CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Explainable AI for Network Intrusion Detection in Modern In-Vehicle Networks

An explainability pipeline for deep learning based intrusion detection systems for in-vehicle networks

Master's thesis in Computer science and engineering

Wenjun TIAN, Yexiao WANG

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

Master's thesis 2026

Explainable AI for Network Intrusion Detection in Modern In-Vehicle Networks

An explainability pipeline for deep learning based intrusion detection systems for in-vehicle networks

Wenjun TIAN, Yexiao WANG

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

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

Explainable AI for Network Intrusion Detection in Modern In-Vehicle Networks
An explainability pipeline for deep learning based intrusion detection systems for in-vehicle networks
Wenjun TIAN, Yexiao WANG

Supervisor: Mohamed Hashim Changrampadi, Department of Computer Science and Engineering, Chalmers/GU; Wouter Hellemans, KU Leuven
Examiner: Magnus Almgren, Department of Computer Science and Engineering, Chalmers/GU

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

Explainable AI for Network Intrusion Detection in Modern In-Vehicle Networks
An explainability pipeline for deep learning based intrusion detection systems for in-vehicle networks
Wenjun Tian; Yexiao Wang
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Deep learning–based network intrusion detection systems (NIDS) have been widely adopted for detecting attacks in Controller Area Network (CAN) traffic due to their superior performance over traditional approaches. However, their black-box nature makes the underlying decision-making process difficult to interpret, limiting their suitability for safety-critical automotive environments. Existing studies on CAN NIDS have largely focused on detection performance and model design; limited work has examined shortcut learning through explanations or used explanations to improve the models. This thesis develops an explainable artificial intelligence (XAI) pipeline for DL-based CAN NIDS. Raw CAN fields are combined with explicit temporal and statistical features. Multilayer perceptron (MLP) classifiers are evaluated on the CAN-MIRGU and can-train-and-test datasets, and the autoencoder is evaluated on the CAN-MIRGU dataset. SHAP, LIME, Integrated Gradients, Trustee, and AE-p-values are used for behavior analysis, while Right for the Right Reasons (RRR) supervision and Gini-based attribution priors are applied to guide training. The MLP achieves high F1-scores on CAN-MIRGU, but its performance decreases substantially under unseen attacks and cross-vehicle tests in can-train-and-test. Explanations show both meaningful use of timing-related features and reliance on dataset-specific payload patterns. For the autoencoder, p-value representations improve attack clustering over raw inputs. Moreover, p-values reveal that in the CAN-MIRGU dataset, DoS and fuzzing attacks are distinguishable, whereas spoofing and replay attacks remain difficult to separate. RRR increases the F1-score for non-zero-payload DoS attacks under distribution shift from 0 to 0.61. Gini-based regularization increases explanation sparsity while maintaining high predictive performance. These results show that explanation analysis is necessary for identifying shortcuts, assessing generalization, and developing more reliable CAN NIDS.

Keywords: In-Vehicle Network Security, Network Intrusion Detection System (NIDS), Explainable AI (XAI), Explanation-Guided Learning (EGL).

Acknowledgements

We would like to express our gratitude to our supervisor and examiner, Professor Magnus Almgren, for his guidance throughout this project. His academic insights and structured feedback at various stages of our research were essential in helping us shape the final thesis and keep the study on the right track.

We are also incredibly grateful to Mohamed Hashim Changrampadi and Wouter Hellemans for their practical assistance and constant availability. Their willingness to answer our questions, discuss individual chapters, and provide continuous support during the process made a significant difference in our day-to-day work.

Finally, our thanks go to all other related staff and academic personnel who contributed to this work. Whether through administrative support or casual academic discussions, their assistance behind the scenes was highly appreciated.

Wenjun Tian & Yexiao Wang, Gothenburg, June 2026

Contents

List of Figures

List of Tables

1

Introduction

1.1 Overview

With the rapid evolution of information technology and the advent of the Internet of Things (IoT), modern automobiles are undergoing a profound paradigm shift, transitioning from isolated, purely mechanical systems into connected, software-defined entities. This digital transformation is driven by the integration of Advanced Driver Assistance Systems (ADAS), infotainment platforms, and V2X (Vehicle-to-Everything) communication modules [3]. To manage these complex functions, a contemporary vehicle typically incorporates a sophisticated internal network composed of tens to over a hundred Electronic Control Units (ECUs) [4]. These distributed controllers are mostly interconnected through the Controller Area Network (CAN), which remains a primary industry standard for most in-vehicle communication due to its low cost and resilience to electromagnetic interference [5].

However, the CAN protocol was designed to prioritize transmission efficiency, deterministic latency, and hardware reliability over cybersecurity [6]. This legacy design lacks fundamental security primitives; for instance, it does not support message authentication, allowing any compromised node to masquerade as another, nor does it provide data encryption, leaving all bus traffic visible in plaintext. Furthermore, the broadcast nature of the CAN bus means that an exploit in a non-critical component can potentially grant an adversary access to safety-critical segments. These structural vulnerabilities make modern automotive networks increasingly susceptible to sophisticated cyber-physical attacks, including Denial-of-Service (DoS), spoofing, and man-in-the-middle injections, which can directly compromise the vehicle's physical dynamics and passenger safety [7].

To protect passenger safety and ensure the functional integrity of safety-critical automotive software and hardware systems against malicious network attacks, Network Intrusion Detection Systems (NIDS) have emerged [8]. In the automotive environment, undetected attack frames can lead to catastrophic physical outcomes. Traditional detection methodologies, which primarily rely on rule-based or purely statistical architectures, while effective as a first line of defense, are insufficient on their own. These systems struggle to detect zero-day vulnerabilities and are often bypassed by complex, low-magnitude anomalies that deliberately mimic normal traffic behavior to avoid triggering simple threshold-based alarms [9].

To overcome these limitations, the automotive research community has turned to Deep Learning (DL) architectures for their ability to represent high-dimensional data. Commonly used models, including MultiLayer Perceptrons (MLPs), Convolutional Neural Networks (CNNs), and Autoencoders (AE) have demonstrated good performance [10, 11] in the realm of NIDS.

Unlike traditional algorithms, these deep neural networks excel at autonomously identifying latent, non-linear relationships within the binary stream of the CAN bus. Specifically, they can learn to recognize subtle temporal and behavioral patterns that are virtually invisible to human-defined rules and conventional statistical models. Due to the powerful data representation ability, DL-based NIDS provide significantly higher detection sensitivity, a more robust ability to generalize to novel threat patterns, and drastically reduced false-positive rates [12].

Despite their high detection performance, deep learning (DL)-based systems inherently operate as black-box models, providing limited transparency into their internal decision-making processes. In the safety-critical automotive domain, particularly in intelligent connected vehicles and autonomous driving systems, achieving high accuracy alone is insufficient to guarantee system trustworthiness and operational safety. Instead, a NIDS deployed in such environments must not only deliver accurate predictions but also provide justifiable and interpretable reasoning for its decisions. Moreover, transparent and interpretable intrusion detection is essential for post-accident investigation and digital forensics, where investigators must reconstruct the sequence of events, determine whether malicious network activity contributed to system failures, and attribute responsibility. Without explainability, DL-based NIDS outputs cannot be reliably used to support forensic analysis or safety audits, limiting their practical value in real-world automotive deployments.

If an autonomous vehicle triggers an emergency maneuver based on a NIDS alert, engineers must be able to verify whether the trigger was a genuine attack or a sensor glitch [13]. Additionally, for maintenance crews and cybersecurity analysts, knowing which specific features contributed to the classification is important for rapid root-cause analysis and hardware patching. Moreover, in terms of regulatory and ethical compliance, global standards such as ISO/SAE 21434 and the UN Regulation No. 155 emphasize cybersecurity throughout the vehicle lifecycle. These frameworks increasingly imply a need for interpretable security, where the decision-making process of automated defense systems can be audited during post-accident investigations.

As the demand for interpreting black-box deep learning models grows, Explainable Artificial Intelligence (XAI) has emerged to make algorithmic decisions more transparent, justifiable, and trustworthy [14]. To make DL-NIDS for in-vehicle networks more transparent and trustworthy, integrating XAI methods into the life cycle of building, evaluating, and improving DL-NIDS is necessary.

Building upon the need for interpretable and trustworthy DL-based NIDS, explanation-guided learning (EGL) [15] has recently emerged as a promising paradigm that goes beyond post-hoc explainability by directly incorporating explanations into the

learning process itself. Unlike traditional XAI methods that are applied after model training, EGL aims to guide model optimization using explanation signals, thereby encouraging models to learn not only accurate but also inherently interpretable representations.

1.2 Objective

The objective of the thesis project is to implement a pipeline that includes the following.

1. Build multiple DL-NIDS with different structures on modern CAN intrusion detection datasets as targets for explainability methods.

2. Apply different kinds of explainability methods to the target IDS, including feature attribution methods and surrogate models. After obtaining the results, evaluate their quality and derive the best explanation from them. The explanation result should provide information about how the model makes its decisions, as well as which parts of those decisions are reasonable or questionable.

3. Use the generated explanations to further enhance the model's explainability, align its behavior with prior knowledge, and promote desirable properties such as explanation sparsity.

The structure of the pipeline is shown in Figure 1.1.

1.3 Outline

The primary contributions of this master's thesis can be outlined as follows.

1. We design and implement a CAN-oriented feature extraction framework based on damped incremental statistics. The extractor extends the AfterImage design from Kitsune [16] to CAN traffic by computing per-ID and global temporal statistics, including weight, jitter, Hamming-distance, payload-entropy, and CAN-ID-entropy features. These statistical features are combined with raw CAN fields and used as inputs to deep-learning-based IDS models. Based on this representation, we implement and evaluate an MLP-based classifier and an autoencoder-based anomaly detector. The MLP model is trained and tested on the CAN-MIRGU dataset [2] and the can-train-and-test (CT&T) dataset [17], while the autoencoder model is evaluated on the CAN-MIRGU dataset.

2. We apply several explainability methods to the trained IDS models and compare their outputs. For the MLP models, we use SHAP, LIME, Integrated Gradients, and Trustee to analyze feature importance and local decision behavior.

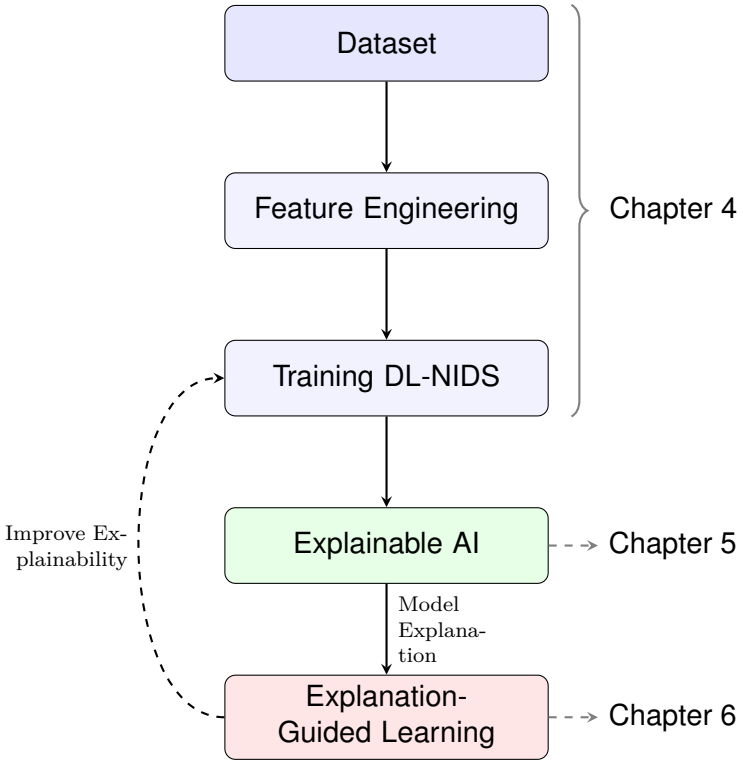


Figure 1.1: Pipeline of explainable AI for network intrusion detection in modern in-vehicle networks

For the autoencoder model, we use AE-p-values to identify anomalous feature dimensions and to study attack structure in an explainable representation space. These analyses show how the models use temporal, payload-related, and entropy-based features, and they also reveal several dataset-specific shortcuts.

3. We investigate Explanation-Guided Learning (EGL) as a way to improve models' explainability. First, Right for the Right Reasons (RRR) supervision is used to suppress reliance on undesired shortcut features, especially payload-related features in DoS detection. This improves generalization in cases where the test distribution differs from the training distribution. Second, attribution priors based on the Gini coefficient are used to encourage sparse feature attributions.

This thesis is organized as follows. Chapter 1 introduces the research background and outlines the main objectives and contributions of this thesis. Chapter 2 introduces the background required for the thesis, including the CAN protocol, common CAN attack types, intrusion detection systems, deep learning models, explainability methods, and Explanation-Guided Learning. Chapter 3 reviews related work on DL-based CAN intrusion detection, XAI methods for IDS, AE-p-values, Trustee, EGL, and XAI evaluation. Chapter 4 presents the datasets, the CAN-specific feature engineering process, and the target IDS models used in the experiments. Chapter 5 reports the application and analysis of XAI methods on the trained MLP and autoencoder models. Chapter 6 presents the EGL experiments, including RRR

supervision and attribution-prior regularization. Chapter 7 concludes the thesis, discusses ethical and sustainability considerations, and outlines directions for future work.

2

Background

2.1 Controller Area Network (CAN)

The Controller Area Network (CAN) is a robust communication bus standard designed to allow Electronic Control Units (ECUs) to communicate with each other without a central host computer [18].

A standard CAN frame consists of several distinct fields that ensure data integrity and priority management. The structure is specifically designed to be content-addressed rather than node-addressed, meaning the identifier describes the data being sent (e.g., engine RPM) rather than the destination.

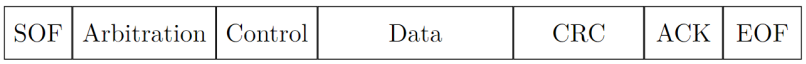


Figure 2.1: Standard CAN data frame structure; adapted from [1]

Figure 2.1 shows an overview of a CAN frame. Details of a CAN frame are as follows:

1. SOF (Start of Frame): A single bit used to mark the beginning of a message and synchronize the nodes on the bus after an idle period.

2. Arbitration Field (Identifier, or ID): This field consists of an 11-bit identifier in Standard CAN, or a 29-bit identifier in Extended CAN. It determines the message priority; in the CAN protocol, a lower numerical value corresponds to a higher priority. In the event of simultaneous transmission, the node with the lower ID wins the arbitration process without losing data.

3. Control Field (DLC): The Data Length Code (DLC) is a 4-bit field that indicates the number of bytes contained in the subsequent Data Field, typically ranging from 0 to 8 bytes.

4. Data Field: This field contains the actual payload of the message, such as sensor readings or control commands. While it is the most critical component for vehicle operations, it is also the primary target for malicious payload injections.

5. CRC (Cyclic Redundancy Check): A 15-bit checksum (plus a delimiter bit) used to detect transmission errors. If a receiving node calculates a CRC that does not match the transmitted value, it flags an error and discards the frame.

6. ACK (Acknowledgment): A 2-bit field (including a delimiter) where any node that correctly receives the message overwrites a recessive bit with a dominant bit to acknowledge a successful transmission.

7. EOF (End of Frame): A sequence consisting of 7 recessive bits that marks the formal end of the CAN message frame.

Notably, CAN frames are highly periodic because automotive ECUs transmit sensor and control data at fixed, deterministic intervals to support real-time control loops [19]. Many ECUs broadcast measurements at predefined frequencies, and the CAN protocol's continuous, broadcast-based communication model reinforces this periodic behavior [20]. This periodicity ensures a predictable and stable statistical pattern for normal CAN frames. In other words, anomalous behaviors cause changes in the statistics, motivating us to focus on extracting statistical features from the original data for attack detection.

2.2 CAN Attack Taxonomy

The broadcast nature of the Controller Area Network bus has been previously introduced. All Electronic Control Units receive unencrypted messages transmitted on the bus, which fundamentally lacks authentication mechanisms. These characteristics make the CAN bus highly vulnerable to attacks. An attacker can use a compromised ECU to perform attacks, including injecting malicious frames or disabling specific ECUs.

Attacks on the CAN bus are mainly categorized into three types, according to author [21]: injection (fabrication) attacks, suspension attacks, and masquerade (impersonation) attacks. Common injection attacks include DoS, fuzzing, spoofing, and replay attacks. These achieve their goals by introducing additional malicious frames to the bus. In contrast, suspension and masquerade attacks operate differently. Each attack type is described below.

DoS Attacks. The principle of a DoS attack on a CAN bus is consistent with its counterpart in standard networking. By maliciously sending a large volume of frames to occupy the communication channel, the attacker makes services unavailable and disrupts normal communication. As previously discussed, CAN determines message priority based on the ID. Attackers can achieve higher efficiency by sending messages with smaller IDs or IDs known to be assigned high priority. Figure 2.2a illustrates a DoS attack on a CAN bus.

Fuzzing Attacks. Similar to fuzzing in standard networking, an attacker uses a controlled node to inject many malicious frames into the bus. The IDs and payloads may be randomly generated, representing a black-box attack, or constructed based

on prior knowledge. Attackers can use this type of attack to gather information about the target, such as how it reacts to specific messages. Figure 2.2b illustrates a fuzzing attack on the bus.

Spoofing Attacks. The attacker sends malicious frames to target specific CAN IDs. In this scenario, the target ECU receives both normal messages and malicious messages sent by the attacker. With precise timing, malicious messages can be delivered immediately after normal messages. Different ECUs may react differently to such message conflicts, such as accepting only the most recent message or ignoring all of them. This technique is known as Flam delivery [22]. Figure 2.2c illustrates a spoofing attack on the CAN bus.

Replay Attacks. The attacker intercepts valid frames and resends them at a different time. Although these frames are legitimate in structure, they can cause unintended results, such as an attacker replaying a frame to unlock a vehicle. Figure 2.2d illustrates a replay attack.

Suspension Attacks. The attacker disables a compromised ECU, preventing it from sending messages according to its schedule. In this case, not only are the messages from that ECU lost, but other messages dependent on that ECU may also be corrupted. Figure 2.2e illustrates a suspension attack.

Masquerade Attacks. The attacker first disables an ECU and then uses either that same ECU or another controlled node to send malicious frames with the same ID and frequency as the original. Since the transmitted messages appear entirely legitimate, this attack is extremely difficult to detect. Detecting such attacks requires an IDS capable of analyzing payload content. Figure 2.2f illustrates the process of a masquerade attack.

2.3 Network Intrusion Detection Systems (NIDS)

An IDS is a specialized security tool designed to monitor network traffic for signs of malicious activity. Unlike a firewall, which enforces access control based on predefined rules, an IDS continuously monitors network traffic to detect anomalous or malicious behavior [23]. Several types of IDS exist, including Network-based IDS (NIDS) and Host-based IDS (HIDS) [24]. In this work, we focus on NIDS, which monitor network traffic to detect malicious or anomalous communication patterns.

A major class of NIDS is the rule-based intrusion detection system. This approach functions similarly to antivirus software, comparing observed network packets against a database of known attack patterns [23]. While highly effective at identifying established threats with minimal false positives, rule-based systems are fundamentally limited by their inability to detect zero-day attacks—threats for which no signature yet exists. Furthermore, maintaining an up-to-date rule database to include various attacks that are rapidly evolving is quite expensive.

