trained CNN. Each sample is a window of 64 CAN messages. Instead of flattening the original 82×64 input, we compute statistical values of each feature over the 64 message positions. Mean, maximum, minimum, and standard deviation are used for the CAN ID, payload, payload length, and timing channels, while only the mean is used for the binary interface one-hot and protocol channels. This results in $71 \times 4 + 11 = 295$ aggregated features per window. The CNN predictions are still computed from the original windows, and TRUSTEE is trained to reproduce these predictions.

The surrogate tree is fitted on 10,000 training windows and evaluated on 10,000 independently sampled test windows using the natural class distribution. The CNN threshold is 0.785. TRUSTEE is run with 50 inner iterations, 10 stability iterations, and 3000 samples per iteration. The resulting full tree is pruned using the top-10 nodes. Fidelity is evaluated by comparing the full and pruned tree predictions with the CNN predictions on the held-out test set.

5.4 Systematic Comparison Setup

The systematic comparison applies SHAP to samples corresponding to the main decision paths of the surrogate tree. The 10,000 evaluation windows from TRUSTEE analysis are routed through the tree. For each leaf, up to 300 windows are randomly sampled for explanation, and SHAP values are computed on the original 82×64 windows with the GradientExplainer, using a background set of 200 windows drawn from the TRUSTEE fitting data.

The tree and SHAP operate on different feature spaces: the tree tests 295 aggregated features as shown in 5.3, while SHAP attributes contributions to every feature-position combination of the raw window. Therefore, both sides are projected onto the shared axis of 82 features. For the TRUSTEE tree, the statistic suffix of each aggregated feature tested along a root-to-leaf path is stripped, leaving its base channel. On the SHAP side, the contribution of every feature is obtained by averaging absolute SHAP values over the 64 message positions and over the explained windows of the leaf. For example, the feature `delta_same_id_interface__min` in a TRUSTEE tree path will be turned into `delta_same_id_interface`, whose SHAP score is the mean absolute SHAP value over its 64 positions.

6 Results and Discussion

In this section we present the results of CNN performance and XAI analysis, as well as discuss the consistency of local and global explanation.

6.1 Evaluation Metrics

Several metrics are used for performance evaluation. First of all, we have the true positives (correctly identified attacks), true negatives (correctly identified benign traffic), false positives (benign traffic labeled as attack) and false negatives (attacks labeled as benign traffic). Based on these metrics we can calculate the following:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{5}$$

$$\text{F}_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{6}$$

$$\text{Macro F}_1 = \frac{\text{F}_{1,\text{benign}} + \text{F}_{1,\text{attack}}}{2} \tag{7}$$

6.2 CNN Results

This section shows the CNN evaluation results. We first report its overall performance on the test set, and then perform per-category error analysis to investigate how the model performs on different attack categories.

6.2.1 Performance Evaluation

Table 5 reports the performance of the baseline CNN on the test set. The decision threshold 0.785 was selected on the validation set that maximises the macro-averaged F_1 score. The model achieves an overall accuracy of 0.961 and a macro F_1 of 0.908. This is strong performance across both the attack and the benign classes despite the substantial class imbalance in the data. On the attack class specifically, it reaches a precision of 0.921 and a recall of 0.769, for an attack F_1 of 0.838.

Figure 4 shows the distribution of the predicted attack probabilities on the test set. Most predictions are concentrated near 0 and 1, meaning the model is highly confident on most predictions. This also explains why the macro-F_1-optimal threshold is 0.785: with few predictions in the middle range, the threshold can be pushed towards 1 to suppress false positives without creating many false negatives. From the figure we also find a noticeable number of attack samples receiving a predicted probability close to 0, far below the decision threshold. These misses cannot be recovered by lowering the threshold, because the model produces almost the same low probability for them as benign samples. No threshold can separate them from the benign class without also yielding a large

Table 5: Performance of the dilated 1D CNN on the test set

Metric	Value
Accuracy	0.961
Benign precision	0.966
Benign recall	0.990
Benign F_1	0.978
Attack precision	0.921
Attack recall	0.769
Attack F_1	0.838
Macro precision	0.943
Macro recall	0.880
Macro F_1	0.908
False positive rate	0.010
False negative rate	0.231

Table 6: Per-category detection performance on the test set

Category	Attacks	Precision	Recall	F1	FN	% of FN
v-mode	11,399	0.484	0.107	0.176	10,176	61.8
spoofing	348	0.737	0.330	0.456	233	1.4
fuzzing	12,852	0.790	0.676	0.729	4,163	25.3
advanced	718	0.834	0.912	0.872	63	0.4
replay	25,616	0.986	0.930	0.957	1,797	10.9
UDS	7,519	0.987	0.995	0.991	36	0.2
DoS	12,925	1.000	0.999	1.000	9	0.1

number of false positives. So these missed attack samples do not result from being put below the threshold value, but they are represented as benign. To investigate which categories these missed attacks come from, and why the model fails to distinguish them, we perform a per-category error analysis in section 6.2.2.

6.2.2 Error Analysis

We decompose the model's performance by attack category at threshold 0.785 to see if there are specific categories the model particularly fails at. Table 6 reports per-category recall and false negatives. From the table we can see DoS and UDS attacks are detected almost perfectly, at recall 0.999 and 0.995, and replay and advanced attacks well above 0.9. However, for stealthier attacks like fuzzing and spoofing, recall falls to 0.676 and 0.330. For v-mode, the recall even drops to 0.107, meaning that the model detects only around 10% of the attack windows in this category.

A single category, v-mode, accounts for 61.8% of all 16,477 false negatives, and together with fuzzing and replay for roughly 98%. The overall false negative rate is therefore driven almost

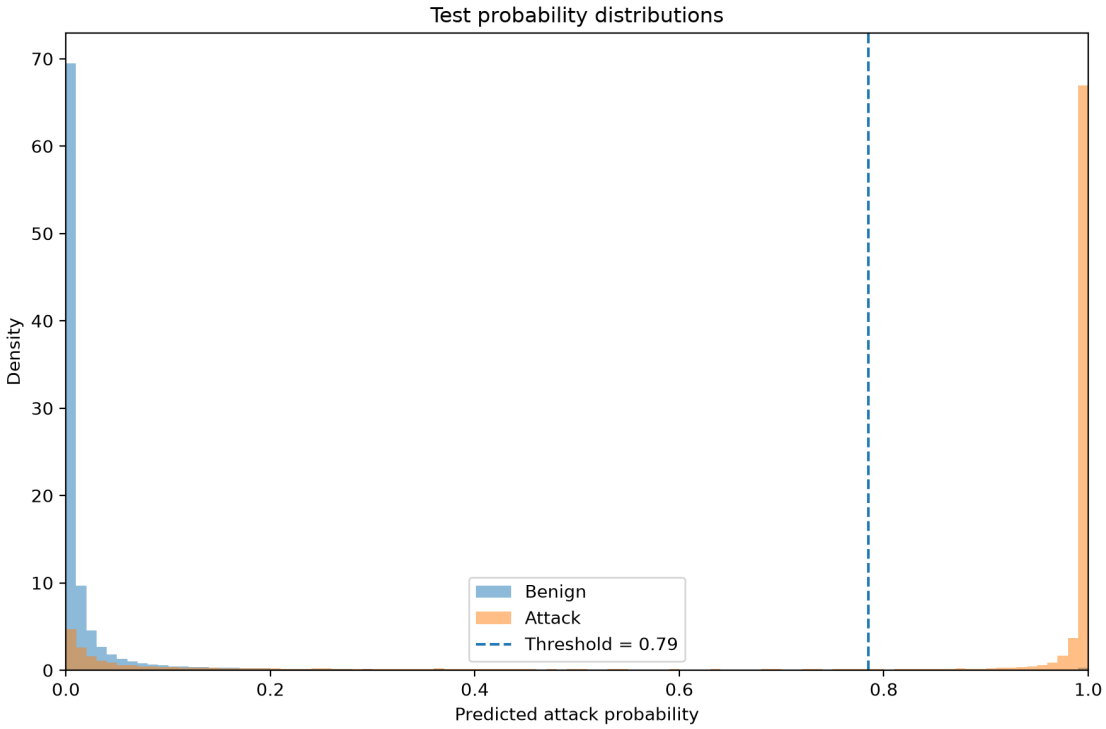


Figure 4: Probability distribution of the test set

entirely by a few categories. Moreover, for the hardest categories, the median predicted probability for actual attack windows is very low: 0.163 for v-mode and 0.193 for spoofing, while for DoS, UDS and replay the median is > 0.99. In other words, the model truly identifies the missed attack samples from v-mode and spoofing as benign, instead of missing them because the threshold happens to be higher than their predicted probabilities. This suggests the model has learned to flag attacks that alter the timing or frequency of CAN traffic, while overlooking content manipulations that do not disturb the overall timing pattern. To identify which features drive these decisions and why the model performs well on some attack types but poorly on others, we turn to the XAI analysis using SHAP and TRUSTEE.

6.3 SHAP Results

This section presents the SHAP results of the CNN predictions. We first examine local explanations for selected true-positive and false-negative attack windows to understand individual decisions, and then aggregate SHAP values across attack windows to identify broader feature-importance patterns.

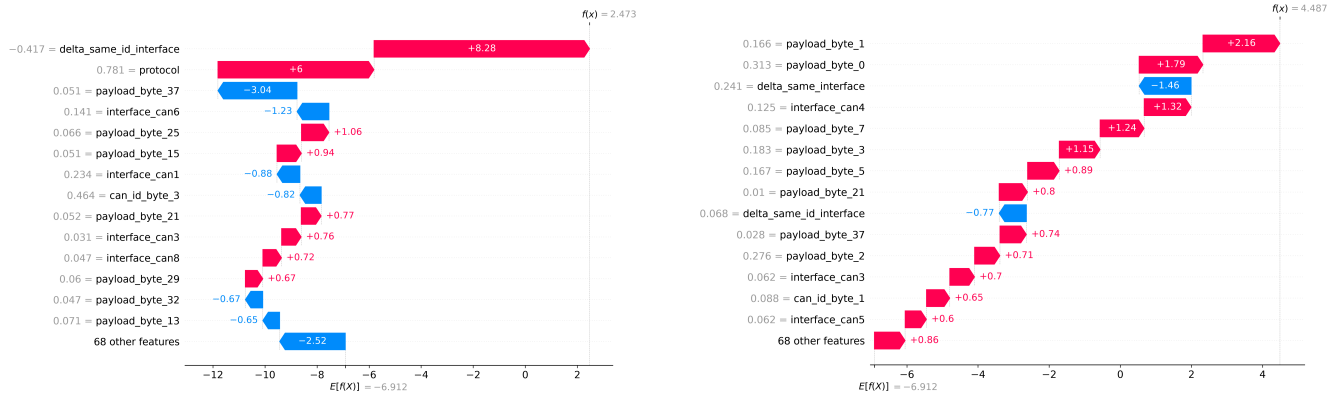
6.3.1 Local Explanation

Figures 5a and 5b present local SHAP explanations for two correctly detected attack windows, one from the replay category and the other from the fuzzing category. The horizontal axis represents each feature's contribution to the predicted logit. A positive value means the feature pushes the

prediction to attack; a negative value means the contrary. The base value, $\mathbb{E}[f(X)] = -6.912$, is the expected model output over the 200 background benign windows used in the SHAP analysis. The value $f(x)$ represents the predicted logit for the explained window. The actual values of the features are displayed in grey next to their names.

For the replay window, the model produces a predicted logit of $f(x) = 2.473$. The prediction is primarily driven by the `delta_same_id_interface` feature, which contributes approximately $+8.28$ to the logit. This accounts for roughly three quarters of the total shift from the baseline value of -6.912. The observed value of `delta_same_id_interface` is -0.417. Since this feature was standardized using the training-set mean and standard deviation, the negative value indicates that the time interval was smaller than its average training-set value. Many CAN messages are transmitted periodically, so an injected message will reduce the period. This is consistent with the characteristics of replay attacks, in which previously recorded CAN messages are retransmitted. Although the CAN identifiers and payload contents are valid, the repeated occurrence of messages with the same ID and interface reduces the inter-arrival time.

The fuzzing window shows an opposite pattern. Instead of being dominated by a single feature, its predicted logit of $f(x) = 4.487$ results from the accumulation of multiple moderate positive SHAP contributions, mainly associated with payload bytes and CAN ID bytes. This also reflects the nature of fuzzing, in which the attacker injects messages with randomized payload bytes or identifiers. These two local explanations suggest that the CNN learns different detection mechanisms for different attacks.



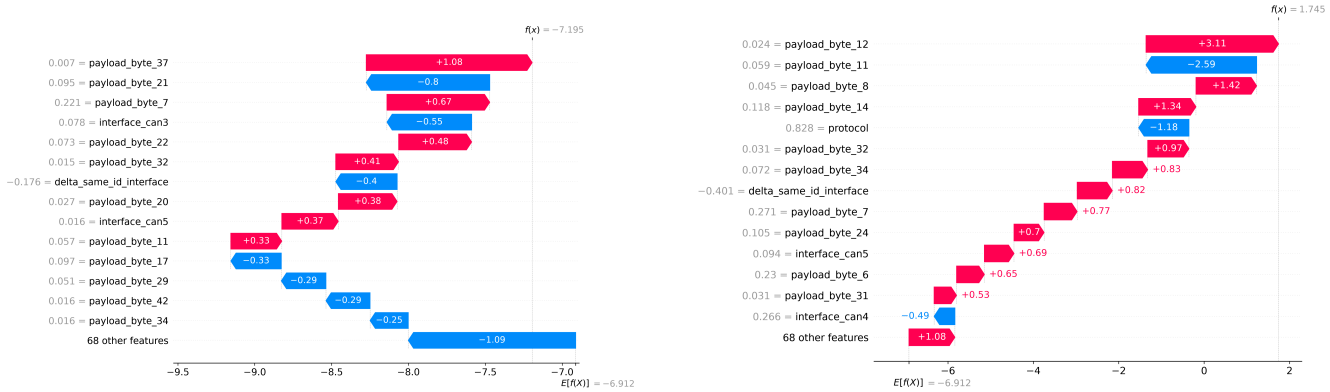
(a) A TP sample from the replay category