Another common category is statistical anomaly-based detection [23]. These systems

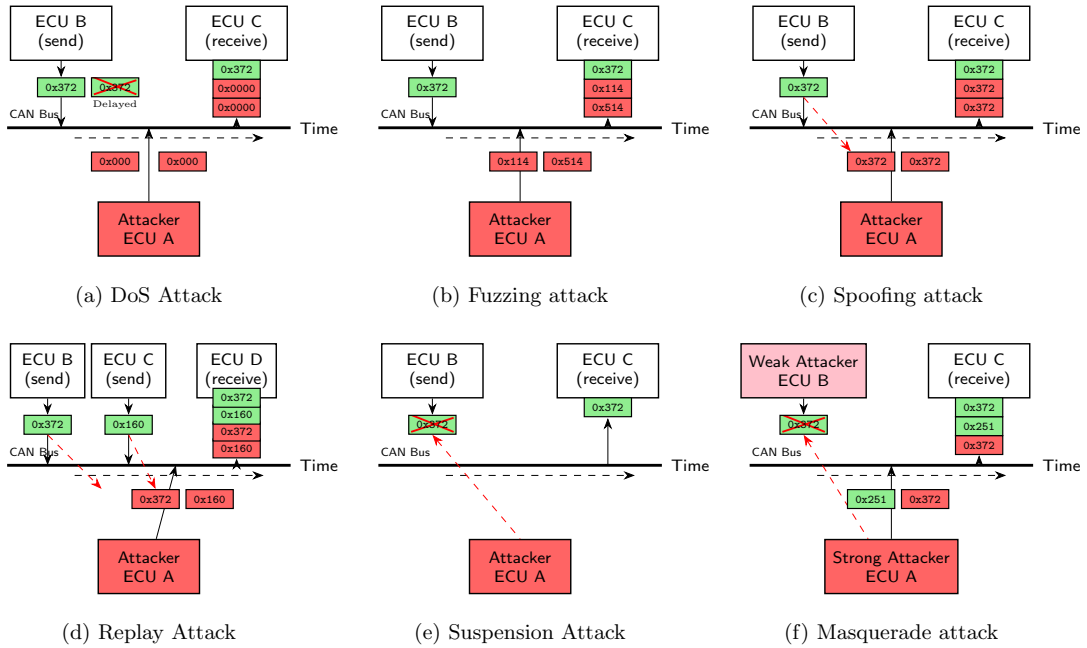


Figure 2.2: Attacks on CAN; reproduced from [2] under CC BY 4.0

establish a baseline of normal network behavior and flag any significant deviations from this norm as potential intrusions. While this allows for the discovery of novel attacks, traditional statistical methods often struggle with high false-alarm rates, since legitimate but rare network behaviors can be misinterpreted as malicious [25].

To handle the inherent complexity of the data distribution of network traffic, machine learning approaches, especially deep learning (DL) ones, were introduced [26]. They possess strong representation power, which enables them to model non-linear and high-dimensional data distribution, and therefore better identify previously unseen or evolving attack behaviors. However, these benefits come with trade-offs, including reduced explainability [27].

2.4 Deep Learning (DL)

Deep Learning (DL) is a specialized subfield of Machine Learning (ML) that is inspired by the organization of the human brain's neural networks [28]. Deep learning utilizes multilayered architectures, commonly known as Artificial Neural Networks (ANNs), to model and solve complex pattern recognition tasks. The term *deep* in deep learning refers to the stacking of multiple layers, including input, output, and numerous hidden layers that allow the model to learn hierarchical representations of data. In these architectures, each successive layer transforms the representation from the previous layer into a slightly more abstract and composite representation [29].

A standard deep learning model consists of interconnected nodes called neurons. Each connection is associated with a weight and a bias, which are adjusted during

the training process using algorithms such as backpropagation and gradient descent. By passing data through non-linear activation functions (e.g., ReLU or Sigmoid), the network can approximate any complex function [30] (this is the so-called Universal Approximation Theorem), enabling it to process unstructured data formats such as images, audio, and network traffic.

In the scope of this thesis, we focus on two types of deep learning model structures: MultiLayer Perceptron (MLP) and Autoencoder (AE).

2.4.1 Motivation

Generally speaking, there are two types of NIDS: signature-based and anomaly based [31]. The signature-based methods recognize predetermined attack patterns known as signatures during network flow surveillance. On the other hand, an anomaly detection system first creates a baseline profile of the normal network flow, and thereafter, any activity that deviates from the baseline is treated as a possible intrusion.

In the context of deep learning, anomaly detectors are often binary classifiers that only differentiate normal and abnormal messages. In the common practice, we train the anomaly detectors in an unsupervised fashion, meaning that we only expose the model to normal data, and it can recognize malicious ones by their difference from the normal ones. In this realm, autoencoders are one of the most popular structures.

Signature-based IDS also has the ability to classify between benign and attacks, but it faces several problems that are hard for it to deal with, as explained below.

The first challenge is unseen attacks. It is impossible to cover all the existing attacks for training a signature-based IDS, and new attacks are emerging all the time. However, on the contrary, the benign behavior of a CAN network is much easier to identify. Thus, we can make the model learn on benign data, and it will have the ability to spot malicious messages that are different from the normal ones.

Another challenge is class imbalance. In most CAN datasets, benign messages are dominant (over 90%), and attacks are sparse. Thus, a signature-based model considering both benign and attack data tends to focus on benign behavior and ignore the attacks [32].

In this project, we built NIDS models using both modes, with MLP and AE architectures, allowing direct comparison and analysis.

2.4.2 MultiLayer Perceptron (MLP)

MultiLayer Perceptron (MLP) is a classic feed-forward artificial neural network. Its origins can be traced back to the perceptron model proposed by Frank Rosenblatt [33] in the 1950s. It later became a foundational method in deep learning during the 1980s with the improvement of the backpropagation algorithm. Structurally, it consists of an input layer, one or more hidden layers, and an output layer.

Neurons between layers are fully connected to transmit signals, and non-linear activation functions are used to map complex input patterns and extract features. The MLP shows strong non-linear fitting capabilities when processing high-dimensional feature vectors. At the same time, its inference process is efficient, which allows it to maintain high speeds while ensuring detection accuracy. The basic structure of MLP is shown in Figure 2.3. It works in the same way that a neural network works, so further details will not be given here.

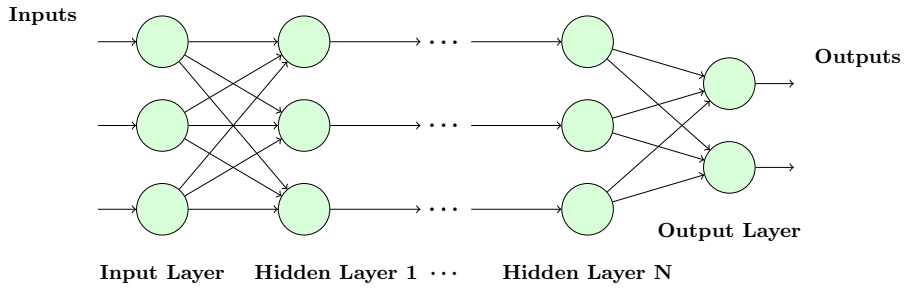


Figure 2.3: Multilayer perceptron structure with N hidden layers

2.4.3 Autoencoder (AE)

Autoencoders are composed of two main components: an encoder and a decoder [34]. The most common symmetric structure is shown in Figure 2.4. Note that the input layer and output layer may contain multiple hidden layers for better data representation ability as a DL-based model.

Given an input vector of dimension N, the encoder maps it into a latent representation of dimension K where $N > K$. The decoder then reconstructs the original N-dimensional input from this compressed latent vector. The difference between the reconstructed output and the original input is referred to as the reconstruction error. The most common form of reconstruction error is the mean squared error: $L = \|x - \hat{x}\|^2$.

When an autoencoder-based NIDS is trained exclusively on normal traffic, it learns to reproduce inputs that follow the normal patterns with minimal reconstruction error. Based on this, an input instance can be flagged as an attack if its reconstruction error exceeds a predefined threshold; otherwise, it is considered normal. Under this mechanism, an autoencoder-based NIDS is capable of identifying previously unseen attacks whose patterns deviate from the learned normal behavior.

2.5 Explainable Artificial Intelligence (XAI)

The black-box nature of deep learning presents significant reliability risks in high-security applications such as IDS. Therefore, explainability methods are necessary to inspect the model's decision logic. EXplainable Artificial Intelligence (XAI) refers to

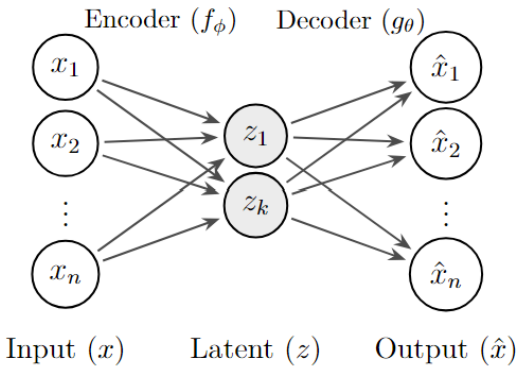


Figure 2.4: Autoencoder architecture: dimensionality reduction through a bottleneck layer

technical approaches that transform complex algorithmic decisions into understandable semantic descriptions or intuitive evidence by revealing the logic paths between internal features and model outputs [35]. XAI techniques provide explanations in various forms and perspectives. They are generally categorized based on when the explanation is generated (intrinsic vs. post-hoc) [36].

Some ML-based models inherently possess explainability to some extent, such as decision trees and Graph Neural Networks (GNNs). However, most DL-based model structures are not intrinsically explainable, so we focus on post-hoc methods in this project. Within the post-hoc methods, feature attribution and surrogate models represent two of the most influential frameworks for explaining complex models [37].

Feature attribution methods aim to quantify the contribution of each input feature to a model's prediction. Popular techniques such as SHapley Additive exPlanations (SHAP) and Integrated Gradients (IG) assign importance scores to features, enabling people to understand which variables most strongly influence the model's output. In contrast, surrogate models aim to approximate the behavior of a complex model using a simpler, more explainable model, such as a linear model, a decision tree, or a set of explicit rules. While feature attribution focuses on decomposing predictions into feature contributions, surrogate models emphasize approximating the decision boundary, offering complementary perspectives on the model's explainability.

The following section introduces the post-hoc feature attribution and surrogate model generation methods that we use in this thesis.

2.5.1 Model-agnostic feature attribution

We start with model-agnostic feature attribution methods. These methods do not depend on the internal structure of the model, such as its weights or gradients. Therefore, they are very general and can be applied to any model.

2.5.1.1 LIME

Although complex machine learning models achieve high performance, their black-box nature makes it difficult for users to trust the predictions. To address the trust issues in individual predictions and model deployment, Ribeiro et al. [38] proposed LIME (Local Interpretable Model-agnostic Explanations). This method provides interpretable explanations for individual predictions of any black-box classifier or regressor.

The core principle of LIME is that even if a black-box model has a highly non-linear global decision boundary, its behavior in the local neighborhood of a specific instance can be approximated by a simple, interpretable model. Formally, LIME formulates the explanation task as the following optimization problem [38].

$$\xi(x) = \arg\min_{g \in G} \mathcal{L}(f, g, \pi_x) + \Omega(g) \tag{2.1}$$

In the Equation 2.1, f is the black-box model, and x is the instance being explained. g is an explanation model from a class of interpretable models G (e.g., linear models or decision trees). $\Omega(g)$ is a complexity measure (e.g., the number of non-zero weights in a linear model) used to ensure the explanation is interpretable. π_x defines a proximity measure around x to represent the local neighborhood. $\mathcal{L}(f, g, \pi_x)$ measures the approximation error of g relative to f within this local region, ensuring local fidelity. In practice, LIME samples instances around x by making random perturbations to its interpretable representation. The algorithm inputs these perturbed samples into f to get predictions and weights them using π_x. Finally, LIME minimizes the objective function on this weighted dataset to learn a local sparse linear model as the explanation.

Let us take the Multilayer Perceptron (MLP) as an example. Because it contains multiple hidden layers and non-linear activation functions, an MLP model acts as a typical black-box model. To apply LIME here, the method takes two inputs: the trained MLP classifier and a specific test instance. LIME generates perturbed samples around this instance and observes the corresponding outputs of the MLP. It then outputs a list of quantitative feature contributions for that prediction. For instance, in our experiment, the input features include the CAN ID, payload, and CAN frame length. If perturbing the CAN ID causes a more significant change in the model's prediction, LIME will assign a higher contribution weight to the CAN ID. The output might show, for example, that the CAN ID has a substantial weight of $+0.75$ toward a specific predicted class (e.g., an anomaly), while the payload and CAN frame length might have smaller weights of $+0.42$ and -0.15, respectively. Using this model-agnostic local approximation, we can identify the specific feature dependencies of the MLP for any given prediction.

2.5.1.2 SHAP

SHAP (SHapley Additive exPlanations) [39] is a unified framework used to explain machine learning model predictions. It quantifies the contribution of each input

feature to the final prediction by assigning an importance value. The core theoretical foundation of this method is the Shapley value algorithm derived from cooperative game theory. Therefore, introducing the mathematical concepts of the Shapley value is necessary before detailing the SHAP framework.

The Shapley value was originally proposed by Lloyd Shapley [40] to fairly distribute the total payout among players in a cooperative game. In the context of machine learning, and specifically in the explainability of deep learning models, the prediction process is formulated as a game, where the input features act as the players. Because it satisfies a unique set of mathematical axioms for fair allocation, the Shapley value is frequently used [41] [42] to quantify the marginal contribution of individual features in black-box deep neural networks, such as intrusion detection systems.

Mathematically, the Shapley value computes the importance of a feature by averaging its marginal contributions across all possible feature subsets. Let F denote the set of all input features, $|F|$ be the total number of features, and S be a feature subset that excludes the feature i being evaluated (i.e., $S \subseteq F \setminus \{i\}$). If $f_S(x_S)$ represents the model prediction using only the features in subset S, the marginal contribution of introducing feature i to this subset is $[f_{S\cup\{i\}}(x_{S\cup\{i\}}) - f_S(x_S)]$. The Shapley value ϕ_i for feature i is calculated using the following Equation [40]:

$$\phi_i = \sum_{S\subseteq F\setminus\{i\}} \frac{|S|!(|F|-|S|-1)!}{|F|!} \Big[f_{S\cup\{i\}}(x_{S\cup\{i\}}) - f_S(x_S)\Big] \tag{2.2}$$

The weighting coefficient $\frac{|S|!(|F|-|S|-1)!}{|F|!}$ represents the probability that the subset S precedes feature i in a random permutation of all features. So, the Shapley value accounts for both the independent effect of a feature and its interactions with other features. Based on this theoretical foundation, Lundberg and Lee [39] proposed the SHAP framework.

With this principle, SHAP was introduced as a unified framework to interpret model predictions. Within the SHAP framework, the explanation is defined as an additive feature attribution method. To explain a complex original model $f(x)$, SHAP defines a theoretical explanation model g that takes the following linear form [39]:

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z'_i \tag{2.3}$$

In Equation 2.3, $z' \in \{0,1\}^M$ is a binary simplified input indicating whether a feature is present, ϕ_i is the attribution value assigned to the feature (i.e., the SHAP value), and ϕ_0 is the baseline prediction. This equation defines the target structure of the explanation, where the number of simplified features M equals the total number of input features $|F|$, and each coefficient ϕ_i is theoretically determined by the Shapley value formula in Equation 2.2. Lundberg and Lee [39] proved that among all additive feature attribution methods, there is a unique solution that satisfies three mathematical properties: 1) Local Accuracy, meaning the explanation

model's output must match the original model's output for the simplified input ($\sum \phi_i + \phi_0 = f(x)$); 2) Missingness, meaning a missing feature in the original input must have an attribution of 0 ($\phi_i = 0$); and 3) Consistency, meaning if a model changes such that a feature's marginal contribution increases or stays the same, that feature's attribution cannot decrease. This unique solution is exactly the Shapley value, providing SHAP with strict theoretical guarantees.

However, exact computation of the classic Shapley value requires iterating over an exponential number of feature subsets, resulting in a time complexity of $O(2^M)$. This becomes computationally intractable for high-dimensional data. To address this limitation, Lundberg and Lee proposed Kernel SHAP, a model-agnostic approximation method. Kernel SHAP combines the linear regression approach of local surrogate models (such as LIME) with Shapley values. It introduces a specific weighting function, the Shapley kernel, ensuring that the solution of a weighted linear least squares regression strictly recovers the Shapley values. In our experiments, data with extracted features has high dimensionality and a large volume, thus we adopt the Kernel SHAP algorithm. This allows us to provide efficient and accurate feature explainability analysis for our model while maintaining the theoretical consistency guarantees

2.5.1.3 Integrated Gradients (IG)

Mukund et al. [43] introduced Integrated Gradients. It is a local explanation method that assigns an attribution score to each input feature of a machine learning model. The method addresses the explainability of deep neural networks by providing a feature attribution approach that satisfies established axiomatic properties.

The method computes attribution by integrating the gradient of the model's prediction with respect to the input along a linear path from a baseline state to the actual input. Let $F : \mathbb{R}^n \to \mathbb{R}$ represent the deep neural network function. Let $x \in \mathbb{R}^n$ denote the actual input vector and $x' \in \mathbb{R}^n$ denote the baseline vector. A typical baseline is a zero vector.

A parameterized straight-line path between x' and x is defined using a parameter $\alpha \in [0, 1]$. A point on this path is given by the Equation 2.4 [43]:

$$\gamma(\alpha) = x' + \alpha(x - x') \tag{2.4}$$

The Integrated Gradient for the i-th feature, denoted as $IG_i(x)$, is computed by multiplying the feature difference by the integral of the gradients evaluated along this linear path. The formal definition is given by the Equation 2.5 [43]:

$$IG_i(x) = (x_i - x'_i) \int_0^1 \frac{\partial F(x' + \alpha(x - x'))}{\partial x_i} d\alpha \tag{2.5}$$

This formulation satisfies the completeness axiom. The sum of the attributions for all input features equals the difference between the model's output at the input x and

its output at the baseline x'. This relationship is expressed as the Equation 2.6 [43]:

$$\sum_{i=1}^{n} IG_i(x) = F(x) - F(x') \tag{2.6}$$

2.5.2 AE-p-values for Autoencoders

AE-p-values [44] is a novel post-hoc explanation method based on the p-values of the reconstruction errors produced by an autoencoder. This metric quantifies the abnormality of each feature, which can be used to rank the usefulness of the features when recognizing attacks using reconstruction.

2.5.2.1 Use case: model-specific feature attribution

Consider an input vector $\mathbf{x}$. For each feature dimension f_i, we denote e_i as the reconstruction error derived from the autoencoder. Meanwhile, r_i is the theoretical reconstruction error variable corresponding to x_i for normal data, i.e., the training set for the autoencoder. The p-value is then defined as the probability shown in Equation 2.7:

$$p_i = \mathbb{P}(r_i > e_i) \tag{2.7}$$

In other words, we take the empirical distribution $\hat{\mathbb{P}}$ of the reconstruction error of x_i as the reference distribution. We then compare the observed e_i to $\hat{\mathbb{P}}$ to compute the p-value of e_i with respect to $\hat{\mathbb{P}}$. A smaller p-value means that e_i has a large reconstruction error with respect to $\hat{\mathbb{P}}$, indicating that it is abnormal.

Given the p-values for each feature dimension, we can use them to rank the contribution of the features.

In our case, the dimension-wise $\hat{\mathbb{P}}$ is discrete, as the values of reconstruction error observations are discrete. Therefore, we use Equation 2.8 to obtain the p-value and produce the explanation results. It literally means the percentile of the reconstruction error e_i within the normal data flow.

$$p_i = \frac{\#\{r_i > e_i\}}{\#\{r_i\}} \tag{2.8}$$

2.5.2.2 Use case: attack clustering

Apart from feature attribution, p-values can further be utilized in clustering attacks to facilitate anomaly detection.

Anomaly detectors only distinguish between benign and malicious traffic, providing limited insight into the specific characteristics or categories of detected attacks. While such binary detection is useful for identifying abnormal behaviors, it does not

offer sufficient information for understanding attack semantics or analyzing attack similarities. To address these challenges, clustering is a commonly employed approach. By grouping anomalous samples according to their behavioral patterns or learned feature representations, clustering can help reveal latent attack structures and identify different types of intrusion activities without requiring fully labeled datasets.