(b) A TP sample from the fuzzing category

Figure 5: SHAP values for true-positive replay and fuzzing samples

Figures 6a and 6b show two missed *v-mode* windows that represent different failure modes. In the first one the predicted logit is -7.195, which is below the threshold logit as well as the benign baseline. From the graph we can see no feature provides a substantial positive contribution, and the same `delta_same_id_interface` feature that drove the replay detection in Figure 5a here pushes the prediction towards benign despite both taking negative values. This shows that the CNN does not just learn the simple rule that shorter `delta_same_id_interface` values result in attack windows, but the classification may also depend on other features' value in the same window [19]. Also, the extremely low attack probability of this window means the CNN sees the window as entirely normal. This results from the limitation of the current CNN's feature representation [26]. The second window, in contrast, receives positive contribution from multiple payload bytes. Its

predicted attack probability of 0.782 falls very close to the 0.785 decision threshold. Therefore, we can say the missed detections do not occur for the same reason. Some cases can be recovered by adjusting the threshold, while other deep misses would require better feature representation of the baseline model.



(a) A FN sample from the v-mode category

(b) Another FN sample from the v-mode category

Figure 6: SHAP values for false-negative v-mode samples

6.3.2 Global Explanation for Attack

Figure 7 presents an aggregated global explanation over the 500 explained attack windows, where each point corresponds to one window, the horizontal position represents the total contribution of a feature to the predicted logit (summed over the 64 timesteps), and the colour encodes the feature values averaged over the window. The features are ranked using their mean absolute SHAP values from high to low. Since the explained set only consists of attack windows, the plot reveals what features the model in general relies on when detecting attacks. Therefore, most point clusters lie on the positive side relative to the benign background.

The most important feature, `delta_same_id_interface`, exhibits two clear clusters in the figure. One cluster of windows receives large positive contributions (up to +20, mostly for low feature values), which indicates for this cluster of windows, short time difference between messages of the same identifier from the same interface pushes their prediction a lot toward attack. The other cluster lies near zero, which shows for this cluster the time difference feature does not have much influence on their prediction. This is consistent with the local analysis in Section 6.3.1, where it drove the replay detection but did not contribute to the fuzzing window. The `protocol` feature ranks second with mostly moderate positive contributions, suggesting that the protocol composition of a window co-varies with attack traffic in the dataset; this may partly reflect dataset characteristics rather than attack patterns. `interface_can9` shows a distinctive split: windows with a high share of traffic on can9 receive strongly positive contributions, while the remaining windows whose traffic is not on can9 cluster around 0. The remaining top features mostly are individual payload and identifier bytes with small contributions, which is in accordance with the fuzzing explanation in Section 6.3.1 with spreaded content-based signals. Overall, the global view supports the conclusion that the CNN has two detection mechanisms: timing-based and content-based.

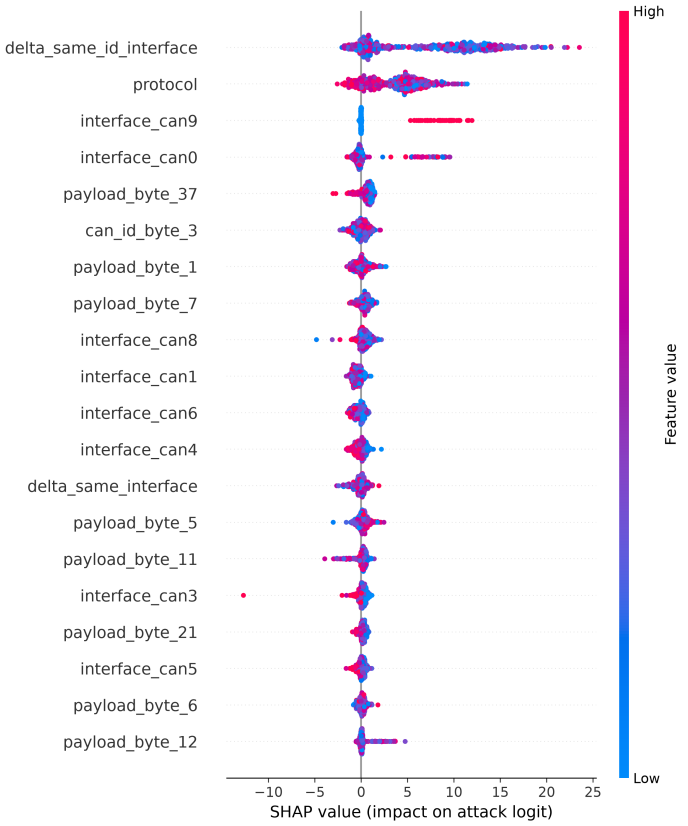


Figure 7: Global feature importance for attack windows

Table 7: Held-out fidelity to the CNN and predictive performance against ground truth

Model	Ref.	Acc.	Macro F_1	Attack class			FPR	FNR
				Prec.	Rec.	F_1		
Full tree	CNN	0.954	0.883	0.792	0.791	0.791	0.026	0.209
Pruned tree	CNN	0.961	0.883	0.984	0.657	0.788	0.001	0.343
CNN	Truth	0.958	0.901	0.918	0.751	0.826	0.010	0.249
Full tree	Truth	0.926	0.826	0.772	0.631	0.694	0.029	0.369
Pruned tree	Truth	0.934	0.822	0.963	0.526	0.681	0.003	0.474

Table 8: Held-out confusion matrices of the surrogate trees

(a) Full tree vs. CNN

		Tree prediction	
		Benign	Attack
CNN prediction	Benign	8,676	228
	Attack	229	867

(b) Pruned tree vs. CNN

		Tree prediction	
		Benign	Attack
CNN prediction	Benign	8,892	12
	Attack	376	720

6.4 TRUSTEE Results

This section evaluates how well the TRUSTEE surrogate tree approximates the behaviour of the CNN, and analyses the decision rules of the surrogate tree. We first assess the fidelity of the full and pruned trees on held-out samples, then analyse the pruned tree to identify the main decision patterns learned from the CNN. Finally, we investigate predictions where the surrogate disagrees with the CNN by attack categories to better understand which model behaviours are not captured by the surrogate tree.

TRUSTEE converged with agreement 0.993 and fidelity reward 0.936, yielding a full surrogate tree of 95 nodes (depth 17) and a compact pruned surrogate tree of 3 nodes (depth 1).

6.4.1 Fidelity and Performance Evaluation

Table 7 reports the trees' held-out fidelity to the baseline CNN and performance against ground truth on the $10,000$ evaluation samples. Table 8 shows the confusion matrices of the full tree and the pruned tree with respect to the CNN's predictions.

Notably, the pruned tree achieves a higher accuracy of 0.961 than the full tree and a similar macro F_1 of 0.883, despite containing only three nodes. However, pruning changes the type of disagreement with the CNN by trading a loss in attack recall ($0.791 \rightarrow 0.657$) for precision gain ($0.792 \rightarrow 0.984$). This is also visible in the confusion matrix where the number of false positives falls from 228 for the full tree to only 12 for the pruned tree, while the number of false negatives increase from 229 to 376. Comparing with the CNN, both trees are conservative: against ground truth the pruned tree's false-positive rate (0.3%) is below the CNN's (1.0%), but it recovers only 52.6% of attacks versus the CNN's 75.1% and has almost twice the CNN's false negative rate; the full tree, although having a higher recall of 63.1% than the pruned tree, still fails the CNN at this

metric. Thus, the fidelity gap between the CNN baseline and surrogate trees mainly lies in the trees missing CNN's attack predictions, which is discussed in 6.4.4.

6.4.2 Effect of Feature Aggregation

The per-channel aggregation reduces the number of dimensions of the TRUSTEE input features, avoiding the issue known as "the curse of dimensionality" [25], where models, facing a high-dimensional feature space, may exploit spurious correlations and learn various shortcuts [12]. Besides, the aggregated statistical features make the decision rules easier to interpret. For example, `delta_same_id_interface__min` means the shortest time gap between two messages with same id and interface, which has richer semantic meaning than `delta_same_id_interface__at_position_50`.

However, the aggregation also removes information from the original 82×64 input features. In particular, it does not preserve the exact temporal positions of individual messages, or the co-occurrence of certain features in the same message. Two windows with different message sequences may therefore have similar aggregated statistics. As a result, message-level feature combinations cannot be represented by the surrogate tree. These limitations should be taken into consideration when evaluating the fidelity of the surrogate tree to the CNN model.

6.4.3 Pruned Tree Analysis

Figure 8 shows the pruned tree of 3 nodes (depth 1), where each box represents a node. Although the pruned tree achieves high overall fidelity, pruning reduces its recall of the CNN's attack predictions from 0.791 to 0.657. We think it is necessary to preserve and analyse some other high priority branches in the full tree. Figure 9 therefore preserves the root split together with the next split on its right branch, while collapsing the remaining subtrees for readability. The complete tree is presented in Appendix A.

In both trees, the first line of internal nodes is the splitting rule. Target samples satisfying the rule go to the left branch, while the others go to the right. The second line is the Gini Impurity of the node, measuring how mixed the labels are among samples reaching it. A value of 0 means the node is all attack or benign; 0.5 means an even 50/50 split. The third line is the percentage of samples reaching the node, which starts at 100% at the root. The fourth line shows the class distribution of the node, with the first element being the benign fraction and the second being the attack fraction. (Note: these are the CNN's predicted labels, since the tree is trained to mimic the CNN). The fifth line represents the majority class of the node. If the node is a leaf node, then it is the outputs of the tree. Color of the nodes encode class information: orange for benign, and blue for attack, with saturation indicating purity.

The pruned tree reconstructs 96.1% of the CNN's held-out decisions, as shown in Table 7. The root and the only split tests whether the minimum time difference between messages of same ID and interface (`delta_same_id_interface__min`) is at most 3.41 ms; 96.6% of windows do not satisfy this and are labeled benign, with 96.5% agreement with the CNN; the remaining 3.4% windows satisfy this and are labeled as attack, with 95.8% agreement with the CNN.

In the adapted full tree, the first split is the same as the pruned tree. The second split does a further division on the right brancg, which concerns the mean value of the can9 feature. Since the mean value is calculated over the 64 messages ina window, this rule is essentially asking if there exists at least one message from can9 in the window. If there is, the window is labeled as attack,

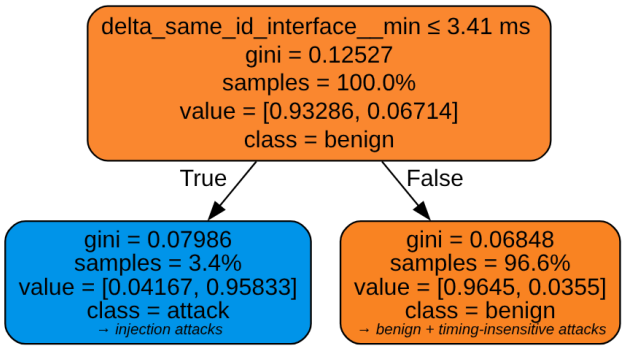


Figure 8: Pruned decision tree from TRUSTEE analysis

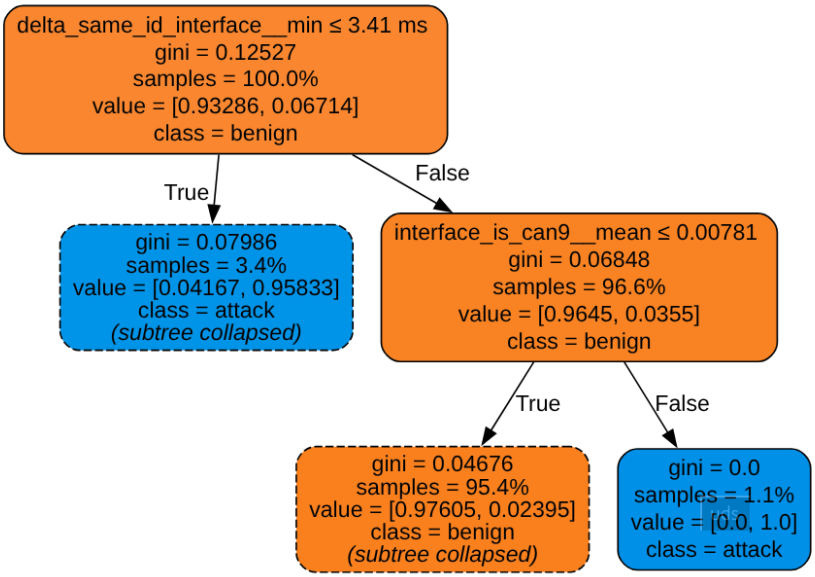


Figure 9: Selected shallow view of the full decision tree

Table 9: Attack categories of samples reaching the two attack leaves

Leaf $p1/f1$ — timing rule			Leaf $f3$ — can9 rule		
Category	Attack	Benign	Category	Attack	Benign
replay	434	8	UDS	140	0
DoS	238	0	benign	0	7
fuzzing	21	16			
advanced	11	1			
v-mode	1	0			
spoofing	0	2			
Total	705	27	Total	140	7
Precision: 96.3%			Precision: 95.2%		

which constitute 1.1% of all samples. If not, the window is benign.