Intuitively, clustering can be directly performed on the input feature vectors. However, this approach presents several challenges:

1. **Difficulty in handling heterogeneous feature distributions.** Simply normalizing all input dimensions is often insufficient, since different features may follow substantially different underlying distributions, such as long-tailed, skewed, or multi-modal distributions. As a result, the overall input space can exhibit highly complex geometric structures, making conventional clustering methods less effective and potentially leading to unstable or misleading cluster assignments.

2. **Limited explainability and anomaly awareness.** Clustering in the raw input space primarily reflects geometric proximity between samples, indicating only that certain groups of data points are close to each other in the feature space. However, such proximity does not necessarily capture how anomalous or semantically meaningful those relationships are. In the context of anomaly detection, it is important not only to identify similar samples, but also to quantify the degree of abnormality and understand why certain behaviors deviate from normal patterns. Purely input-based clustering therefore provides limited explainability for security analysis and attack understanding.

These considerations motivate us to employ p-values as representations for attack clustering models. The reasons are twofold:

1. **Unified statistical semantics across dimensions.** Each dimension of the p-value representation measures the degree of deviation from the null hypothesis, which corresponds to normal data behavior. Consequently, heterogeneous feature dimensions are transformed into a unified statistical space with consistent semantic meaning. This helps reconcile differences among features with diverse underlying distributions and reduces the complexity of clustering in the original input space.

2. **Improved Explainability for anomaly analysis.** p-values naturally encode anomaly significance, making the resulting clusters more interpretable in the context of intrusion detection. Samples with similar abnormal characteristics, feature importance patterns, or deviation profiles can therefore be grouped together more meaningfully. Compared with raw feature-space clustering, clustering in the p-value space better reflects shared anomalous behaviors and facilitates higher-level attack understanding and analysis.

2.5.3 Surrogate model: Trustee

Trustee is a post-hoc global explainability framework proposed by Jacobs et al. [45] to detect underspecification issues, such as shortcut learning or spurious correlations, in network security machine learning models. The framework converts complex black-box models into human-interpretable Decision Trees. As a model-agnostic method, Trustee satisfies four critical requirements for intrusion detection: high fidelity, low complexity, model stability, and independence from the underlying black-box structure. The implementation details of the Trustee algorithm are as follows:

Inner Loop and Dataset Augmentation. The inner loop of Trustee generates high-fidelity candidate trees through imitation learning. In each iteration, the algorithm uses uniform sub-sampling to randomly select a fraction (e.g., 30%) of the dataset to train a CART-based student model. This sub-sampling approach prevents the decision tree from overfitting to the entire dataset. Furthermore, Trustee employs a dataset augmentation mechanism: it identifies samples where the student tree's prediction differs from the black-box model, queries the black-box oracle for the correct labels, and adds these corrected samples back into the training set for the next iteration. After multiple iterations, the algorithm selects the tree with the highest fidelity, which is defined as the degree of agreement with the black-box model's predictions rather than the ground truth labels.

Top-k Pruning for Complexity Control. High-fidelity decision trees often contain hundreds of nodes, making them difficult to interpret with manual inspection. To balance fidelity and explainability, Trustee uses a specialized Top-k pruning method instead of standard cost-complexity pruning. This method assumes that the main branches covering the majority of samples represent the core decision logic of the black-box, while smaller branches are likely artifacts of overfitting. The algorithm ranks branches by the number of samples they classify and retains only the top k decision rules. This post-processing step ensures that the final tree is concise, allowing us to directly examine the network traffic features relied upon by the model.

Outer Loop and Model Stability. Because the inner loop relies on random sub-sampling, a single extraction may produce misleading results due to data distribution bias. Trustee addresses this by using an outer loop to ensure model stability. The algorithm runs the extraction process S times to generate a set of candidate trees. It then calculates the pairwise "Agreement" between these trees, which measures how often different trees make the same decision for the same input data. Finally, Trustee outputs the tree with the highest mean agreement. This mechanism effectively filters out unstable or accidental rules, ensuring that the provided global explanation is reliable for intrusion detection analysis.

We apply Trustee to generate decision trees for our black-box models, such as MLP. By inspecting these trees, we can understand the general logic of the model and identify potential shortcuts.

2.6 Explanation-Guided Learning (EGL)

Explanation-Guided Learning (EGL) [15] is a paradigm that incorporates explanation signals into the training process to improve both model performance and explainability. Instead of treating explanations purely as a post-hoc analysis tool, this approach uses them as an additional source of guidance during learning.

Two common strategies of EGL are supervision and regularization [15]. In explanation-based supervision, models are trained with explicit constraints on feature importance, encouraging them to align their predictions with human experts' prior knowledge. In contrast, explanation-based regularization introduces penalty terms into the loss function to promote desirable explanation properties, such as sparsity or stability across similar inputs. By integrating these mechanisms, explanation-guided learning helps ensure that models not only make accurate predictions but also rely on meaningful and trustworthy features.

In the following part, we introduce several EGL methods that we focus on in this project.

2.6.1 Right for the Right Reasons (RRR)

Right for the Right Reasons (RRR) [46] is a well-known aggregation-based global supervision method, which enforces models to rely on human-interpretable and task-relevant features by explicitly constraining the sensitivity of the output with respect to the input features. Instead of only supervising the predictive performance, RRR incorporates an additional regularization term that penalizes gradients of the model output with respect to input features in regions specified as irrelevant by prior knowledge. The general structure of the loss function is shown in Equation 2.9.

$$\mathcal{L}_{RRR} = \mathcal{L}_{\text{CE}} + \lambda \left\| \boldsymbol{A} \odot \nabla_{\boldsymbol{x}} \hat{y} \right\|_2^2 \tag{2.9}$$

Since we consider a multi-class classification setting, the primary task loss is the Cross-Entropy (CE) loss. The binary mask $\boldsymbol{A} \in \{0,1\}^{N \times D}$ encodes prior domain knowledge, where each entry indicates whether the corresponding feature is allowed to influence the prediction (0 indicates irrelevant or undesired features, and 1 indicates features that are permitted to contribute). By penalizing the squared norm of masked input gradients, the model is encouraged to avoid relying on spurious or non-causal correlations in the input space.

In many practical scenarios, prior knowledge is often available only for a subset of classes or a specific target concept. To address this problem, we adopt a logit-wise extension of RRR, which directly constrains the sensitivity of a specific class logit with respect to the input, regardless of the ground-truth label of the sample. This formulation allows us to apply explanation constraints for a target class across all training samples, thereby enabling class-specific explainability control.

Let $\boldsymbol{z}(\boldsymbol{x}) \in \mathbb{R}^C$ denote the logits of the model for all classes, and let $z_t(\boldsymbol{x})$ denote the logit corresponding to the target class t that we aim to regularize. The resulting logit-wise RRR objective is defined as Equation 2.10:

$$\mathcal{L}_{RRR} = \mathcal{L}_{\text{CE}} + \lambda \left\| \boldsymbol{A} \odot \nabla_{\boldsymbol{x}} z_t(\boldsymbol{x}) \right\|_2^2 \tag{2.10}$$

This formulation penalizes the input gradients of the target class logit for every training sample, regardless of its ground-truth label. As a result, the model is explicitly encouraged to suppress sensitivity to irrelevant features when forming the decision boundary for the target class, leading to more faithful and human-aligned explanations in a class-specific manner.

2.6.2 Attribution Priors

Attribution priors [47] are a class of regularization techniques that allow domain knowledge to be encoded directly into the model's explanation (attribution) rather than just its weights. Instead of treating feature importance as a post-hoc discovery, these priors supervise the feature attribution during training to ensure the model's decision logic aligns with desired properties, such as sparsity.

2.6.2.1 Definition of attribution priors

In terms of implementation, we add the regularization term of explanation sparsity to the training loss, and thus the model can learn to increase sparsity during training. Denote that $\mathcal{L}_{task}$ is the loss for attack classification, $\phi \in \mathbb{R}^F$ is the attribution (feature importance) vector, where $\phi_i(x)$ is the importance score for the i^{th} feature of input sample x, and $\Omega(\cdot)$ is the regularization term. The total loss can be represented as Equation 2.11, where λ controls the weight of regularization.

$$\mathcal{L}_{total} = \mathcal{L}_{task} + \lambda \cdot \Omega(\phi) \tag{2.11}$$

2.6.2.2 Expected Gradients (EG) as the ϕ

In the context of attribution priors, a feature attribution method called Expected Gradients (EG) is introduced [47]. It is an axiomatic feature attribution method designed to explain the predictions of deep learning models by integrating gradients over a path. It serves as a principled extension of Integrated Gradients (IG), specifically adapted to handle the challenge of choosing a single baseline. As Equation 2.12 shows, the attribution for a specific feature i in input x is defined by integrating the gradient of the model output f with respect to that feature across a path between a reference x' and the input x. The expected component arises from taking the expectation of these integrated paths over an entire reference distribution D (typically the training dataset):

$$\phi_i(x) = E_{x' \sim D} \left[(x_i - x_i') \int_{\alpha=0}^{1} \frac{\partial f(x' + \alpha(x - x'))}{\partial x_i} d\alpha \right] \tag{2.12}$$

In practice, this is efficiently approximated by sampling random references from the dataset and random points along the path for each reference.

As we can see, EG is essentially the expectation of Integrated Gradients (IG) over a distribution of baselines. While IG compares an input to a single baseline (such as a black image or zero vector), EG compares the input to the entire data distribution.

In summary, the following reasons motivate the usage of EG in attribution priors:

1. Differentiability: The primary advantage of EG is that it is based on the integration of gradients, which are inherently differentiable in frameworks like PyTorch and TensorFlow. This allows the model to compute the gradient of the attribution (a second-order derivative), enabling the loss function to propagate directly back to the model parameters.

2. Completeness: Same as IG, EG satisfies Completeness, ensuring the sum of attributions equals the difference between the current prediction and the expected prediction under a reference distribution, i.e., $\sum_{i=1}^{d} \phi_i(x) = F(x) - F(x')$, where $\phi_i(x)$ is the attribution score for the i^{th} feature of input vector x.

3. Avoidance of Reference Bias: By integrating over an entire dataset (the expectation) rather than a single arbitrary baseline (like a zero-image), EG provides a more global and representative explanation of feature importance.

2.6.2.3 Gini coefficient as the $\Omega(\cdot)$ for sparsity

Sparsity is a property of explanation in which the model relies on a few high-impact features rather than many weak or noisy ones. To quantify and enforce this, the Gini coefficient is utilized as a differentiable prior. The Gini coefficient is a statistical measure of distribution inequality, originally developed in economics. In the context of feature importance, a higher Gini coefficient indicates that the input's contribution is concentrated in a few key features, while a lower value suggests a more uniform or flat distribution.

In Equation 2.13, for a vector of non-negative feature importance values ϕ sorted in non-decreasing order ($\phi_1 \leq \phi_2 \leq \cdots \leq \phi_n$), the Gini coefficient is defined as:

$$\Omega(\phi) = G = \frac{\sum_{i=1}^{n}(2i - n - 1)\phi_i}{n \sum_{i=1}^{n} \phi_i} \tag{2.13}$$

The coefficient ranges from 0 to 1, where $G = 0$ represents perfect equality (all features are equally important) and $G \approx 1$ represents maximal inequality (a single feature accounts for all importance). Clearly for better sparsity, the G value should be close to 1.

3

Related Work

In this chapter, we review existing research in two primary areas: deep learning-based intrusion detection for CAN buses and the application of XAI in cybersecurity.

3.1 DL-based CAN IDS

Amato et al. [48] studied deep learning for CAN-bus attack detection using neural networks and multilayer perceptrons. Their method represents each CAN message with the eight data bytes and evaluates binary and multi-class detection on the HCRL Car-Hacking dataset [49], including DoS, fuzzing, gear spoofing, and RPM spoofing attacks. The MLP models performed better than the basic neural network, with the best multi-class setting reaching a weighted precision of 0.974 and a weighted recall of 0.965.

Khan et al. [50] and Meng et al. [51] both evaluated IDS models on OTIDS dataset [52]. Khan et al. [50] proposed DivaCAN, an ensemble IDS that combines deep neural networks, multilayer perceptrons, LightGBM, extra trees, random forest, bagging, and k-nearest neighbors. On the public CAN intrusion dataset OTIDS, DivaCAN reported 0.949 precision, 0.950 recall, and a 0.950 F1-score for DoS, fuzzy, impersonation, and normal traffic. Meng et al. [51] proposed GB-IDS, which converts CAN ID sequences into graphs and trains a variational autoencoder only on attack-free samples. On the OTIDS dataset, GB-IDS reported 0.997 accuracy for DoS, 0.997 for fuzzing, and 0.996 for impersonation, but its graph-window design cannot identify the exact injected message inside an anomalous window.

The studies above mainly focus on model design and model architecture. They use complex architectures with the expectation of better detection performance. In contrast to our work, most of their evaluations are based on relatively simple datasets, such as the Car-Hacking dataset and OTIDS. The attack patterns in these datasets are simple and limited. To address this limitation, our thesis compares several CAN intrusion detection datasets in terms of their data properties and explains why these datasets are not used as the main evaluation datasets in our experiments.

Recent methods have also adopted graph-based representations to model relations between CAN messages. Zhou et al. [53] proposed CGTS, an unsupervised CAN intrusion detection method based on CAN message graphs. The method constructs

message graphs from CAN sequences, uses a Graph Transformer to extract structural information, and applies Support Vector Data Description to model normal traffic under class imbalance. Experiments were reported on the Car-Hacking dataset and the can-train-and-test (CT&T) dataset [17]. CGTS achieved an average accuracy above 0.990 and an average F1-score around 0.993. For CT&T dataset, however, the evaluation focused on selected attack types and did not cover all test subsets of the full dataset.

Hasan et al. [54] extended this line of work with CAN-GraphiT, which combines graph features and temporal features with a transformer attention network. The model was evaluated on several CAN datasets, including the HCRL Car-Hacking dataset, ROAD [22], and CAN-MIRGU [2]. CAN-GraphiT reported 0.985 overall accuracy and per-attack accuracies above 0.960 for DoS, fuzzy, gear, RPM, suspension, fabrication, and masquerade attacks. Although the work includes CAN-MIRGU, the evaluation uses the suspension attack from that dataset and does not cover the other attacks in CAN-MIRGU dataset.

Frenken et al. [55] further used graph learning in KD-GAT, where CAN traffic is represented as sliding-window graphs and classified with a Graph Attention Transformer (GAT) under a knowledge distillation framework. A large teacher GAT is trained first, and a compact student GAT, with 6.32% of the teacher model size, is trained using soft and hard supervision. The method was evaluated on Car-Hacking dataset, Survival Analysis dataset [56], and CT&T dataset. The student model reached 0.999 accuracy on Car-Hacking and 0.993 accuracy on Survival Analysis. On CT&T dataset, however, performance dropped on unseen test sets; the paper mainly attributed this to severe class imbalance, while the underlying causes of this reduced transfer across subsets were not examined in detail.

These studies use more recent datasets, such as CAN-MIRGU and CT&T, but their evaluations remain partial because they focus on selected attack types or subsets. This may overlook generalization limits across the complete dataset. Unlike these evaluations, our thesis evaluates CAN-MIRGU and CT&T more comprehensively and uses explainability methods to study why model performance decreases on some test subsets.

3.2 Application of XAI in CAN IDS

Several studies have applied XAI methods to CAN intrusion detection models.

Kibriya et al. [57] proposed a lightweight explainable IDS for autonomous vehicle CAN traffic. Their architecture combines convolutional layers, LSTM layers, GRU layers, and a residual connection to capture spatial and temporal patterns in CAN messages. The model was evaluated on the HCRL Car-Hacking dataset for DoS, fuzzy, RPM spoofing, and gear spoofing attacks, reporting about 0.999 accuracy and F1-scores close to 1.0. For explainability, they applied LIME to the trained model and used local explanations to show which CAN ID and payload fields contributed to attack and normal predictions.

Wickramasinghe et al. [58] addressed explainability from an anomaly detection perspective with RX-ADS, an interpretable system for electric-vehicle CAN data. RX-ADS extracts window-based frequency and payload features from CAN traffic and trains a ResNet autoencoder on normal data only. Detected anomalies are then explained through an adversarial method: the system finds the minimum feature changes needed to move an anomalous window toward normal behavior and uses these changes as explanations. The method was evaluated on OTIDS and Car-Hacking for DoS and fuzzy attacks, with performance comparable to HIDS on OTIDS and better than HIDS, GIDS, OCSVM, and LOF on Car-Hacking. The explanations also matched domain knowledge: high-priority ID 0000 is associated with DoS, while many unseen random IDs are associated with fuzzy attacks.

Sajib et al. [59] applied a different explanation method in CANGuard, a CNN-GRU-attention architecture for intrusion detection in in-vehicle CAN networks. The model uses CNN layers for local payload patterns, bidirectional GRU layers for temporal dependencies, and an attention mechanism for feature weighting. It was trained and evaluated on CICIoV2024, which includes benign traffic, DoS, and spoofing attacks against gas, RPM, speed, and steering-wheel signals. CANGuard reported 0.998 accuracy, precision, recall, and F1-score. The authors applied SHAP to explain the model at the CAN-payload level and reported that several data bytes, especially DATA 4 and DATA 5, had the largest effect on the model's predictions.

The studies above train IDS models on CAN intrusion detection datasets and apply explainability methods to interpret the models. However, the datasets used in these works are still relatively simple. In addition, explainability is usually treated as an auxiliary analysis rather than the main research focus. Most of these studies use only one explainability method. To address this gap, our thesis uses several explainability methods to interpret the internal logic of the model and compares the explainability methods themselves.

Overall, these other studies place most of their emphasis on model architecture. These studies design complex models that can implicitly extract temporal features from the data. However, explainability methods do not always show directly which features the model depends on. The approach taken by us in this thesis first designs and extracts explicit features, decouples related information into separate features, and then trains a simple MLP model. This design makes the explanation results easier to interpret because the features used by the model have clear meanings.

Furthermore, to the best of our knowledge, no prior work has tried to improve the explainability of DL-based CAN intrusion detection systems using EGL methods, including explanation regularization and supervision.

4

Framework for CAN IDS

In this chapter, we outline our overall design approach, detailing the core rationale behind combining explicitly extracted temporal features with lightweight machine learning models. Next, we review the current state of datasets within the specific field of in-vehicle network intrusion detection, and explain our reasons for selecting datasets. We then describe our feature engineering process, explaining how we extract time-series dependencies from raw CAN traffic. Following this, we present the training setup and evaluation results of our MLP models. Finally, we detail the implementation and performance of our autoencoders. These components are very important parts of our entire thesis project, serving as the fundamental basis for all of the experiments that we will conduct.

4.1 Overall Design Approach

This chapter presents the overall design approach of our proposed framework, detailing the core design rationale. We design a statistical feature extractor based on damped incremental statistics and define a set of time-related features. These features are combined with the raw CAN data and used as the input to ML models.

In Chapter 3, we review several related studies in this field. In general, recent work has focused on model innovation. Many studies improve detection performance by changing the model principle or architecture, with the aim of better distinguishing attack frames on the CAN bus.

However, this direction also has limitations. As models become more complex, both training cost and runtime cost increase. This may be acceptable in some domains, but it is a practical concern for in-vehicle intrusion detection. On one hand, such systems have strict real-time requirements: the IDS must make a decision and trigger a response within a very short time. A complex model may require more time for inference. On the other hand, the computing resources and memory available in in-vehicle systems are often limited. Therefore, models with lower overhead are generally preferable. One possible solution is to combine the outputs of several lightweight machine-learning models for the final decision.

Another issue is related to the temporal structure of CAN traffic. Message sequences on the CAN bus often show strong time-series dependencies, which can

also be described as context-dependent properties [20]. To achieve good detection performance, the model must be able to capture sequence-related or context-related features. GRU-based and LSTM-based models are common examples. However, these models extract temporal features implicitly, and the extracted representations are often difficult to interpret. As a result, when explainability methods are applied to study their internal behavior, the explanations are not always easy to relate to concrete CAN traffic properties.