We name the left leaf in Figure 8 $p1$, and the right leaf $p2$, and the 3 leaves in Figure 9 from left to right $f1$, $f2$, and $f3$. Note that leaf $f1$ is a collapsed subtree whose entry condition is identical to the root split of the pruned tree; the windows reaching $f1$ are therefore exactly the 732 windows of leaf $p1$, so we use the two labels interchangeably. Table 9 shows the source of the samples reaching leaf $p1/f1$ and $f3$. Categories refer to the attack type of the source log file. Since samples from attack log files can still be labelled as benign, the ground truth label of the samples are also shown to reveal whether they are false positives. From the table we can see that samples reaching leaf $p1/f1$ are mostly injection attacks like replay and DoS, which is in accordance with the splitting rule whether (`delta_same_id_interface__min`) is at most 3.41 ms. Among the samples reaching leaf $f3$, most of them are from the UDS attack log file and are all true positives, with only 7 samples from the benign log file and are all false positives. Since the related splitting rule is about the presence of can9, this suggests a potential interface-specific shortcut.

In summary, the pruned tree shows that the CNN's dominant decision path is a timing-anomaly detector on same-ID same-interface arriving gap. This path is sensitive to injection attacks such as replay and DoS, but remains blind to content altering attacks, which is discussed in Section 6.4.4. The adapted full tree shows that apart from the timing pattern, the tree also relies on the presence of can9 to detect UDS attack, which suggests a potential interface-specific shortcut and is analyzed in Section 6.5.

6.4.4 False Negatives Analysis

Table 10 decomposes the pruned tree's 376 false negatives with respect to the CNN by traffic category. These false negatives are predicted as attacks by the CNN but as benign by the surrogate tree. "CNN prob." is the mean CNN attack probability on missed windows; "true attack" is the fraction of missed windows that are attacks under ground truth. The ethernet category had no CNN-flagged windows.

The false negatives are concentrated in fuzzing and UDS. They account for 299 of the 376 missed CNN attack predictions, or approximately 79.5% of all false negatives. Attacks altering timing patterns of the traffic are reproduced almost perfectly: the pruned tree misses no DoS windows

Table 10: Pruned-tree false negatives against the CNN

Category	CNN-flagged	Pruned tree FN	Pruned tree Miss	Full tree FN	Full tree Miss	CNN prob. on FN	True attack frac.
fuzzing	192	158	82.3%	153	79.7%	0.974	0.80
UDS	141	141	100%	1	0.7%	0.986	0.99
v-mode	46	45	97.8%	43	93.5%	0.905	0.47
replay	452	18	4.0%	18	4.0%	0.954	0.94
DoS	238	0	0.0%	5	2.1%	—	—
benign	8	8	100%	3	37.5%	0.893	0.00
spoofing	5	4	80.0%	4	80.0%	0.921	0.50
advanced	14	2	14.3%	2	14.3%	0.907	0.00

and only 4.0% of replay windows. This is consistent with Section 6.4.3, where the tree encodes specific timing patterns as attack signals. An interesting phenomenon here is that the pruned tree performs better than the full tree on the DoS category, which suggests deeper nodes focusing on ID and payload bytes may disturb the correct decision on this category made by important nodes.

The tree does not perform so well on the other attacks. It misses all the attack predictions of the CNN on the v-mode and UDS categories, and around 80% of the attack predictions on the spoofing and fuzzing categories. There are multiple causes for these false negatives. For the UDS category, we can see the full tree reproduces the CNN predictions quite well, missing only one attack sample from all the 141 CNN predicted attacks. This means nodes identifying the UDS attack patterns exist in the full tree, but are all removed in the pruned tree. By contrast, fuzzing, v-mode, and spoofing are missed at high rates by both trees, although the full tree performs slightly better than the pruned tree. Since these three attacks are content-based, the performance gap between the full tree and pruned tree results from pruning the splitting rules relating to payload and ID bytes. The performance gap between the full tree and the baseline CNN may result from information lost through the per-channel aggregation discussed in 6.4.2.

Finally, not all surrogate errors are errors with respect to the ground truth. Looking at the last column of ground-truth attack fraction, we can see all FNs of the benign and advanced categories, as well as half of the v-mode FNs are windows where the CNN itself falsely alarms, so the tree's disagreement actually improves its accuracy against truth.

6.5 TRUSTEE vs SHAP Analysis

Table 11 presents the SHAP–TRUSTEE agreement over the samples of the two leaf nodes of the pruned surrogate tree. For each subset of data samples reaching the leaf node, T is the set of features used in the selected path and S is the set of features identified by SHAP that contribute most to the prediction, and the rank is the worst position of the T-feature in the SHAP ranking, where 1 means the most important of the 82 features. $T \subseteq S@k$ means T is in the top-k SHAP features.

From the table we can see among the 10,000 samples selected for TRUSTEE evaluation, 732 of them reaching leaf $p1$ are classified as attack by the surrogate tree. Among the averaged SHAP

Table 11: SHAP–TRUSTEE agreement for the pruned tree

Leaf	Class	Windows	T (path features)	Worst rank	S@3	S@5	S@10
$p1$	attack	732	`delta_same_id_interface`	1	✓	✓	✓
$p2$	benign	9268	`delta_same_id_interface`	1	✓	✓	✓

Table 12: SHAP rankings of the `interface_is_can9` feature for important leaf nodes

Target explanation samples	Rank / 82	mean |SHAP|
$p1$ – injection, attack (300)	73	0.0002
$p2$ – benign (300)	58	0.0023
$f3$ – UDS, attack (147)	**1**	**0.1336**

values of the 82 features of these samples, `delta_same_id_interface` ranks first. The remaining 9268 samples reaching leaf $p2$ are classified as benign, and `delta_same_id_interface` still ranks first in the averaged SHAP values of the 82 features. Therefore, TRUSTEE and SHAP agree with each other at the subset level.

Table 12 shows the mean absolute SHAP value rankings of the `interface_is_can9` feature among 82 features for leaves $p1$ and $p2$ in the pruned tree and leaf $f3$ in the adapted full tree. Numbers in parentheses represent the windows explained per leaf (capped at 300). From the table we can see that for leaves $p1$ and $p2$, which separate injection attacks such as replay and DoS from benign windows in the pruned tree, the `interface_is_can9` feature's mean absolute SHAP value ranks low among the 82 features, and makes little contribution to the prediction. However, for leaf $f3$ in the adapted full tree, the `interface_is_can9` feature has a SHAP value of 0.1336 and dominates the prediction, confirming the interface-presence shortcut identified by the surrogate tree in Figure 9.

Figure 10 shows the top-10 SHAP feature ranking of leaves $f1$ and $f3$. Leaf $f1$ has its most samples falling into the replay and DoS attack categories, and is mostly driven by the `delta_same_id_interface` feature, which has a mean absolute SHAP value of around 0.25, more than twice the value of the second feature. By contrast, leaf $f3$ mainly consists of UDS attacks, and its most important feature `interface_is_can9` has a mean absolute SHAP value of around 0.13, which dominates the other features including `delta_same_id_interface`. So the two attack leaves are using two different rules for identifying different attacks. All in all, the agreement of SHAP and TRUSTEE analysis in the two attack leaves confirms that both attack rules of the surrogate reflect genuine CNN behaviour rather than artefacts of the tree fitting.

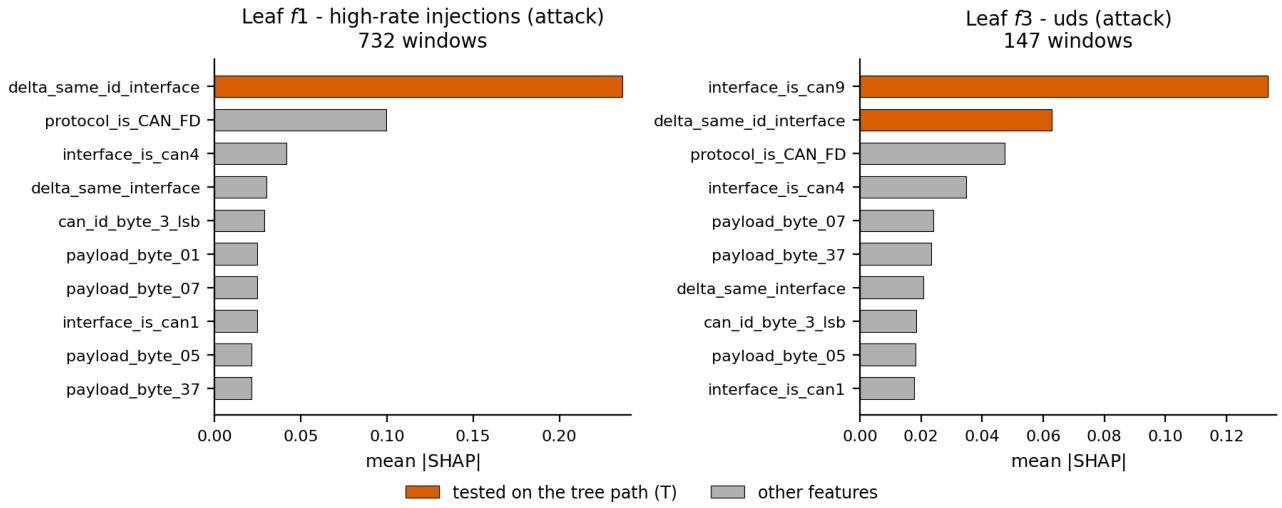


Figure 10: Top-10 features by mean absolute SHAP for two attack leaves of the surrogate tree

7 Conclusions and Future Work

This thesis investigates how SHAP and TRUSTEE can improve the interpretability of a CNN-based automotive IDS. The CNN achieves an accuracy of 0.961 and a macro F1 score of 0.908, but its performance varies strongly across attack categories. While DoS, UDS, and replay attacks are detected reliably, the model performs poorly on stealthier attacks such as spoofing and v-mode. The local SHAP explanations show that the CNN uses different signals for different attacks. The global SHAP results and the TRUSTEE surrogate tree both show that timing features, especially the minimum time difference between messages with the same identifier and interface, play a dominant role in the CNN's decisions. Except for this dominant timing rule, the adapted full tree reveals an interface-specific shortcut for UDS attacks based on the presence of CAN interface 9. The subset-level comparison with SHAP confirms that both the timing rule and the can9 rule reflect genuine CNN behaviour rather than artefacts of the surrogate tree.

Together, the XAI explanations suggest that the CNN relies strongly on a small number of features, most prominently timing patterns. This helps explain its strong performance on attacks that alter message timing or frequency and weaker performance on stealthier content-based attacks. The interface-specific shortcut suggests the presence of dataset-specific correlations that the CNN exploits, which may lead to poor generalization beyond the current dataset. Future work could perform an ablation study by removing the can9 interface feature to evaluate its effect on UDS detection; or we can introduce richer feature representation, such as features capturing the co-occurrence patterns of certain contents, which may improve model generalization across a broader range of attack categories.

References

[1] Wael Aljabri, Md Abdul Hamid, and Rayan Mosli. Enhancing real-time intrusion detection system for in-vehicle networks by employing novel feature engineering techniques and lightweight modeling. *Ad Hoc Networks*, 169:103737, 2025.

[2] Stefan Axelsson. Intrusion detection systems: A survey and taxonomy. Technical report, 04 2000.

[3] Pengzhou Cheng, Kai Xu, Simin Li, and Mu Han. Tcan-ids: intrusion detection system for internet of vehicle using temporal convolutional attention network. *Symmetry*, 14(2):310, 2022.

[4] Irina Chiscop, András Gazdag, Joost Bosman, and Gergely Biczók. Detecting message modification attacks on the can bus with temporal convolutional networks. *arXiv preprint arXiv:2106.08692*, 2021.

[5] Weiping Ding, Ibrahim Alrashdi, Hossam Hawash, and Mohamed Abdel-Basset. Deepsecdrive: An explainable deep learning framework for real-time detection of cyberattack in in-vehicle networks. *Information Sciences*, 658:120057, 2024.

[6] Gabriel Erion, Joseph D Janizek, Pascal Sturmfels, Scott M Lundberg, and Su-In Lee. Improving performance of deep learning models with axiomatic attribution priors and expected gradients. *Nature machine intelligence*, 3(7):620–631, 2021.

[7] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM computing surveys (CSUR)*, 51(5):1–42, 2018.

[8] Wouter Hellemans, Jannis Hamborg, Timm Lauser, Md Masoom Rabbani, Bart Preneel, Christoph Krauß, and Nele Mentens. Cards-controller area network and automotive ethernet realistic data set. In *2025 IEEE Annual Computer Security Applications Conference (ACSAC)*, pages 798–814. IEEE, 2025.

[9] Wouter Hellemans, Laurens Le Jeune, Md Masoom Rabbani, Bart Preneel, and Nele Mentens. Toward a real-time intrusion detection system for modern in-vehicle networks. *IEEE Transactions on Intelligent Transportation Systems*, 2025.

[10] Md Delwar Hossain, Hiroyuki Inoue, Hideya Ochiai, Doudou Fall, and Youki Kadobayashi. An effective in-vehicle can bus intrusion detection system using cnn deep learning approach. In *GLOBECOM 2020-2020 IEEE global communications conference*, pages 1–6. IEEE, 2020.

[11] Jun Huang, Mingli Zhao, Yide Zhou, and Cong-Cong Xing. In-vehicle networking: Protocols, challenges, and solutions. *IEEE Network*, 33(1):92–98, 2018.