Therefore, we adopt a more explicit design. Following common practice in other machine-learning-based traffic analysis, we first manually design and extract features, such as the number of messages within a time window or the sum of payload Hamming distances in that window. These features are described in detail in the next section. We then provide both the extracted features and the raw data to the model as input features with equal status, allowing the model to learn which features are useful for prediction. Based on this design, we use an MLP as the main target model in our experiments. The MLP is simple and close to a basic deep neural network. It also treats all input dimensions uniformly, so the extracted features and the raw CAN fields have the same opportunity to influence the decision. This design has two advantages. First, it allows us to use a simpler model. Second, it decouples correlated temporal properties into separate input features. In the later explainability experiments, this makes it possible to observe directly which features the model relies on.

Based on these goals, we need to select suitable datasets for the experiments and design the feature engineering process. The following sections describe these two parts.

4.2 Datasets

In this section, we are going to provide a detailed introduction to the information related to the datasets that are being used throughout this entire project. This includes a basic introduction to various datasets that are frequently utilized in this research field at the present time, including their specific characteristics as well as their advantages and disadvantages. By conducting a comparison of these different datasets, we can explain the reasons why we have chosen to use some of them instead of others.

A large number of CAN intrusion detection datasets are available in this research field. However, selecting a highly suitable dataset is challenging for several reasons. First, the sheer volume makes it impractical to perform a comparative analysis of every single one. Second, while many existing papers have attempted to compare these datasets, their analyses often lack objectivity; most of these comparisons tend to highlight the superiority of the authors' newly proposed datasets over existing ones. Consequently, relying on existing literature does not provide a clear, unbiased picture of dataset quality.

Despite this situation, we can still get a very good look at the current state of this

field by looking at the analysis provided in papers such as Verma et al. [22] and Lampe et al. [17]. From these sources, we can see that there are very few high-quality datasets available, and the vast majority of the existing datasets suffer from a variety of different problems. The primary limitations include insufficient attack-free data, variation (both in attacks and types of vehicles), and labeling, as well as a lack of precision and modernity [17].

The low quality of these available datasets has a negative impact on the work we are conducting, and it is difficult for us to find ways to reduce or mitigate these problems in an effective way. We fully understand why the current situation is like this. It is definitely not easy to collect attack data from a car that is actually being driven on the road, because doing so could create dangerous situations for the vehicle itself and for the driver. However, many of these issues are very serious problems for our research. For example, many datasets, such as the one in Verma et al. [22], do not have any labels. Because our time and our capabilities are limited, we are not able to add missing information or make corrections to these flawed datasets, such as by performing the labeling work ourselves. On the other hand, some datasets, such as the Car-Hacking Dataset [49], are considered to be too simple according to the analysis by Rajapaksha et al. [60]. We have a very strong need for a high-quality dataset because the work we will do later involves these datasets, especially the underlying logic of the attacks. Using a dataset that does not meet the necessary standards would have a very negative impact on our work. Table 4.1 compares the main properties of commonly used datasets in this field.

Table 4.1: Comparison of different CAN datasets

Dataset	**L**	**T**	**M**	**D**	**F**	**S**	**R**	**Q**	**P**
CAN-MIRGU [2]	√	√	√	√	√	√	√	√	√
can-train-and-test [17]	√	√	√	√	√	√	×	×	×
CICIOV 2024 [61]	√	×	√	√	×	√	×	×	×
Car Hacking [49]	√	√	×	√	√	√	×	×	×
Survival Analysis [56]	√	√	×	√	√	√	×	×	×
OTIDS [52]	×	√	×	√	√	√	×	×	×

Note: √ indicates the feature is supported, while × indicates it is not available.
Header Abbreviations: **L**: Label, **T**: Timestamp, **M**: Multiple attack modes, **D**: DoS attack, **F**: Fuzzing attack, **S**: Spoofing attack, **R**: Replay attack, **Q**: Masquerade attack, **P**: Suspension attack.

It can be observed that many existing studies in this field, including those discussed in Chapter 3, are based on earlier and relatively simple datasets, such as the Car-Hacking dataset and CICIOV2024. These datasets are classical benchmarks in CAN intrusion detection, but most of them have clear limitations. One common issue is the use of very limited attack patterns. For example, in the Car-Hacking dataset, except for fuzzing attacks, the remaining three attack types each target only a single CAN ID, use only one payload pattern, and are injected at a fixed frequency.

CICIOV2024 does not provide timestamps. In this dataset, the real time interval between two adjacent messages may be 0.0001 s or 0.2 s, but this information is not available, which removes useful temporal information. In contrast, CAN-MIRGU and can-train-and-test (CT&T) contain more diverse attack types and attack patterns. Since they were released more recently, only limited work has used them so far. Therefore, we select CAN-MIRGU and CT&T as the main datasets in this thesis. This allows us to conduct a complete evaluation on these datasets. Their more complex attack settings also provide richer behavior for the subsequent XAI experiments and support a more detailed analysis.

CAN-MIRGU dataset [2]. This dataset was published by Sampath et al. in 2024. We have chosen this dataset for several reasons, which are described in detail below.

The data in this set was collected from commercial vehicles that were actually being driven in real-world conditions on public roads. In addition, the paper also provides a record of the actual physical effects that each attack had on the vehicle during its operation. It is very important for us to have this level of authenticity and real-world data to serve as a strong support for the experiments we are conducting.

This dataset comes with a complete and thorough set of labels, and the documentation provided for it is very clear. Furthermore, the dataset includes metadata related to the attacks, which provides information about how the attacks were actually performed. All of these factors are very helpful for us because they allow us to gain a much deeper understanding of the nature of the data and make it much easier for us to carry out our experiments.

The types of attacks that are included in this dataset are very diverse, including Denial of Service (DoS), fuzzing, replay, and spoofing, as well as masquerade and suspension attacks, and there are different patterns for the same kind of attack. This is a specific characteristic that many other datasets in this field do not possess. Having this variety is beneficial for our work because it allows us to observe the behavior of our models in more detail. This, in turn, helps us to identify potential "shortcuts" and provides a good foundation for the experiments we will perform in next stages.

can-train-and-test dataset [17]. This dataset was published by Lampe et al. in 2023. There are several reasons why we decided to use this dataset in our research.

First, this dataset was collected from actual vehicles that were being driven in real-world environments and it is properly labeled. Furthermore, it includes a large and diverse set of different attack categories and specific attack patterns. In these aspects, this dataset is quite similar to the CAN-MIRGU dataset that we described previously.

More importantly, this dataset contains a collection of information about the same types of attacks as they occurred on different models of vehicles. Because the authors organized the data in a very systematic way, we have been able to obtain four

different test subsets. When these test subsets are compared with the data used in the training set, we can see that they are specifically designed to include data from either the same or different vehicles, and they contain either the same or different types of attacks. This structure is very useful because it allows us to test the generalization capabilities of our models. Specifically, we can evaluate whether the models are still able to correctly identify an attack when they are faced with data from a vehicle or attacks they have not seen before. This approach is very well-suited to the core goals of our thesis. In situations where a model fails to perform correctly when it is given unknown data, we can use explainability methods to investigate and study the underlying reasons why this failure occurred.

These different datasets are quite similar to one another so we are able to replace the dataset we are using with another one whenever there is a need to do so, or we can choose to compare these datasets against each other to better understand our results. In actual practice, all of these datasets possess the same basic structure. This structure consists of the CAN ID, a payload that contains 8 bytes of data, and a specific timestamp for each message. With only a small amount of data processing and formatting work, these datasets can be modified so that they can serve as a standardized and unified input for our models.

This approach successfully addresses two separate but important challenges. On one hand, it allows us to interchange and replace the different datasets at any time whenever it is necessary to do so. This is very helpful because it prevents the specific problems of a single dataset from having a negative impact on our experiments. On the other hand, even though these datasets were collected from several different models of vehicles, and their network traffic characteristics might be very different from one another, the core logic and the actual implementation methods of the various attacks remain very similar across all of them. By using these datasets, we are able to conduct a horizontal comparison between them. This allows us to identify any potential problems or issues that might exist in our research.

4.3 Feature Engineering

As discussed above, messages on the CAN bus often show strong time-series dependencies, which can also be described as context-dependent properties. Therefore, we need to extract time-related features explicitly. The AfterImage feature extraction framework from the Kitsune project [16] is well suited for this purpose. With minor modifications, it can be adapted to the feature extraction requirements of this thesis.

Kitsune. Kitsune, introduced by Ymirsky et al. [16], is an efficient network traffic intrusion detection system using artificial neural networks. It consists of a group of lightweight autoencoders that are designed to rebuild the traffic. AfterImage is a component of the Kitsune system.

We modified this feature extractor to adapt it to the CAN message structure. The original design of the Kitsune framework was created to monitor network communication protocols. These types of protocols usually involve two parties that are

communicating with each other, and therefore AfterImage feature extractor uses a unique identifier for each specific conversation to calculate statistical features separately. However, the Controller Area Network (CAN) operates as a bus-based communication system. In this system, all messages are sent out as a broadcast, and there is no specific target address included in the message. We need to make adjustments to handle this difference. To be more specific, we have modified the framework so that AfterImage treats each individual CAN ID as if it were a source IP address. This allows the system to calculate time-based features for each CAN ID separately.

At the same time, we sometimes also need to pay attention to the global features of the entire network. For example, in the case of a fuzzing attack that uses random CAN IDs, the system would likely fail to identify the attack if it only tracked information for each specific ID. To solve this problem, we have added an extra statistical channel. This channel records data for every single message regardless of what its CAN ID is, which allows us to obtain a set of global features for the network.

This feature extractor provides the ability to set the size of the time windows to any value that is required. It is then able to calculate a variety of different statistical features across different channels within those specific time windows. This functionality is very well-suited to the requirements that we have for extracting features from the CAN datasets. In fact, this is the primary and core reason why we made the decision to make modifications to this feature extractor and use it for our experiments.

To provide an example of how this process works, if we choose to set up three different time windows, such as 0.1 seconds, 1.0 second, and 10.0 seconds, the system will operate in the following manner. For any single CAN message, we will calculate and track several different statistical features for all of the previous messages that have the same CAN ID as the current one. We look at the messages that appeared within the time periods of 0.1 seconds, 1.0 second, and 10.0 seconds prior to the timestamp of the current message. These calculated features might include, for instance, the entropy of the data payload within those messages, or the level of density and frequency of the messages during those intervals. Once we have successfully obtained these various statistical features, we attach them to the end of the current message so that they can serve as its input features. In this way, we have effectively added the time-based statistical information onto the message.

We selected some features that AfterImage will extract from CAN datasets. First, it is important to explain that AfterImage feature extractor is able to calculate both the mean and the standard deviation (std). Almost every feature that we calculate in our work includes both of these different statistics. Therefore, we will not repeat this point again in the sections that follow. We have kept some of the features that were already implemented in the original Kitsune framework, such as the weight and the jitter.

In addition to these features, we have also designed and implemented several new fea-

tures that are intended to better handle the unique characteristics of CAN datasets. For instance, we added features like the Hamming distance and the payload entropy, which allow the model to examine the specific patterns within the payload of the CAN data. Many of the existing CAN intrusion detection systems choose to ignore the payload entirely and only focus on the CAN ID and the timing features. This is often done because the payload data can vary a great deal between different models of vehicles. However, we have made the decision to include the payload itself and other payload-related features as inputs for our models. One reason for doing this is to check for potential shortcuts that the models might be taking. This refers to the possibility that a model might make its decisions by identifying features that are specific to one dataset but cannot be generalized to other systems. This is a very important point that our research aims to investigate. Table 4.2 lists all of the specific features that we have selected for our research, along with an explanation for each of them.

Table 4.2: Features from AfterImage feature extractor

Feature Name	**Description**
weight	The current frequency (activity intensity) of a specific CAN ID.
jitter	The average time interval and the timing fluctuation (stability) between consecutive messages of a specific CAN ID.
hamming distance	The average number and the variation of bit flips (Hamming distance) between consecutive payloads of a specific CAN ID.
payload entropy	The average complexity (Shannon entropy) and the variability of the payload data for a specific CAN ID.
global weight	The total frequency and overall traffic volume of all messages across the entire CAN bus.
global jitter	The average interval and the timing stability of all consecutive messages sent on the bus, regardless of ID.
global CAN ID entropy	The average diversity and the distribution stability of different CAN IDs appearing on the bus.

It is important to mention that the feature selection part of our work, much like the datasets themselves, remains a part of the process that is very flexible and can be easily changed or replaced. This flexibility applies both to the features themselves and to the use of different values for the λ (lambda) parameter. In particular, we have discovered during our experiments that it is often best to extract different combinations of lambda values for different datasets. If this is not done, there is a risk that many of the features might become completely ineffective for a specific dataset. In a practical, real-world application, a design of this type would certainly need to take a wide range of different time scales into account at the same time. This is necessary so that the system can identify and detect attacks that are carried out at different speeds, whether they last for only a few microseconds or continue for several tens of seconds. The flexibility that we have mentioned regarding both

the datasets and the features provides a very solid foundation for us as we continue to improve our models and the features themselves.

4.4 CAN IDS: MLP training and results

We build an MLP model with three hidden layers of sizes 256, 128, and 64. ReLU activations are used, and dropout with a rate of 0.2 is applied to reduce overfitting. After Z-score normalization, the same input representation is used for different classification settings: a multi-class task on CAN-MIRGU, where class weights are used to address imbalance, and a binary task on CT&T, where the decision threshold is calibrated by grid search. The model is trained with the Adam optimizer and cross-entropy loss. Early stopping and model checkpoints are used to retain the best validation model. The final model is evaluated on the test set using accuracy, F1-score, and confusion matrices.

In terms of the results, we achieve results that can be considered acceptable for the MIRGU dataset. Table 4.3 shows the result we got from CAN-MIRGU dataset.

Table 4.3: Results on CAN-MIRGU dataset

Category	Accuracy	Precision	Recall	F1-score	Support
Benign	0.9997	1.0000	0.9997	0.9999	2214370
DoS	0.9999	1.0000	0.9983	0.9991	1740
Fuzzing	0.9998	0.8371	1.0000	0.9113	1639
Spoofing	0.9996	0.9751	0.9863	0.9807	22192
Replay	0.9999	0.9942	0.9999	0.9970	8589

However, the situation is different for the CT&T dataset. Although the authors of the original paper mentioned that they achieve a very high level of performance with an MLP, they do not provide parameters in their public report. Based on our own experiments, we are not able to reach a good result even after trying for a long period of time. The performance of the model is quite poor for a large portion of the data. Table 4.4 shows the detailed results we get from the CT&T dataset. In Table 4.4, four rows refer to different test sets, which in order are test set 01 on known vehicle and with known attacks, test set 02 on unknown vehicle and with known attacks, test set 03 on known vehicle and with unknown attacks and test set 04 on unknown vehicle and with unknown attacks.

The results show that the model achieves high accuracy across all test sets, as the vast majority of the traffic in the dataset is normal. However, the model shows poor generalization when facing unknown vehicles, with F1 scores approaching 0. Even for attacks involving known vehicles, the F1 scores are not high because the model fails to learn the features of certain attacks. Fig 4.1 shows the confusion matrix. As indicated by the confusion matrix, the low F1 scores for test set 02 and test set 04 are mainly attributed to a high false negative rate. When the frames are collected from unfamiliar vehicles, the model only predicts all frames as normal. A

more detailed analysis and explanation of these results will be provided in the next chapters of this thesis.

While this might be due to issues in our own implementation, we have made the decision not to spend more time trying to get a better result for this specific dataset. The reason is that the final goal in this thesis is not necessarily to build an IDS system that is perfect and ready for actual use. Instead, we want to use these experiments as a way to gain insights and a deeper understanding of how it works. Even an MLP model that shows poor performance can still be a very good object to study for our research.

Table 4.4: Results on CT&T dataset

Test Set	Average Accuracy	Average Precision	Average Recall	Average F1 Score
Test 01 (K,K)	0.9991	0.9717	0.6582	0.7164
Test 02 (U,K)	0.9809	0.0000	0.0000	0.0000
Test 03 (K,U)	0.9985	0.5418	0.2532	0.2756
Test 04 (U,U)	0.9983	0.0000	0.0000	0.0000

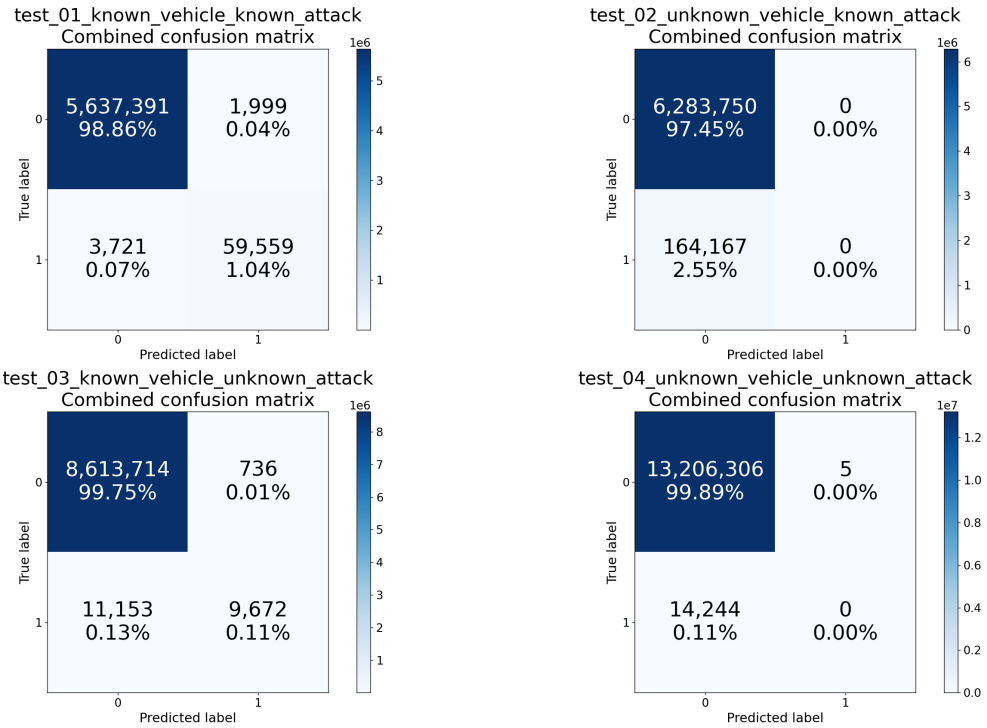


Figure 4.1: Confusion matrix of the MLP model on the CT&T test set

4.5 Autoencoders

In addition to the MLP models introduced as our primary supervised learning targets for general XAI methods, we also trained autoencoders on the CAN-MIRGU dataset as an unsupervised complementary target for the AE-p-values analysis.

4.5.1 Model training

Feature normalization. Feature normalization is essential for autoencoder training, as the reconstruction loss is scale-sensitive. In typical formulations (e.g., the aforementioned mean squared error), each feature contributes proportionally to its numerical magnitude. As a result, features with larger value ranges can dominate the loss function, while features with smaller scales have a minor impact on the loss function. This imbalance leads the model to prioritize reconstructing high-magnitude features, resulting in biased representation learning. Note that we do not perform dimensionality reduction here, since each of the input dimensions (features) has its own explainability semantics.

Model structure. The basic structure we used is the symmetric MLP-based structure shown in Figure 2.4. We use ReLU for activation, which accelerates convergence by mitigating the vanishing gradient problem, enhances computational efficiency through simple thresholding, and promotes representation sparsity, which helps the model learn more robust features by deactivating less relevant neurons.

The key hyperparameter to tune is the size of the bottleneck. The bottleneck forces the network to ignore noise and capture only the latent representation of the input data. If it is too wide, the model simply memorizes the input via an identity function, causing overfitting; if it is too narrow, the model is unable to represent the inherent complexity of the input space, leading to underfitting. The optimal size of the bottleneck is such that the model can capture the manifold of the input data, while being sensitive enough to anomalies.