[12] Arthur S Jacobs, Roman Beltiukov, Walter Willinger, Ronaldo A Ferreira, Arpit Gupta, and Lisandro Z Granville. Ai/ml for network security: The emperor has no clothes. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1537–1551, 2022.

[13] Seonghoon Jeong, Sangho Lee, Hwejae Lee, and Huy Kang Kim. X-canids: Signal-aware explainable intrusion detection system for controller area network-based in-vehicle network. *IEEE Transactions on Vehicular Technology*, 73(3):3230–3246, 2023.

[14] Serkan Kiranyaz, Onur Avci, Osama Abdeljaber, Turker Ince, Moncef Gabbouj, and Daniel J Inman. 1d convolutional neural networks and applications: A survey. *Mechanical systems and signal processing*, 151:107398, 2021.

[15] A Kumar and VL Thing. Evaluating the explainability of state-of-theart machine learning-based online network intrusion detection systems. *arXiv preprint arXiv: 2408.14040*, 2024.

[16] Brooke Lampe and Weizhi Meng. Intrusion detection in the automotive domain: A comprehensive review. *IEEE Communications Surveys & Tutorials*, 25(4):2356–2426, 2023.

[17] Zachary C Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, 2018.

[18] Hampus Lundberg, Nishat I Mowla, Sarder Fakhrul Abedin, Kyi Thar, Aamir Mahmood, Mikael Gidlund, and Shahid Raza. Experimental analysis of trustworthy in-vehicle intrusion detection system using explainable artificial intelligence (xai). *Ieee Access*, 10:102831–102841, 2022.

[19] Scott M Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. From local explanations to global understanding with explainable ai for trees. *Nature machine intelligence*, 2(1):56–67, 2020.

[20] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

[21] Shraddha Mane and Dattaraj Rao. Explaining network intrusion detection system using explainable ai framework. *arXiv preprint arXiv:2103.07110*, 2021.

[22] Yangyang Mei, Weihong Han, and Kaihan Lin. Intrusion detection for intelligent connected vehicles based on bidirectional temporal convolutional network. *IEEE Network*, 38(6):113–119, 2024.

[23] Cosmas Nwakanma, Love Ahakonye, Judith Njoku, Jacinta Odirichukwu, Stanley Okolie, Chinebuli Uzondu, Christiana Ndubuisi Nweke, and Dong-Seong Kim. Explainable artificial intelligence (xai) for intrusion detection and mitigation in intelligent connected vehicles: A review. *Applied Sciences*, 13:1252, 01 2023.

[24] Eva I Prakash, Avanti Shrikumar, and Anshul Kundaje. Towards more realistic simulated datasets for benchmarking deep learning models in regulatory genomics. In *Machine Learning in Computational Biology*, pages 58–77. PMLR, 2022.

[25] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition, 2010.

[26] Md Hasan Shahriar, Yang Xiao, Pablo Moriano, Wenjing Lou, and Y Thomas Hou. Canshield: Deep-learning-based intrusion detection framework for controller area networks at the signal level. *IEEE Internet of Things Journal*, 10(24):22111–22127, 2023.

[27] Hyun Min Song, Jiyoung Woo, and Huy Kang Kim. In-vehicle network intrusion detection using deep convolutional neural network. *Vehicular Communications*, 21:100198, 2020.

[28] Adrian Taylor, Nathalie Japkowicz, and Sylvain Leblanc. Frequency-based anomaly detection for the automotive can bus. In *2015 World Congress on Industrial Control Systems Security (WCICSS)*, pages 45–49. IEEE, 2015.

[29] Adrian Taylor, Sylvain Leblanc, and Nathalie Japkowicz. Anomaly detection in automobile control network data with long short-term memory networks. In *2016 IEEE international conference on data science and advanced analytics (DSAA)*, pages 130–139. IEEE, 2016.

[30] Pengcheng Wei, Bo Wang, Xiaojun Dai, Li Li, and Fangcheng He. A novel intrusion detection model for the can bus packet of in-vehicle network based on attention mechanism and autoencoder. *Digital Communications and Networks*, 9(1):14–21, 2023.

[31] Marko Wolf, André Weimerskirch, and Thomas Wollinger. State of the art: Embedding security in vehicles. *EURASIP Journal on Embedded Systems*, 2007(1):074706, 2007.

[32] Clinton Young, Joseph Zambreno, Habeeb Olufowobi, and Gedare Bloom. Survey of automotive controller area network intrusion detection systems. *IEEE Design & Test*, 36(6):48–55, 2019.

A Additional Results

Table 13: Window-level class distribution for each category and dataset split.

Category	Split	Total windows	Benign windows	Attack windows	Attack ratio (%)
benign	Train	299385	299385	0	0
benign	Validation	74841	74841	0	0
benign	Test	48402	48402	0	0
advanced	Train	75554	67977	7577	10.03%
advanced	Validation	18887	16562	2325	12.31%
advanced	Test	29781	29063	718	2.41%
DoS	Train	45372	21641	23731	52.30%
DoS	Validation	11342	7046	4296	37.88%
DoS	Test	24483	11558	12925	52.79%
ethernet	Train	95418	95418	0	0
ethernet	Validation	23852	23852	0	0
ethernet	Test	93107	93107	0	0
fuzzing	Train	87177	74838	12339	14.15%
fuzzing	Validation	21792	18607	3185	14.62%
fuzzing	Test	60355	47503	12852	21.29%
replay	Train	92711	80704	12007	12.95%
replay	Validation	23176	20040	3136	13.53%
replay	Test	49178	23562	25616	52.09%
spoofing	Train	74967	74301	666	0.89%
spoofing	Validation	18741	18575	166	0.89%
spoofing	Test	35230	34882	348	0.99%
UDS	Train	220894	211468	9426	4.27%
UDS	Validation	55223	53431	1792	3.25%
UDS	Test	153088	145569	7519	4.91%
v-mode	Train	98135	85803	12332	12.57%
v-mode	Validation	24532	21600	2932	11.95%
v-mode	Test	51047	39648	11399	22.33%