Loss function. We use the mean squared error (MSE) as the optimization objective. The mathematical rationale of MSE is the assumption that reconstruction errors are normally distributed. That is, we assume that the model has a reconstructed output $\hat{x}$ for an input x. The input can be considered as: $x = \hat{x} + \epsilon$, where ϵ is the reconstruction error, and commonly we assume that $\epsilon \sim N(0, \sigma^2)$. Thus, the probability density function (likelihood function) of observing a specific sample x is defined as $P(x|\hat{x}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\hat{x})^2}{2\sigma^2}}$, i.e., $-\log P(x|\hat{x}) = \frac{(x-\hat{x})^2}{2\sigma^2} + \text{const}$. Therefore, minimizing $(x-\hat{x})^2$ is equivalent to maximizing the likelihood of input x. For NIDS, normal traffic features are assumed to be stable; thus, minimizing MSE equates to maximizing the likelihood of normal data. Moreover, MSE magnifies significant deviations (anomalies) via the squaring operation, and it provides smooth gradients to descend for optimizers like Adam, leading to stable model convergence.

Training. Generally, we train the model on the dataset for multiple epochs until its loss on the hold-out validation set is minimized, or the iteration limit is reached.

To minimize the loss, we use the optimizer Adam, which is based on the gradient descent mechanism.

Testing. We need the model to have a binary output (benign/malicious) instead of a numerical reconstruction error. Therefore, a threshold of the reconstruction errors is set to classify the inputs. If the reconstruction error is larger than the threshold, the corresponding input is classified as malicious data; otherwise, it is considered normal. We test the autoencoders on all the data, including benign flow and malicious attacks, and observe two metrics: the false positive rate for the benign data, and the detection rate for the attacks.

Choosing a suitable threshold is also challenging. If the threshold is too low, most of the input will be identified as malicious, meaning that the false positive rate will be high. If the threshold is too high, most of the input will be identified as benign, meaning that the detection rate for attacks will be low. Therefore, in the following experiments, we evaluate multiple threshold values, observe how the metrics vary, and select the configuration that yields the best performance for further analysis.

4.5.2 Experiment results

In this part, we focus mostly on choosing the best bottleneck and threshold. We experimented with models using bottleneck sizes of 24, 32, and 40, and tested threshold values from 11 to 20. The results, including false-positive rates (FPR) and true-positive rates (TPR, or detection rates), are recorded.

To compare the performance between models with different bottleneck sizes and different thresholds, we use the Receiver Operating Characteristic (ROC) curve. It is used to visualize the performance of binary classification models. The ROC curve is created by calculating the TPR and FPR for possible thresholds and connecting the dots. Note that we only sample discrete thresholds, using them to approximate a continuous ROC.

To rank models with different bottleneck sizes without considering specific thresholds, we use the Area Under the Curve (AUC). It focuses on the ranking ability of a model, i.e., it represents the probability that a randomly chosen positive instance will be ranked higher than a randomly chosen negative instance by the model. Therefore, AUC is inherently threshold independent.

After the model is settled, we then consider Youden's J Statistic, i.e., $J = TPR - FPR$. It is used to find the optimal threshold. On the ROC curve, Youden's J represents the vertical distance between the ROC curve and the diagonal "random chance" line. The threshold that maximizes J is often chosen as the optimal cutoff point because it equally weights false positives and false negatives. In summary, it identifies the specific threshold where the model is most effective.

The summary of the results is shown in Figure 4.2. This figure shows that all 3 curves of 3 models with different bottlenecks are way above the random guess baseline (red dashed line), and their AUC are similar, which are close to 0.9. This indicates that

the models work decently. Moreover, we notice that the model with bottleneck size of 32 and threshold of 11 reached the highest Youden's J Statistic, which motivates us to use it in the following experiments.

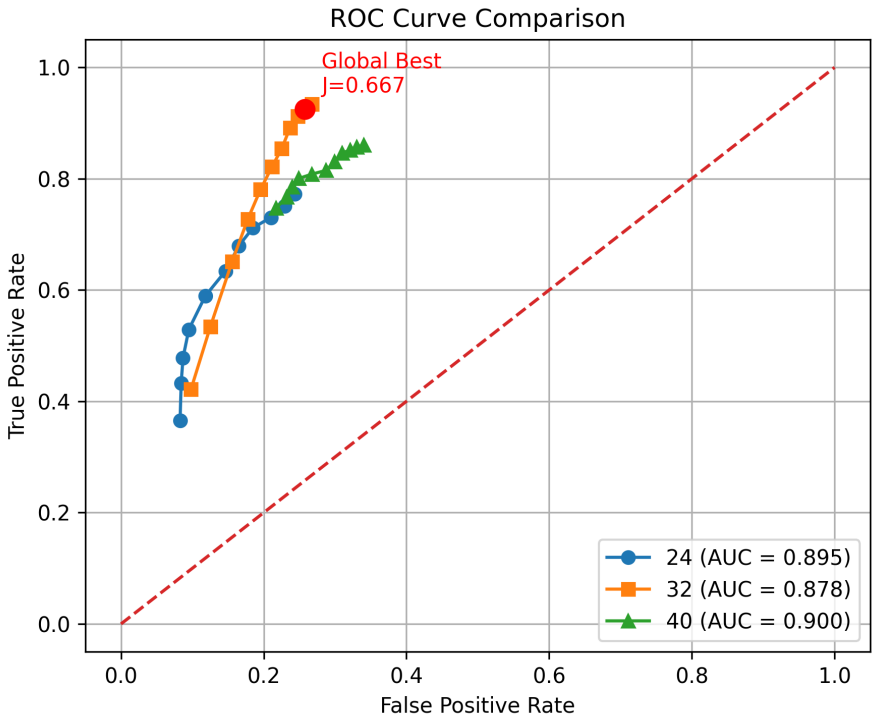


Figure 4.2: Comparison of autoencoders

Notably, the curves are truncated within $FPR < 0.4$ and $TPR > 0.3$, meaning that lower or higher thresholds (< 11 or > 20) are not considered, as excessively high false-positive rates or extremely low true-positive rates are unacceptable. Although Youden's J Statistic may peak at lower or higher thresholds, it is useless for anomaly detection.

In summary, we trained multiple autoencoders on the CAN-MIRGU dataset and selected a target model with an appropriate detection threshold for use in the subsequent experiments.

5

XAI Methods

The preceding chapter described the datasets and models used in the experiments. This chapter introduces the explainability methods, how they are applied and the results. These methods use different approaches to interpret the model predictions, providing a way to understand the internal black-box mechanisms of the models.

5.1 XAI Methods for MLP

This section describes the application of several explainability methods, including SHAP, LIME, Trustee, and Integrated Gradients (IG). We use these methods to explain the MLP models trained in the previous chapter to obtain importance scores and rankings for each input feature. While the model performed well on the CAN-MIRGU dataset, its performance on the can-train-and-test (CT&T) dataset was poor. By analyzing the explainability results and the features of various attacks in the datasets, we attempt to understand how the model identifies intrusions. This analysis aims to identify potential shortcuts and the reasons why the model does or does not utilize them.

In Chapter 2.5, the theoretical principles of the explainability methods used in this thesis were introduced. Therefore, those details will not be repeated here. Instead, this section provides a brief description of the configurations used for each method. The primary focus remains on the analysis of the explainability results and the datasets.

5.1.1 Experimental implementation

SHAP. For the SHAP method, we utilize the KernelSHAP [39] implementation provided in the Python SHAP package. While other variants such as GradientSHAP and DeepSHAP are designed specifically for deep learning models, KernelSHAP is model-agnostic. We choose KernelSHAP because we intend to test a more universal approach.

KernelSHAP operates by perturbing the data. It replaces specific input features with values from background samples to mask them, then compares the model's new prediction with the original one. This process is repeated for various features,

and a weighted average algorithm is applied to calculate the importance score for each feature.

Because KernelSHAP is computationally slow, we balance execution time with the quality of the results. We set the number of background samples, the sample count, and the number of perturbations all to 100. This configuration provides stable and representative results without requiring excessive time. To select the background data, we use K-means clustering to extract the 100 most representative data points from the training set. The test samples are randomly selected from the test set based on their classification categories.

The results are visualized using a beeswarm plot, and also bar plots. The plot provides a direct view of the SHAP results, indicating which features are more important and whether their impact on the output is positively or negatively correlated. This is useful for a quick understanding of the results. Additionally, we save all result data for subsequent detailed analysis. In the following sections, we do not display the beeswarm plots or histograms due to space constraints; instead, we present the top-ranked features in a tabular format.

LIME. The LIME method calculates results for one row of data at a time to produce a local explanation. To compare this method with other explainability techniques, we sample 1,000 rows of data, calculate their LIME values, and average them to obtain a macro-level explanation. We use the Python lime package for these calculations. After obtaining the results, we generate bar charts for visualization and recorded the data for detailed analysis. Additionally, we use LIME to generate explanations for specific individual cases, such as false positive samples, to observe the model's decision-making process for those small amount instances.

Integrated Gradients. Integrated Gradients differs from the other explainability methods as it is a white-box method that requires calculating internal model gradients. We choose this method because of its computational efficiency and high stability, making it suitable for comparison with other methods. The configurations remain consistent with those described previously. We randomly sample data from the datasets for calculation, generate bar charts of the results, and save all raw data.

5.1.2 Results and analysis

This part analyzes the explainability methods' results and the characteristics of the datasets to understand model behavior and identify the underlying causes. We conducted a detailed analysis of the MLP performance on both CAN-MIRGU dataset and CT&T dataset. Due to space limitations, we do not present every individual result here. Instead, we focus on examples that are clear and representative. Each dataset is analyzed separately, followed by a comparison of their results.

5.1.2.1 Analysis of XAI results on CAN-MIRGU

The MLP trained on CAN-MIRGU dataset uses multi-class classification to categorize data into benign messages, DoS attacks, fuzzing attacks, spoofing attacks, and

replay attacks. While the dataset also contains data for masquerade and suspension attacks, we did not include them because they are simulated, not collected from a real car [2]. In the following sections, we introduce and analyze the explainability results obtained for each of these four attack types.

DoS attack. The essential feature of a DoS attack is a surge in the number of messages on the CAN bus within a short time, regardless of the payload content. Therefore, the model is expected to focus on features such as message weight (the number of messages arriving within a time window) and jitter (the variation in message frequency).

However, an analysis of the explainability results in Table 5.1 shows that while the three methods provide slightly different feature rankings, they all indicate that the model focuses more on payload entropy within short time windows and raw data from specific payload bytes. Message weight is secondary to these features. This can be traced back to the configuration of DoS attacks by the dataset authors. In the CAN-MIRGU dataset, there are two types of DoS attacks: one injected every 0.001s and another every 0.02s. Although the injection frequencies differ by a factor of 20, the payloads of the injected messages are consistently 8 bytes of 0xFF.

As a result, payload features are more prominent than temporal features. A high volume of messages with all-0xFF payloads causes a sharp decrease in payload entropy within short time windows. In contrast, because different attacks may share similar injection frequencies, the model cannot easily distinguish specific attack types using temporal features alone. Additionally, the model focuses on raw data in specific bytes to differentiate DoS attacks from other categories, which will be analyzed in detail later.

Table 5.1: Top features of DoS attack in CAN-MIRGU dataset

Method	Top 1	Top 2	Top 3	Top 4	Top 5
Integrated Gradients	L500 mean p-entropy	payload 6	L100 weight	L100 mean p-entropy	payload 7
LIME	L500 mean p-entropy	payload 7	L100 mean p-entropy	L100 weight	L500 weight
SHAP	L500 mean p-entropy	payload 6	L100 weight	L100 mean p-entropy	L10 weight

Note: The 'L' prefix denotes the temporal window size; larger numerical values indicate smaller windows (e.g., L500 $\approx 1/500 = 0.002$ s). The same applies to other similar features.

Fuzzing attack. For the fuzzing attack, attackers inject a large number of random IDs and random payload contents. The essential features should be a significant increase in both ID entropy and payload entropy. We designed corresponding features during the feature extraction stage to capture these changes.

However, the explainability results in Table 5.2 show significant differences among the three methods. IG and SHAP suggest that the model focuses on jitter-related

features. In contrast, LIME indicates that the model prioritizes payload entropy over short time windows. While the latter seems more reasonable, a closer examination reveals an unexpected conclusion. Although high entropy is typically a defining feature of fuzzing, the model here associates lower entropy with fuzzing attacks. This behavior is caused by the dataset configuration. In the CAN-MIRGU dataset, all payloads in the fuzzing attacks are identical, with most bytes set to 0xFF. As a result, the model developed an illogical decision-making pattern based on this specific data bias.

Furthermore, we analyzed why the model failed to learn features related to ID entropy, despite the presence of random IDs in the fuzzing attacks. Compared to other features like payload entropy, the difference in global ID entropy between attack and normal messages is not distinct enough. Because our ID entropy is a cumulative global statistic, its value does not change significantly between adjacent messages. Consequently, normal data near a fuzzing attack are almost indistinguishable from the attack data based on ID entropy alone. Therefore, the model does not rely on this feature. While this reflects a limitation in our feature design, the resulting behavior is insightful. To defend against random ID fuzzing, a more effective approach would be to implement alerts for unknown CAN IDs, as the IDs on a CAN bus are generally fixed.

Table 5.2: Top features of fuzzing attack in CAN-MIRGU dataset

Method	Top 1	Top 2	Top 3	Top 4	Top 5
Integrated Gradients	L1 mean jitter	L1 std jitter	L10 mean jitter	L100 mean jitter	L500 std ham
LIME	L500 mean p-entropy	payload 7	L100 mean p-entropy	L1 mean id entropy	payload 6
SHAP	L1 mean jitter	L10 mean jitter	payload 6	L500 mean p-entropy	L1 std jitter

Spoofing attack. The CAN-MIRGU dataset contains a large number of different spoofing attacks. These target various ECU units to achieve different effects. In terms of injection methods, there are primarily two types: periodic injection (typically every 0.02s) and flam delivery (See Section 2.2). Although these two methods differ slightly, their injection frequencies are generally similar. We cannot generalize which features the model should or should not prioritize for spoofing attacks, as this typically depends on the specific ECU unit being spoofed. Generally, the model should focus on specific payload bytes and target IDs, depending on the specific implementation of different vehicles.

An analysis of the explainability results in Table 5.3 shows that while different methods vary slightly, they all indicate that the model focuses heavily on message weight within short time windows. This is a reasonable result, as introducing additional messages onto a highly regular CAN bus naturally increases the message frequency.

Replay attack. Replay attacks and spoofing attacks are very similar. The only difference is that the payloads in replay attacks are almost entirely legitimate.

Table 5.3: Top features of spoofing attack in CAN-MIRGU dataset

Method	Top 1	Top 2	Top 3	Top 4	Top 5
Integrated Gradients	L500 weight	L100 weight	L10 weight	payload 6	L1 weight
LIME	L500 weight	L500 std jitter	L100 weight	L10 weight	L500 std id entropy
SHAP	L500 weight	L100 weight	payload 6	can id	L10 weight

Based on the explainability results in Table 5.4, the features the model focuses on when predicting replay attacks are very similar to those for spoofing attacks. Both prioritize message weight within short time windows. This occurs because the injection frequency for replay attacks in the CAN-MIRGU dataset is consistently 0.02s. The model also focuses on specific payload bytes. The logic is clear: the model uses these specific bytes to distinguish between a replay attack and a spoofing attack. This is an obvious shortcut resulting specifically from the features of the current dataset, similar to the patterns observed in the previously discussed attacks.

Table 5.4: Top features of replay attack in CAN-MIRGU dataset

Method	Top 1	Top 2	Top 3	Top 4	Top 5
Integrated Gradients	L500 weight	L500 std p-entropy	L100 weight	payload 5	payload 1
LIME	L500 weight	L100 weight	L1 std p-entropy	L10 std jitter	L500 std jitter
SHAP	L500 weight	L500 std p-entropy	payload 1	L500 mean p-entropy	payload 5

Trustee result. We applied Trustee [45] to generate a decision tree to approximate the MLP's behavior. The decision tree reached a fidelity of 97.6%, meaning its predictions matched the MLP's output with 97.6% similarity. Since the MLP achieved near-perfect accuracy on the CAN-MIRGU dataset, the decision tree's behavior primarily reflects the attack patterns inherent in the data rather than the MLP's internal logic. Figure 5.1 shows the resulting decision tree.

The following patterns can be summarized from the tree.

The tree first examines the last payload byte. A high value identifies the sample as a DoS or fuzzing attack. This is likely because only DoS attack and fuzzing attack in this dataset use 0xFF for the final byte, while other categories reach a maximum of 0xF8.

For DoS attack or fuzzing attack samples, the tree checks the CAN ID entropy; higher entropy indicates fuzzing attack, while lower entropy indicates DoS attack.

If the sample is neither, the tree checks the jitter. High jitter suggests a spoofing

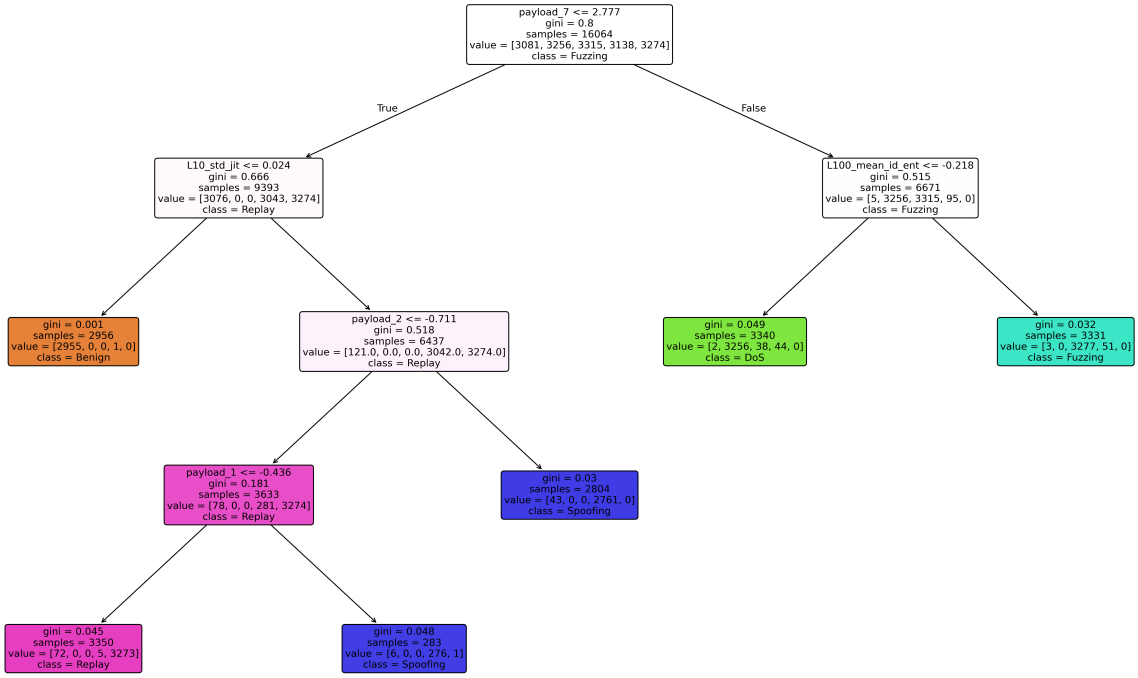


Figure 5.1: Trustee decision tree of MLP on CAN-MIRGU dataset

attack or replay attack, while low jitter indicates a benign frame. To differentiate between spoofing attacks and replay attacks, the tree inspects specific payload bytes.

In short, the decision tree achieves high accuracy by relying on jitter and a few payload bytes. This is possible because most attacks in the dataset share similar injection frequencies and use limited, distinct payload patterns. The simplicity of the Trustee-generated decision tree demonstrates how a model might adopt the most expedient strategy for prediction, revealing the underlying flaws of the dataset.

Overall, we observed that the MLP focuses on features primarily to differentiate between various attack categories. It appears less concerned with the fundamental distinction between attack and benign traffic. While this is acceptable given the current high accuracy, it may represent a vulnerability in certain scenarios. Zhang et al. [62] implement an IDS that first determines whether traffic is normal or an attack, then utilizes a second classifier to categorize the specific attack type. This two-stage approach could be preferable, as it helps mitigate shortcuts that might otherwise lead to a failure in detecting attacks.

5.1.2.2 Analysis of XAI results on can-train-and-test

The CT&T dataset differs from the CAN-MIRGU dataset. Its attack patterns are difficult to categorize into several types. Consequently, we adopted a binary classification approach to distinguish only between normal and attack traffic. The test data is divided into four subsets, as described in Chapter 4.2. Structurally, each attack type is stored in a separate CSV file. We calculated individual explainability results for each file to allow for the analysis of each specific attack. We will examine

representative cases and explain why the model failed entirely on certain test subsets.

DoS attack. In the training set of the CT&T dataset, the payloads of all DoS attacks consist of all 0x00. However, the test set includes DoS attacks with entirely different payloads. The results demonstrate the model's generalization capability, as it successfully identifies the majority of these attacks despite the payload differences.

Explainability results in Fig 5.2 show that the model does not rely on payload entropy features to identify attacks. Instead, it prioritizes the weight feature (message count) within extremely short time windows. This aligns with the essential characteristics expected of a DoS attack. This outcome occurs because the training process requires the model to identify a broad range of attacks, and while high message density is common to most attack types in this dataset, low payload entropy is unique to DoS. Compared to the analysis of DoS attacks in the CAN-MIRGU dataset, the results here are significantly more robust.

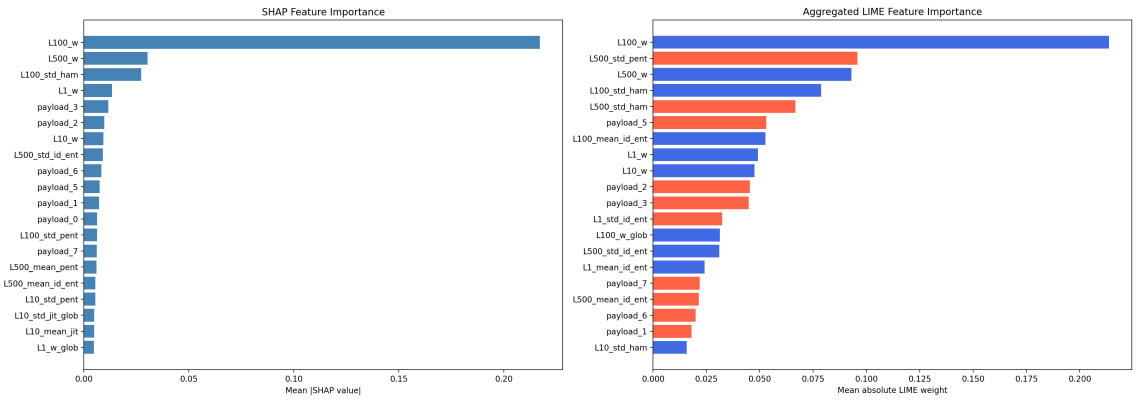


Figure 5.2: SHAP and LIME results of DoS attack in CT&T dataset

Note: In LIME results, blue positively contributes to the prediction, while red indicates the opposite. Subsequent figures follow the same logic.

Force neutral spoofing attack. The force neutral spoofing attack also achieved high accuracy. However, unlike DoS attacks, the high performance in this case stems from a shortcut. Explainability results consistently show that the model first examines the message weight within short time windows, followed by specific payload bytes.

The model uses message weight in short time windows to identify attacks, which is a reasonable approach. In this dataset, spoofing attacks inject new frames at high frequencies, leading to an increase in message counts. To distinguish attack frames from benign frames within the same attack window, the model then inspects specific payload bytes, as these values consistently differ between the two categories. In this instance, the model successfully achieves high performance by combining fundamental characteristics with shortcuts. This represents a case where shortcut learning leads to a successful outcome.

The following paragraph describes an instance where the reliance on shortcuts leads to failure.

RPM spoofing attack. For the RPM spoofing attack, the explainability results indicate that the model primarily relies on the Hamming distance within short time windows. This feature is negatively correlated with the prediction: a smaller Hamming distance increases the likelihood that the model classifies the data as an attack. This behavior occurs because the attack payloads are identical, and the injection frequency during an attack is significantly higher than the normal message frequency. Consequently, the sequence of the target CAN ID becomes saturated with identical messages, leading to a very low Hamming distance.

However, the baseline Hamming distance for the normal RPM messages is high. Because our statistical features are cumulative, several consecutive attack messages must be processed before the Hamming distance is reduced sufficiently to trigger an attack classification. This requirement leads to the failure of most predictions. Hamming distance is an undesirable feature for identifying RPM spoofing, and the model's over-reliance on it resulted in poor performance. Table 5.5 shows an RPM spoofing attack message sequence in the dataset. Three consecutive attack messages were predicted as benign because their features of hamming distance were not low enough. The last attack message was finally detected by the model.

Table 5.5: CAN message sequence of RPM spoofing attack

Message count	Result	other features	Hamming Distance	other features
1	TN	...	7.174	...
2	FN	...	11.433	...
3	FN	...	4.846	...
4	FN	...	1.902	...
5	TP	...	0.762	...

This dataset contains four test subsets. The examples mentioned above are all from a test set with known attacks collected from known vehicles. In comparison, the results for unknown attacks collected from known vehicles differ. These attacks are entirely absent from the training set, allowing for a test of the model's generalization capability against new threats.

Fuzzing attack. The unknown attacks include fuzzing attacks, which inject random payloads into multiple CAN IDs. The model gets zero F1-score on these attacks because the training data does not encourage the model to focus on fuzzing-related features, such as global statistics for CAN IDs.

Speed spoofing attack. Speed spoofing attack is also included among the unknown attacks. Although the target IDs and payloads differ from those in the training set, the injection patterns and frequencies are very similar to known attacks. For instance, both speed spoofing and the previously mentioned force neutral spoofing use an injection interval of 0.003s, and their target IDs have similar transmission

frequencies. This allows the model to identify these attacks with very high accuracy. Explainability results show that the model employs nearly identical decision logic for speed spoofing as it does for force neutral spoofing. Figure 5.3 shows the SHAP results of the two spoofing attacks. They show great similarity.

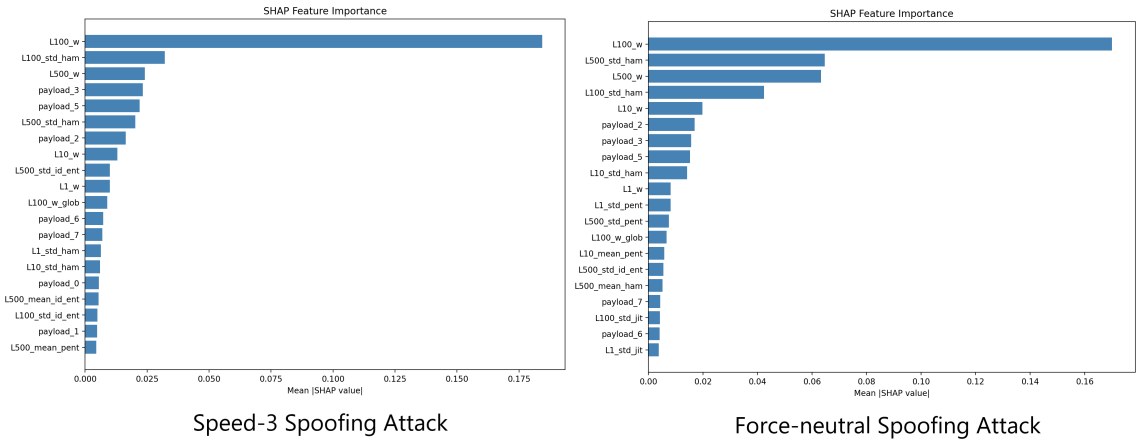


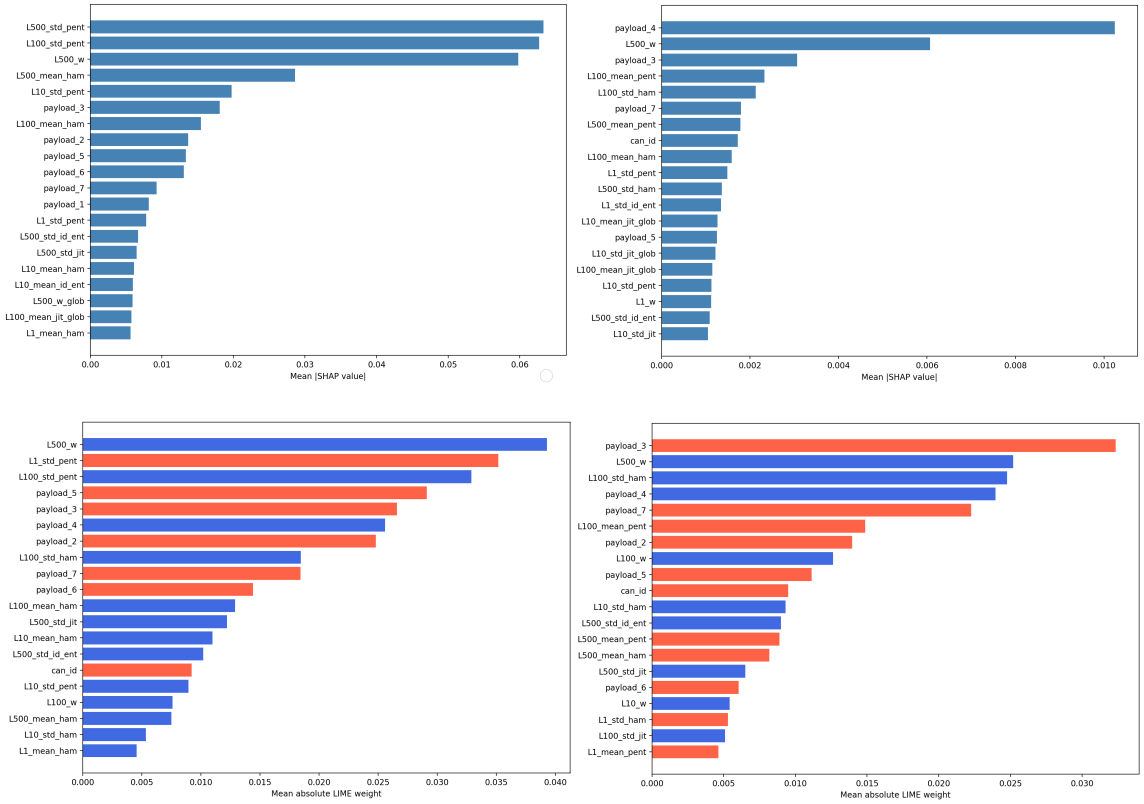
Figure 5.3: Comparison of SHAP results between speed spoofing attack and force-neutral spoofing attack

Interval spoofing attack. The unknown attacks also include a group of interval spoofing attacks. These target the same ID but use different payloads, resulting in a notable contrast: the model identifies one attack perfectly but fails to detect the other. Both attacks utilize flam. Because the target ID has a low frequency, a short time window contains only one normal message and one attack message with the targeted CAN ID. This creates a distinct statistical difference between benign and attack data within that window. However, because the message counts for other attacks in the training set are much higher, the model does not consider a two times increase (from 1 to 2) sufficient to trigger an attack label. Consequently, temporal features are ineffective here.

To make a prediction, the model relies more on payload-related statistical features. Explainability results show that the model focuses on payload entropy. Due to the dataset configuration, one attack uses a payload that is nearly all 0x00, which differs significantly from normal messages and causes a large change in payload entropy. The other attack uses complex payloads similar to legitimate messages, resulting in minimal entropy change. Since the model relies heavily on this specific feature, it successfully detects the first attack while completely ignoring the second.

5.1.3 Description of degenerated explanations

The CT&T dataset contains two sub testsets with data collected from different vehicles from the training set, on which the model failed completely. It classified all messages as benign, failing to identify any attacks. This outcome is expected, as the data patterns and statistical features of these testsets differ entirely from the



Result of interval spoofing attack with payload "00 00 00 01 00 00 00 00"

Result of interval spoofing attack with payload "1B 4C 05 11 1B 51 1C 69"

Figure 5.4: SHAP and LIME results of interval attacks with different payloads

training data. In practice, a deep learning model would typically require transfer learning using data from the new environment rather than being applied directly.

Applying explainability methods to these cross-vehicle subsets also resulted in a total failure. Warnecke et al. [63] refer to this phenomenon as the generation of "degenerated explanations." Because the model consistently predicts all data as benign, perturbations applied to the samples fail to alter the model's decision, preventing methods like SHAP and LIME from generating valid explanations. Similarly, although Integrated Gradients appeared to produce results, they were meaningless; the method generated identical explanations for all files within a subset, and the result shifted—while remaining uniform—upon changing the random seed. This indicates that the results are invalid and lack significance.

5.1.4 Discussions of explainability results

Based on the analysis above, several conclusions can be drawn:

- Shortcuts are pervasive. It is common for models to adapt to specific dataset characteristics. Depending on how a shortcut is defined, a model may rely on these patterns either entirely or partially to make decisions.
- Explainability methods are essential for behavior analysis. These techniques allow for the observation and understanding of model logic, which is critical for refining models and identifying potential vulnerabilities.
- Dataset characteristics often dictate shortcut reliance. The failure to capture essential features is sometimes driven by the nature of the dataset rather than insufficient model capacity or flawed training procedures. Consequently, unless a dataset is confirmed to cover all possible attack scenarios, it is necessary to rigorously audit and refine the model after the training phase.

5.2 Evaluation of XAI Methods

The analysis in the previous section indicates that different explainability methods provide varying, and sometimes contradictory, results for the same model and data. These discrepancies stem from their distinct underlying principles. Consequently, a basis is required to determine which method is more reliable.

Warnecke et al. [63] proposed several metrics to evaluate explainability methods, including Accuracy, Sparsity, Completeness, Stability, Efficiency, and Robustness. This section explains these attributes and uses them to compare the explainability methods applied to our MLP model.

Accuracy. This metric measures the alignment between the feature importance assigned by an explainability method and the actual impact of those features on the model's predictions. Accuracy is typically evaluated using a feature ablation (or masking) test: features are removed (or replaced with a baseline value) in order

of their importance, and the resulting drop in prediction accuracy or confidence is observed. A sharp decline in model performance after removing high-weight features indicates that the method accurately identifies the core factors driving the prediction. Conversely, if the prediction remains largely unaffected, the explanation is considered biased or inaccurate. Figure 5.5 shows the accuracy evaluation for the MLP trained on two different datasets. For the MLP trained on the CT&T dataset, all methods demonstrate high accuracy, as prediction confidence drops rapidly after removing the first few features. SHAP performs slightly better in this case. For the MLP trained on the CAN-MIRGU dataset, Integrated Gradients (IG) significantly outperforms the others; removing the top 15 features reduces the model's performance to the level of random guessing.

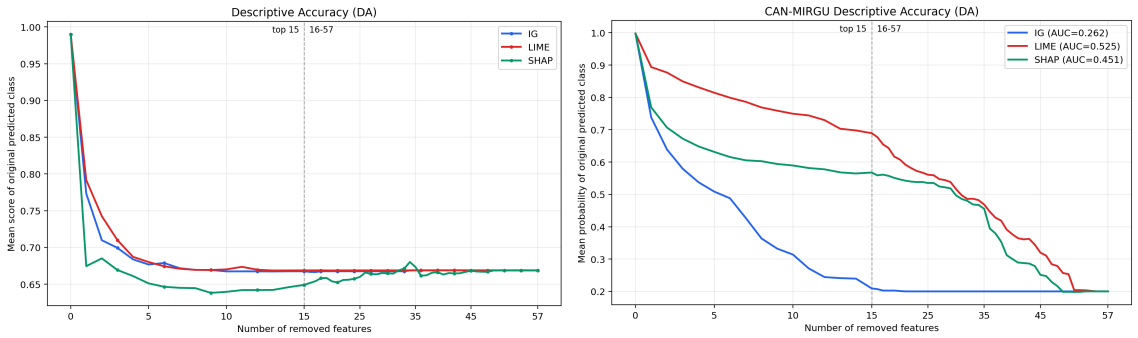
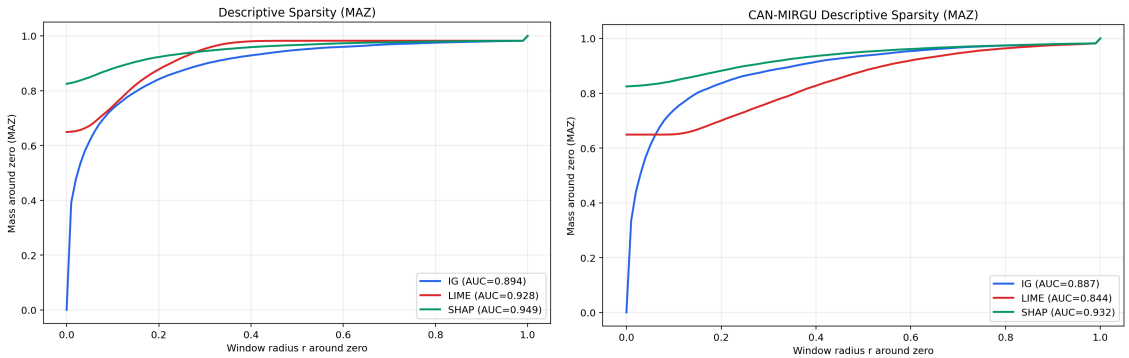


Figure 5.5: Accuracy of XAI methods

Sparsity. Sparsity describes whether the explanation is concentrated on a few key features or spread thinly across many. Highly sparse results are preferred as they filter out background noise and highlight the most decisive features, simplifying subsequent analysis. In our implementation, we scaled the attribution values for each sample to the range [-1, 1] and calculated the proportion of values falling within a near-zero window [-r, r]. A higher AUC of the Mass Area over Zero (MAZ) indicates higher sparsity. Figure 5.6 shows the sparsity results. Regardless of the dataset, SHAP exhibits slightly higher sparsity than the other two methods, while IG shows the lowest sparsity.

Completeness. Completeness refers to the ability of a method to consistently generate non-degenerate (meaningful and valid) explanations for all possible input samples. As previously mentioned in Section 5.1.3, SHAP and LIME fail to provide non-degenerate explanations in certain cases. While IG always produces an output, it can also be invalid under specific conditions.

Stability. Stability measures the consistency of the feature importance distribution when the method is presented with identical or highly similar inputs. According to the measuring method in the paper [63], we evaluate the stability of the explanation methods when processing inputs from the MLP models trained on CAN-MIRGU dataset and CT&T dataset respectively. We apply the explanations to the same



Result of MLP trained on can-train-and-test dataset Result of MLP trained on CAN-MIRGU dataset

Figure 5.6: Sparsity of XAI methods

samples over multiple runs and measure the average intersection size between the runs.

Table 5.6 shows the average intersection size between the top 10 features for four runs of the three XAI methods. We can see that in both situations, LIME methods obtains a higher result of 0.632 and 0.515, while SHAP method gets results under 0.5. The outputs of IG are deterministic. It provides consistent explanations, so the result is 1.0.

Table 5.6: Stability of different XAI methods

XAI methods	Result of MLP on CAN-MIRGU	Results of MLP on CT&T
SHAP	0.400	0.379
LIME	0.632	0.515
IG	1.000	1.000

Robustness. Robustness evaluates the ability of an explainability method to resist deception and maintain truthful results when faced with adversarial inputs or malicious attacks. Currently, according to Warnecke et al. [63], there are no well-established methods to test the robustness of explainability techniques. Thus this attribute is not explored in this study.

Efficiency. Efficiency assesses the computational resources, primarily execution time, required to generate an explanation. During our experiments, the execution time for KernelSHAP was significantly longer than that for LIME and IG. While the computational time for LIME and IG grows linearly with the number of samples, KernelSHAP's complexity grows exponentially. Due to their speed, LIME and IG can be integrated into security systems for real-time explanation generation, whereas SHAP is generally limited to post-hoc research.

Comprehensive analysis of these three methods indicates that SHAP provides the

best overall performance but requires the most computation time. IG is fast and stable but lacks sparsity. LIME, being designed for individual sample explanations, is best suited for investigating specific cases, such as analyzing the model's logic behind a single false positive within a large dataset.