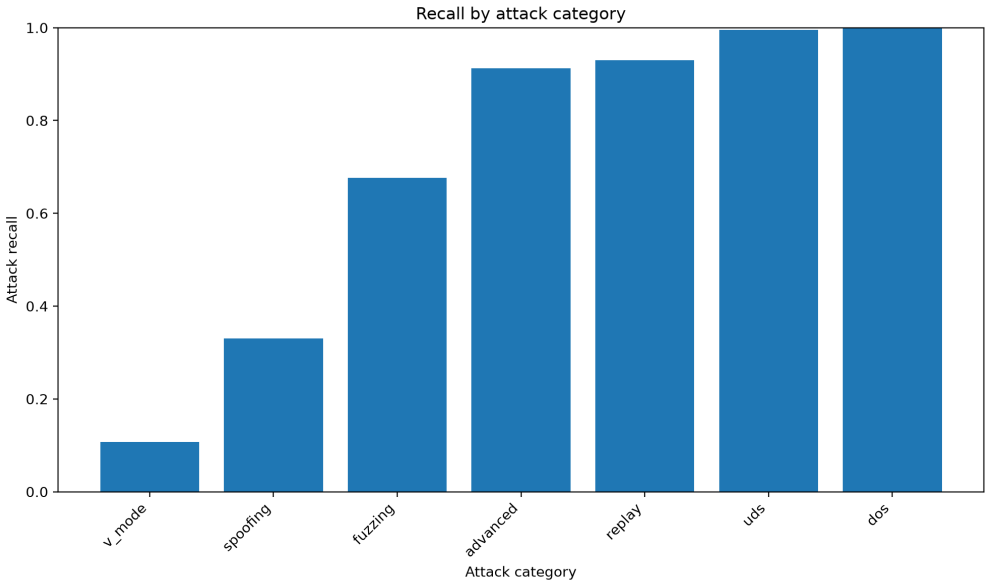


Figure 11: Recall of CNN predictions by attack category

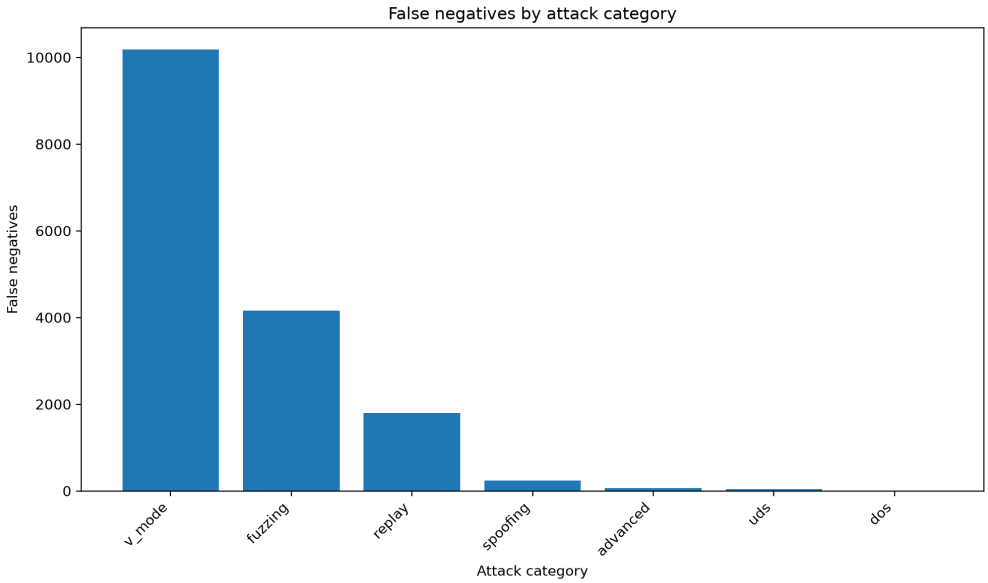


Figure 12: False negatives of CNN predictions by attack category

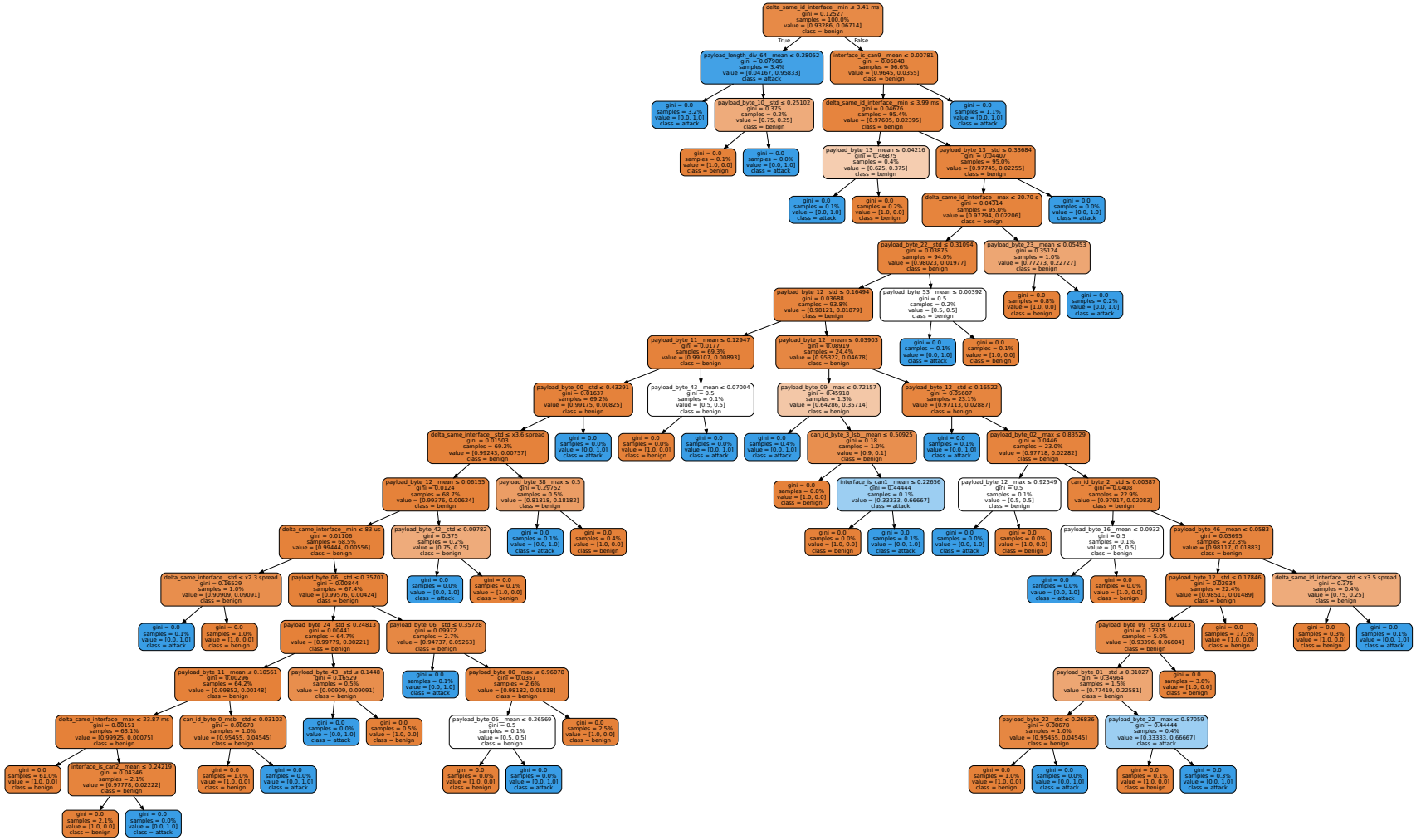


Figure 13: Full decision tree from TRUSTEE analysis