5.3 AE-p-values for Autoencoders

We now employ AE-p-values to generate explanation results for the autoencoder model trained on the CAN-MIRGU dataset, analyzing its behavior and the corresponding characteristics of the dataset.

5.3.1 Feature attribution using p-values

In the following experiment, we examine the feature attribution results of the autoencoder trained on the CAN-MIRGU dataset, derived from p-value–based explanations. Notably, the computation is expensive, so we implemented an efficient GPU-accelerated script for computing AE-p-values. The implementation details are provided in appendix A.1.

Table 5.7 provides a general overview of the feature attribution results by summarizing, for each attack type, the five features that most frequently appear among the top five anomaly-ranked features. Specifically, for each sample, features are ranked according to their p-values, where smaller p-values indicate stronger deviations from normal behavior and therefore higher anomaly significance. For example, suppose there are 100 DoS attack samples, and feature X appears among the top five most anomalous features in 43 of those samples. In this case, the presence frequency of feature X within the top-five feature set is calculated as 43%. Based on this criterion, we select, for each attack category, the five features with the highest presence frequencies as the most representative anomalous features for that attack type.

Table 5.7: Top-5 anomalous features for each attack

Attack	**Top-5 anomalous features**
DoS	L1_mean_jit_glob (94.7%), L0.1_mean_jit_glob (91.2%), L1_mean_pent (72.7%), L0.1_mean_pent (72.5%), L5_mean_pent (71.1%)
Fuzzing	L0.1_mean_pent (61.9%), L1_mean_id_ent (58.8%), L1_mean_pent (49.0%), L1_mean_jit_glob (48.9%), L1_std_jit_glob (48.3%)
Spoofing	L0.1_mean_jit_glob (93.4%), L1_mean_jit_glob (82.1%), L1_std_jit_glob (81.6%), L5_mean_id_ent (58.9%), L5_std_jit_glob (55.7%)
Replay	L1_mean_jit_glob (99.5%), L0.1_mean_jit_glob (97.6%), L1_std_jit_glob (92.7%), L5_mean_id_ent (88.0%), L1_mean_id_ent (77.4%)

We then evaluate the behavior of the autoencoder through the feature attribution results derived from p-values.

For the **DoS** attack, global jitter-related features are the most important, appearing among the top five anomalous features in over 90% of the samples. This observation is consistent with the fundamental nature of DoS attacks, where high-frequency message injection disrupts the normal timing behavior of CAN communications. Such attacks significantly alter transmission periodicity and timing consistency, making temporal jitter features highly discriminative indicators of anomalous behavior.

However, payload entropy features also frequently appear among the top-ranked anomalous features. This may indicate that the DL-NIDS has partially learned a dataset-specific shortcut rather than relying solely on robust attack characteristics. In the CAN-MIRGU dataset, DoS attack messages contain fixed payloads consisting entirely of byte 0xFF. Consequently, when these attack messages dominate the CAN bus, the overall payload entropy becomes unusually low, making entropy-based features highly predictive for the model. Nevertheless, this pattern may not generalize well to real-world scenarios, where the payloads of DoS messages can be randomized or dynamically generated. Under such conditions, payload entropy deviations may become much less reliable as indicators of attack behavior. Therefore, from a robustness and generalization perspective, the ideal IDS should rely more heavily on intrinsic temporal abnormalities caused by high-frequency injection, rather than overfitting to dataset-specific payload entropy patterns.

For the **fuzzing** attack, payload entropy and ID entropy emerge as the most important features. This observation is intuitive, since fuzzing attacks arbitrarily manipulate CAN IDs and payload values, thereby significantly disrupting the statistical distributions of both message identifiers and payload contents. As a result, entropy-related features become highly sensitive indicators of such abnormal behaviors.

In the CAN-MIRGU dataset, the fuzzing attack messages contain payloads filled entirely with `FF` values. Consequently, the injected attack traffic reduces the diversity of payload values on the CAN bus, leading to lower payload entropy and ID entropy measurements. However, in real-world scenarios, fuzzing attacks may instead use randomly generated payloads and identifiers, which could increase rather than decrease entropy values during attacks. Despite this discrepancy, the model may still generalize reasonably well to such cases. This is because both unusually high and unusually low entropy values represent significant deviations from normal CAN traffic behavior. Therefore, either increase or decrease in entropy can produce substantial reconstruction errors in the corresponding feature dimensions, allowing the anomaly detection model to identify fuzzing attacks even when the attack distribution differs from that observed in the training dataset.

For the **spoofing** and **replay** attacks, the attribution patterns are less straightforward. Intuitively, the most salient feature deviations for these two attack types would be expected to occur in per-ID jitter-related features, since both attacks primarily involve injecting messages associated with specific CAN IDs. Such targeted

injections should mainly disrupt the temporal behavior and transmission regularity of the affected IDs rather than the overall global traffic distribution.

However, the attribution results indicate that the model relies more heavily on global features, even though the injection frequencies of these attacks are relatively low compared with DoS-style attacks. From an explainability perspective, these learned feature preferences do not fully align with prior domain knowledge regarding the expected characteristics of spoofing and replay attacks. Nevertheless, the target model still achieves strong detection performance, reaching an approximately 88% detection rate for these attacks. This indicates that the selected features still contain discriminative information useful for identifying anomalous behaviors, even if the underlying decision process differs from expectations.

In summary, the analysis of p-value-based feature attribution provides important insights into the decision-making behavior of the model. On the one hand, the results validate several intuitively meaningful detection strategies learned by the autoencoder, such as relying on global jitter-related features to identify DoS attacks and focusing on entropy-related features to recognize fuzzing attacks. These findings are consistent with the known behavioral characteristics of the corresponding attack types and suggest that the model captures certain semantically relevant anomaly patterns.

On the other hand, the attribution analysis also reveals several potential shortcut features exploited by the model, such as the strong reliance on payload entropy for detecting DoS attacks in the CAN-MIRGU dataset. While such features may improve performance on the benchmark dataset, they may not generalize reliably to real-world attack scenarios where attack payload distributions differ significantly. Therefore, the attribution results not only help explain why the model is able to detect attacks effectively, but also provide valuable insight into whether the learned decision logic is robust, semantically meaningful, and aligned with domain knowledge. Overall, this analysis improves our understanding of how the autoencoder-based IDS identifies anomalous CAN traffic and highlights the characteristics of the attacks in the target dataset.

5.3.2 Clustering attacks using p-values

In this experiment, we employ p-values to perform clustering attacks and motivate their use by comparing the clustering performance obtained from p-values with that derived from raw inputs.

To obtain the best clustering outcome, we first evaluate multiple clustering candidates, including raw inputs and p-values, and then select the configuration that achieves relatively strong performance. Based on the chosen clustering result, we further analyze the characteristics of the detected attacks.

5.3.2.1 Clustering candidates

We perform clustering in both the raw input space and the explainable representation space. For the input-space approach, the original feature vectors are directly used as clustering candidates. In contrast, the explainable representation space is designed to capture anomaly-related semantics derived from the autoencoder. Specifically, feature-wise reconstruction errors are used as a baseline explainable representation, while p-value representations are adopted as the primary target for performance evaluation.

To further improve clustering quality, we additionally introduce several variants of the p-value representation:

1. **Surprisal values (s-values)**: Defined as $s_i = -\log(p_i + \epsilon)$, this metric evaluates the rarity of an observation by quantifying how small its associated probability is. The negative logarithmic transformation specifically accentuates extreme anomalies where $p_i \approx 0$. In this regime, minor variations in p_i correspond to substantial shifts in the deviation of the input dimension from the null hypothesis H_0; the log transformation effectively magnifies these subtle yet critical deviations.

2. **Sparsified s and p vectors**: To mitigate noise, we can retain only the top 10 most significant dimensions, characterized by the smallest p_i and largest s_i, while setting the values of all remaining dimensions to their respective baseline states (i.e., $p_i := 1$ and $s_i := 0$). This sparsification strategy is inspired by the distance collapse phenomenon [64] inherent to high-dimensional spaces, where the relative contrast between the maximum and minimum distances vanishes, i.e., $\frac{d_{\max} - d_{\min}}{d_{\min}} \to 0$. By applying this sparse mask, irrelevant background noise is suppressed, thereby significantly enhancing the separability of true anomalous signals.

In summary, the selected clustering candidates are: input vectors x, dimensional reconstruction error $\|\hat{x} - x\|$, p-values p, s-values s, sparsified p-values p_{topk} and sparsified s_{topk}.

5.3.2.2 General performance improvement strategies

In addition, we introduce two general strategies to enhance clustering performance:

1. **Class balancing**: This step is crucial in clustering paradigms because severe class imbalance tends to bias algorithms toward majority groups, thereby obscuring well-defined minority structures. In the case of the CAN-MIRGU dataset, the sample sizes are highly skewed: DoS and fuzzing attacks comprise approximately 10k samples each, whereas replay and spoofing attacks contain around 147k and 57k samples, respectively. To address this, we downsample the overrepresented attack categories to a uniform baseline of 10k samples.

2. **Dimensionality reduction (PCA)**: Principal Component Analysis (PCA) effectively filters out background noise and redundant features, rendering the

underlying geometry of the data more discernible. By projecting the high-dimensional data onto a lower-dimensional subspace that captures the maximum variance, PCA significantly improves cluster separability and alleviates the curse of dimensionality.

5.3.2.3 Evaluation metrics for clustering results

To assess the quality of the clustering outcomes, we evaluate the following metrics:

1. External metrics: Adjusted Rand Index (ARI), Normalized Mutual Information (NMI), Fowlkes–Mallows Index (FMI), and Purity. Those metrics measure the agreement between predicted clusters and ground truth labels from different perspectives. Details are shown in Table 5.8.

   Notably, purity only works when the number of classes is reasonable, since smaller clusters tend to show higher purity. Thus, it is only used to pick up the best setup combined with number of clusters, and will not be used in most of the comparisons.

2. Internal metrics: Davies–Bouldin Index (DBI), Calinski–Harabasz Index (CH), and Noise Ratio (NR). Those metrics evaluate clustering quality based solely on the intrinsic data structure, without relying on ground-truth labels. They typically assess the trade-off between intra-cluster compactness and inter-cluster separation. Details are shown in Table 5.9.

Table 5.8: External clustering evaluation metrics: ARI, NMI, FMI, and purity

Metric	**Principle**
$\text{ARI} = \frac{\text{RI} - \mathbb{E}[\text{RI}]}{\max(\text{RI}) - \mathbb{E}[\text{RI}]}$	Pair counting metric with adjustment for chance agreement between cluster assignments (higher is better).
$\text{NMI} = \frac{I(X;Y)}{\sqrt{H(X)H(Y)}}$	Measures information-theoretic dependence between clustering result and ground truth labels (higher is better).
$\text{FMI} = \sqrt{\text{Precision} \cdot \text{Recall}}$	Geometric mean of pairwise precision and recall for clustering consistency (higher is better).
$\text{Purity} = \frac{1}{N} \sum_k \max_j \lvert C_k \cap L_j \rvert$	Measures the extent to which each cluster contains a single dominant ground-truth class (higher is better).

5.3.2.4 Clustering algorithm: HDBSCAN

The clustering algorithm selected for our experiments is HDBSCAN [65], which offers several distinct advantages over traditional methods such as K-means or DBSCAN. Notably, HDBSCAN automatically determines the optimal number of clusters, thereby eliminating the need for manual specification. Unlike algorithms that

Table 5.9: Internal clustering evaluation metrics: DBI, CH, and noise ratio

Metric	Principle
$\text{DBI} = \frac{1}{K}\sum_{i=1}^{K}\max_{j\neq i}\frac{\sigma_i+\sigma_j}{d(c_i,c_j)}$	Measures the average similarity between clusters by comparing intra-cluster dispersion with inter-cluster separation (lower is better).
$\text{CH} = \frac{\text{tr}(B_k)/(K-1)}{\text{tr}(W_k)/(N-K)}$	Evaluates the ratio of between-cluster dispersion to within-cluster dispersion, favoring well-separated and compact clusters (higher is better).
$\text{NR} = \frac{N_{\text{noise}}}{N}$	Quantifies the proportion of data points identified as noise, reflecting robustness and outlier sensitivity (lower is better).

assume spherical cluster shapes or uniform density distributions, HDBSCAN is capable of identifying clusters of arbitrary shapes and varying densities by constructing a density-based hierarchy. Furthermore, it exhibits robust noise handling by explicitly classifying outliers rather than forcing every data point into a cluster.

The performance of HDBSCAN is primarily governed by two critical hyperparameters that require tuning: *min_cluster_size*, which dictates the minimum number of samples required to form a cluster; and *min_samples*, which parameterizes the model's sensitivity to noise. All other hyperparameters were kept at their default values throughout our experiments.

5.3.2.5 Quality of clustering using different candidates

First, we conducted experiments to evaluate the empirical necessity of class balancing and dimensionality reduction. Specifically, fixing the HDBSCAN hyperparameters at *(min_cluster_size, min_samples)* = (1000, 100), we performed clustering on all representation candidates (e.g., raw input, p-values, and surprisal values) across configurations with and without PCA and downsampling. We then extracted the optimal results among these candidates for each specific setup. These comparative results are summarized in Table 5.10, where the best-performing metrics are highlighted in bold.

Evidently, under identical hyperparameter settings for HDBSCAN, incorporating both class balancing and dimensionality reduction consistently enhances performance across the majority of evaluating metrics.

Table 5.10: Performance comparison with and without using PCA/downsampling

PCA/Balance	ARI	NMI	FMI	NR	DBI	CH
N/N	0.1472	0.3763	0.5951	**0.0832**	0.8822	163230.3
Y/N	0.1890	0.3403	**0.6357**	0.0865	0.5945	267504.2
Y/Y	**0.3932**	**0.5150**	0.6237	0.0937	**0.1190**	**1007336.1**

Having validated the efficacy of PCA and downsampling, we subsequently conducted experiments to evaluate and compare the clustering performance across different representation candidates. Regarding the hyperparameters, we explored nine distinct *(min_cluster_size, min_samples)* combinations, executing the clustering pipeline, integrated with PCA and downsampling, for all candidates under each configuration.

Serving as the baseline, the clustering performance achieved using raw input vectors represents the upper bound of our control group, with the corresponding optimal results compiled in Table 5.11. It is worth noting that these peak performances were harvested from multiple separate hyperparameter setups rather than appearing simultaneously within a single configuration.

Table 5.11: Best performance for clustering using input vectors

ARI	NMI	FMI	NR	DBI	CH
0.3428	0.5062	0.5200	0.0832	0.8822	163230.3

For the experimental group, which comprises explainable representations of the input, we evaluated per-feature reconstruction errors, p-values, surprisal values (s-values), as well as the sparsified variants (top-10 features) of both p-values and s-values. To quantify their efficacy, we recorded the frequency with which each candidate representation achieved the optimal performance across the evaluated setups. These comparative frequencies are summarized in Table 5.12.

Table 5.12: Performance for clustering using explainable representations

Candidate	ARI	NMI	FMI	NR	DBI	CH	Sum
rec-err	1	1	0	1	1	2	6
p-values	4	3	3	5	0	0	15
s-values	4	5	6	3	6	6	30
p-sparse	0	0	0	0	0	0	0
s-sparse	0	0	0	0	2	1	3

Evidently, surprisal values (s-values) achieved the optimal performance across the vast majority of evaluated scenarios. Specifically, in terms of external validation metrics (i.e., fidelity to the ground-truth labels), both p-values and s-values significantly outperformed the alternative candidates. Regarding the intrinsic geometric quality of the clusters (i.e., intra-cluster cohesion and inter-cluster separation), the s-value representation demonstrated absolute dominance.

The best results of clustering using explainable representations are shown in Table 5.13. Same as Table 5.11, the best results are selected in multiple setups, and they did not appear simultaneously.

Comparing Table 5.13 with Table 5.11 reveals that clustering based on explainable representations consistently outperforms the raw input vectors across all evaluated metrics.

Table 5.13: Best performance for clustering using explainable representations

ARI	NMI	FMI	NR	DBI	CH
0.4314	0.5826	0.6659	0.0276	0.4475	1007336.1

In conclusion, among all the evaluated clustering candidates, including raw input and explainable representations, p-values and their logarithmic variant, s-values, show superior capability and yield better clustering structures than the other candidates.

5.3.2.6 Analysis of the best clustering result

We selected a representative configuration of the explainable representations for in-depth analysis. Specifically, under the hyperparameter setup of *(min_cluster_size, min_samples)* = (3000, 30), the clustering quality achieved via the s-value representation is detailed in Table 5.14.

Table 5.14: Quality of the target clustering result

ARI	NMI	FMI	NR	DBI	CH	Purity	# of clusters
0.4314	0.5709	0.6650	0.0728	0.7734	70892.6	0.6074	4

Evidently, the external evaluation metrics (ARI, NMI, and FMI) of the selected clustering result are relatively good compared to alternative setups, indicating a high degree of fidelity between the resulting clusters and the ground-truth labels. Moreover, the generated number of clusters is in perfect alignment with the actual number of attack categories, accompanied by an acceptable level of cluster purity. Additionally, the model maintains a well-bounded noise ratio of merely 7%, which is highly reasonable for this context.

The visualization of the selected clustering result is presented in Figure 5.7, which details the cluster-wise composition of different attack categories. Specifically, true labels 1, 2, 3, and 4 correspond to DoS, fuzzing, spoofing, and replay attacks, respectively. Additionally, the leftmost bar (designated as cluster ID = -1) isolates the noise points that were rejected by the algorithm, encompassing approximately 7% of the total dataset. The relative size of the clusters is shown in Figure 5.8.

According to Figure 5.7, we can extract several informative messages:

1. **Clusters 0 and 1**: These clusters correspond explicitly to fuzzing and DoS attacks, respectively. Each captures the vast majority of its respective attack category (Cluster 0 encapsulates 71% of all fuzzing samples, while Cluster 1 contains 72% of DoS samples). Furthermore, both clusters exhibit exceptionally high purity, quantified at 96% for Cluster 0 and a perfect 100% for Cluster 1.

2. **Cluster 2**: This cluster delineates the overlapping boundary between DoS and fuzzing attacks within the explainable representation space (i.e., the p-value

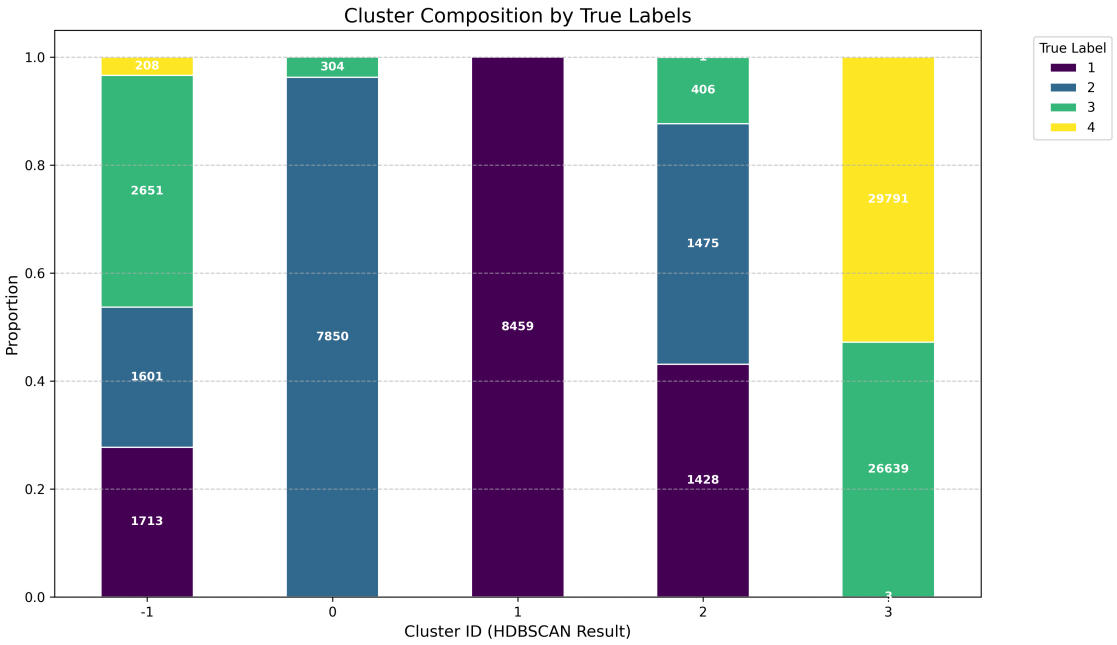


Figure 5.7: Proportion of each attack type in the different clusters

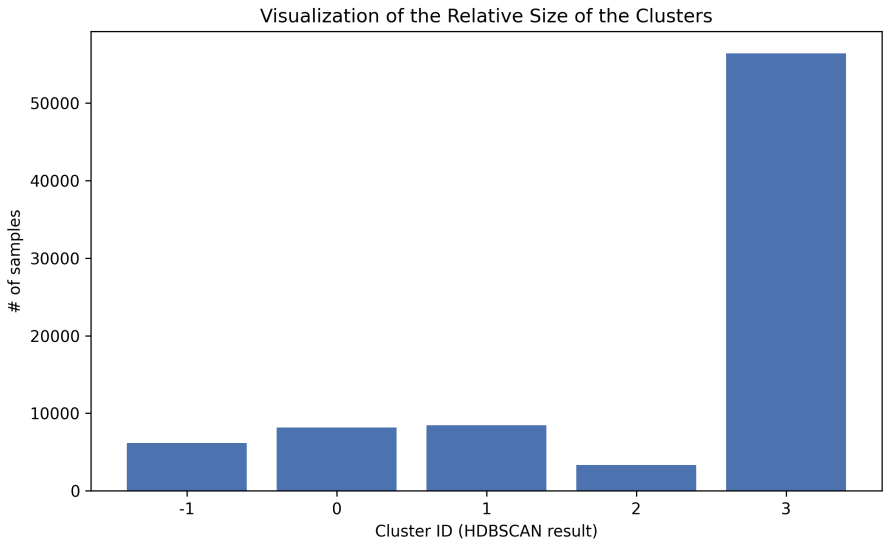


Figure 5.8: Visualization of the relative size of the clusters

space). It aggregates approximately 13% of the samples from both attack domains, thereby quantifying the intrinsic feature overlap and cross-class ambiguity between these two distinct attack vectors.

3. **Cluster 3**: This cluster contains the substantial majority of both spoofing and replay attacks (encompassing 88% of spoofing and 99% of replay instances). This high concentration strongly indicates that, under our current feature extraction framework, the anomalous manifestations of these two attack types are statistically similar, projecting onto indistinguishable anomaly manifolds.

According to the cluster-wise analysis, under our current feature extraction framework, the anomalous patterns associated with DoS and fuzzing attacks can be effectively distinguished from other threat categories. Conversely, the anomalous profiles of replay and spoofing attacks exhibit significant resemblance. This finding motivates us to improve feature engineering by designing specialized features that explicitly amplify the inter-class discriminability between replay and spoofing attacks. However, as indicated by our experimental outcomes, the inherent similarity between these two attacks makes them intrinsically challenging to decouple.

In summary, leveraging the optimal clustering result, we revealed the distinct characteristics of each attack type, evaluated the capacity of our feature engineering pipeline, and established a goal for its subsequent optimization.

6

EGL Methods

In this chapter, we focus on Explanation-Guided Learning (EGL) methods that improve the explainability of DL-based IDS of CAN based on the explanation results of the aforementioned XAI methods.

6.1 Right for the Right Reasons (RRR)

6.1.1 Analysis of explanation alignment induced by RRR fine-tuning

Based on the explanation results in Section 5.1, we identified a shortcut learned by the MLP model trained on the CAN-MIRGU dataset when handling DoS attacks. As shown in Figure 6.5, among the top-10 most influential features, L500_mean_pent, payload_6, and L100_mean_pent exhibit high importance. This behavior is expected for the CAN-MIRGU dataset, where all DoS samples contain all 0xFF payloads; consequently, payload values and their associated entropy become highly discriminative, since a DoS injection in this dataset causes payload entropy to drop dramatically. However, in real-world scenarios, DoS injections may carry random payloads, which do not necessarily cause noticeable deviations in payload statistics. Therefore, it is important to guide the model toward inherently robust indicators, including weights and jitter-related features, to improve its generalization capability against DoS attacks.

In the first experiment, we fine-tune the MLP model trained on the CAN-MIRGU dataset using the logit-wise RRR loss in Equation 2.10, targeting the DoS class. We then compare the global explanations of DoS samples before and after fine-tuning against domain-specific prior knowledge. Note that RRR directly supervises the vanilla-gradient explanation; however, vanilla gradients generally provide unsatisfactory attribution quality. Therefore, we additionally include SHAP attributions using the Gradient Explainer implementation to obtain more reliable feature-importance estimates.

Although our objective is to improve the explanatory rationale for the DoS class, we still train and evaluate the model on the full dataset. This is because the logit-wise RRR loss supervises the specific logit corresponding to the DoS class for all inputs,

and its influence may propagate across classes. Consequently, it is necessary to assess the overall predictive performance for all classes to ensure that enforcing explanation constraints for DoS does not inadvertently degrade accuracy on other attack types.

In terms of hyperparameters, we consider the supervision mask and the weight of the RRR loss. For the mask $\boldsymbol{A}$, we set all weight- and jitter-related features to 0 and all remaining features to 1, meaning that the RRR penalty suppresses attributions on non-weight and non-jitter features. For the weight of the RRR loss, we choose $\lambda = 0.1$ in the experiment, and the result shows that fine-tuning with this weight does not degrade the prediction performance, and meanwhile, shows a noticeable supervision strength.

Figure 6.1 shows the dynamics of the total loss (prediction loss plus RRR loss) and the validation accuracy during fine-tuning with the RRR objective applied to the DoS logit. The validation accuracy remains stable at approximately 99%, indicating that the model's predictive performance was not degraded by the introduction of the explanation-level supervision. Meanwhile, the total loss decreased steadily and plateaued around epoch 20, suggesting that the fine-tuning process converged smoothly.

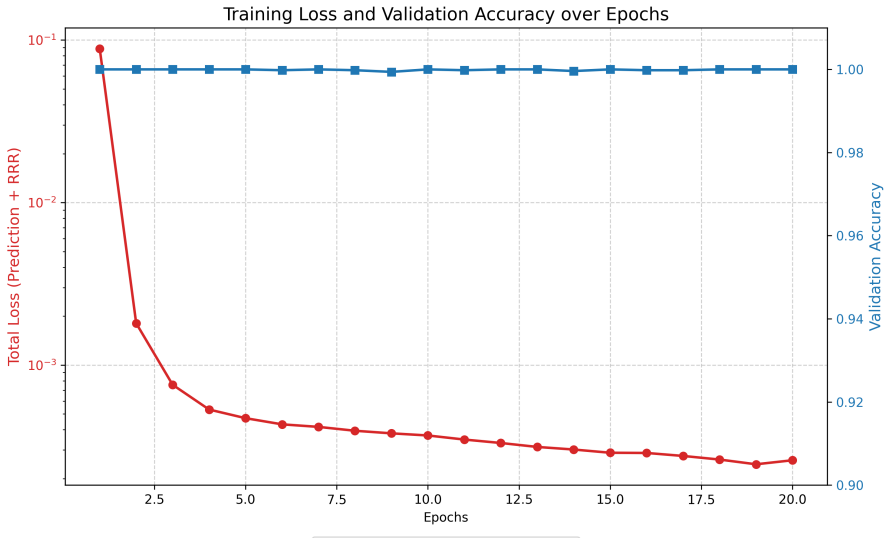


Figure 6.1: Training dynamics during fine-tuning with RRR loss for the DoS logit

Figure 6.2 presents the top-10 normalized global feature importance scores for DoS attack classification, comparing the model's behavior before and after RRR supervision.

Under the original model, SHAP analysis reveals that among the top 10 most influential features, several are spurious attributes unrelated to the characteristics of DoS attacks (i.e., non-weight and non-jitter features). Specifically, features such as L500_std_pent, payload_6, L1_std_pent, and L100_mean_pent account for a combined 21% of the total feature importance.

Following RRR supervision, the importance of such irrelevant features, including

L1_std_ham, payload_4, and L500_std_pent, drops to 12%, representing a 43% reduction in the model's reliance on unintended shortcuts.

In contrast, explanations derived from vanilla gradients yield a cumulative importance of approximately 53% for these undesired features, a value that remains largely unchanged after supervision. This discrepancy is expected, as vanilla gradients are susceptible to local noise and saturation, making them unstable for reliable attribution [43, 66], even when the supervision loss explicitly targets the gradient space.

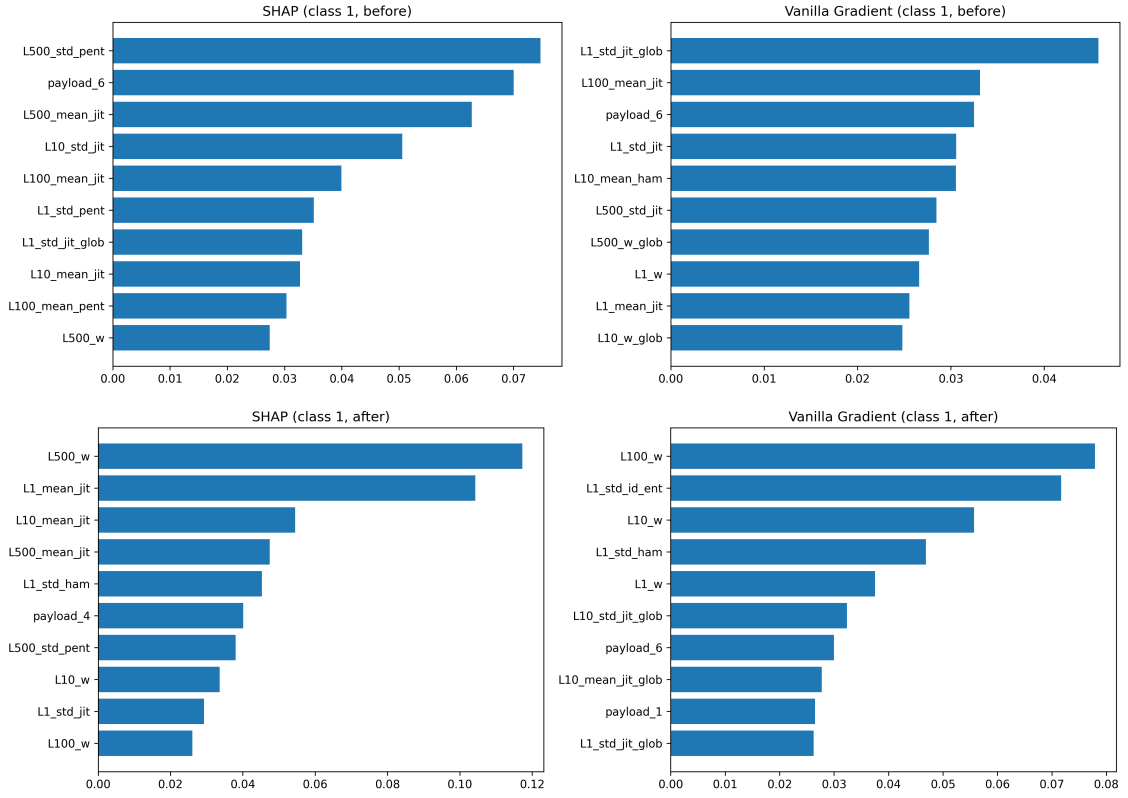


Figure 6.2: Top-10 normalized global feature attribution of the DoS attack before and after RRR supervision

In summary, our results demonstrate that the MLP model trained on the CAN-MIRGU dataset relies on some shortcut features when detecting DoS attacks, and that applying RRR supervision effectively mitigates this shortcut behavior in terms of SHAP explanation.

6.1.2 Generalization performance after RRR fine-tuning

The MLP model for the CT&T dataset was trained on `set_01/train_01`, a subset containing only zero-payload DoS attack samples. However, the test file `set_01/test_02_unknown_vehicle_known_attack/DoS-3.csv`, representing unknown vehicles with known attacks, includes samples with non-zero payloads. Under these conditions, if the model relies heavily on payload-related features for DoS detection, its generalization to such test sets will degrade. As shown in Table 6.1, the

MLP model trained solely on zero-payload DoS attack samples yields a 0% detection rate on this specific test set with non-zero-payload DoS attack samples, thereby validating our hypothesis regarding shortcuts for the model.

Table 6.1: Test results on non-zero-payload DoS attack data for the MLP model trained exclusively on zero-payload DoS samples before RRR supervision

Class	Precision	Recall	F1-score	Support
0 (Benign)	0.8568	1.0000	0.9229	943319
1 (DoS)	0.0000	0.0000	0.0000	157652
Accuracy			0.8568	1100971
Macro avg	0.4284	0.5000	0.4614	1100971
Weighted avg	0.7341	0.8568	0.7907	1100971

To address this issue, we fine-tuned the model on the training set containing only zero-payload DoS samples, while applying RRR supervision to suppress attributions assigned to payload-related features. Figure 6.3 shows the dynamics of validation accuracy and total loss (prediction loss and supervision loss) during fine-tuning. The validation accuracy remains stable at approximately 99%, indicating that no performance degrade happened on the training data (DoS attack only), and meanwhile, the total loss decreased steadily and plateaued around epoch 30, suggesting convergence of the fine-tuning.

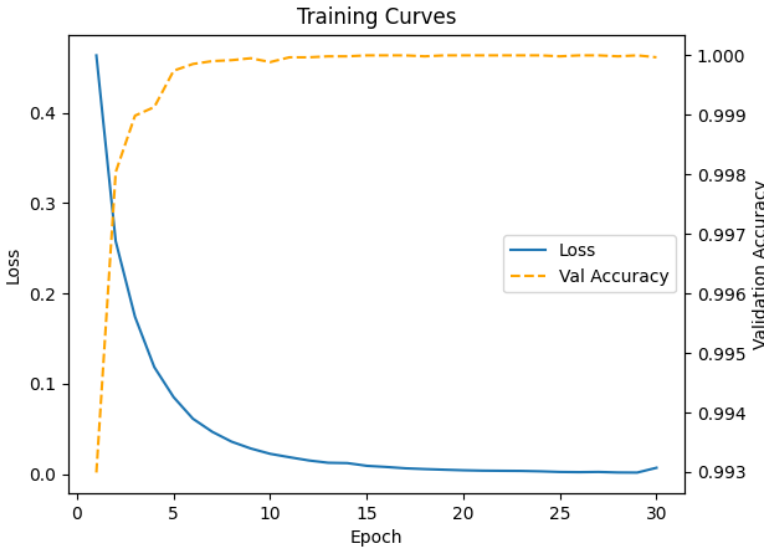


Figure 6.3: Training dynamics during fine-tuning with RRR loss for the MLP of the CT&T dataset

Table 6.2 presents the performance of the MLP model after fine-tuning with RRR supervision on the training set with only zero-payload DoS attack samples. It is evident that the model gains the capability to detect non-zero payload DoS attacks after supervision, achieving an F1-score of 0.61. While the F1-score for benign data

slightly decreased from 0.92 to 0.88, and the overall accuracy dropped from 0.85 to 0.82, these trade-offs remain within an acceptable range. Notably, both macro and weighted averages for precision, recall, and F1-score show overall improvement, except for the weighted average recall, which decreased by around 3%.

Table 6.2: Test results on non-zero-payload DoS attack data for the MLP model trained exclusively on zero-payload DoS samples after RRR supervision

Class	Precision	Recall	F1-score	Support
0 (Benign)	1.0000	0.7917	0.8838	943319
1 (DoS)	0.4452	1.0000	0.6161	157652
Accuracy			0.8216	1100971
Macro avg	0.7226	0.8959	0.7499	1100971
Weighted avg	0.9206	0.8216	0.8454	1100971

The RRR fine-tuning targeted at the DoS attack also affects the detection accuracy of other classes. Table 6.3 compares the model's performance on the original test dataset (known vehicles and known attacks in set 01) before and after DoS-targeted RRR fine-tuning. Clearly, the overall detection performance degrades slightly. This outcome is as expected, since the model trained on the can-train-and-test dataset is a binary classifier (attack versus benign). Unlike the multi-class model trained on CAN-MIRGU, where RRR supervision constrains only the logit of a specific target class, applying RRR to a binary classifier directly perturbs the sole decision boundary governing both classes. As a result, the supervision intended for the DoS class inevitably influences the classifier's decisions for all classes, even when such global effects are undesirable. However, in our case, the performance degradation remains well controlled, with all metrics decreasing by only a single-digit percentage.

Table 6.3: Performance comparison on the original test dataset (including all attacks and benign samples from known vehicles and known attacks in set 01) before and after RRR fine-tuning specifically targeted at the DoS attack

Class	**Metric**	**Before RRR**	**After RRR**	**Δ**
Benign	Precision	0.9993	0.9986	-0.0007
	Recall	0.9996	0.9994	-0.0002
	F1-score	0.9995	0.9990	-0.0005
Attacks	Precision	0.9675	0.9420	-0.0255
	Recall	0.9412	0.8756	-0.0656
	F1-score	0.9542	0.9076	-0.0466
Overall	Accuracy	0.9990	0.9980	-0.0010
	Macro Avg F1	0.9768	0.9533	-0.0235
	Weighted Avg F1	0.9990	0.9980	-0.0010

In conclusion, these findings indicate that explanation supervision enhances the model's generalization ability on the DoS attack, while causing minor performance

degradation on the other classes. Notably, this approach enables an MLP model, despite being trained solely on zero-payload DoS samples, to successfully identify non-zero payload DoS attacks during testing, thereby overcoming the constraints of distribution of the training data.

6.2 Attribution Priors

6.2.1 Experiment setup

In this part, we fine-tune the MLP model trained on the CAN-MIRGU dataset using the Gini coefficient sparsity regularization term, and compare the explanation sparsity based on expected gradients before and after fine-tuning.

Prior to model fine-tuning, we establish a baseline by recording the initial Expected Gradients (EG) feature attributions and their corresponding Gini coefficients. Throughout the fine-tuning process, we monitor the interplay between validation accuracy and the evolution of the Gini coefficient. Post-training, we evaluate the updated attributions and coefficients, comparing them against the baseline to demonstrate the effect of the attribution prior.

Additionally, there are several important hyper-parameters to be considered during this procedure:

1. Number of baseline samples for EG (k). As Equation 2.12 shows, EG is the expected value of IG over the baseline $x' \sim D$. Iterating every baseline in the dataset D is computationally infeasible. Therefore, Monte Carlo Estimation is adopted, using k randomly sampled datapoints from the dataset D to approximate the expected value, i.e., $E_{x' \sim D}[f(x')] \approx \frac{1}{k} \sum_{i=1}^{k} f(x'_i)$. The selection of k presents a critical trade-off: a small k ensures computational efficiency but yields noisy and unstable attribution results. Conversely, a large k improves estimation stability at the cost of significantly higher computational overhead. In this experiment, we selected $k = 20$, which is computationally efficient enough for our hardware, and the stability of Gini coefficient during the training process is acceptable.

2. Weight of regularization term (λ). λ balances the trade-off between prediction accuracy and the explanation regularization, which is sparsity here. In this experiment, we selected $\lambda = 0.1$. Experimental results demonstrate that this parameter setup maintains validation accuracy, avoiding any significant drop-off during the training phase.

3. The number of epochs and the learning rate for training. Because we fine-tune a pretrained model, and a pretrained model already contains useful representations, using a small learning rate, which is 10^{-4} in our case, prevents overwriting these learned features too aggressively and allows fine-tuning to make controlled, incremental updates. The number of epochs for training depends

on the evolution of validation accuracy and Gini coefficient. If validation accuracy drops dramatically, or the Gini coefficient reaches a plateau, the training process should be terminated.

Furthermore, during the training process, we have to compute feature attribution using EG for each backpropagation, which is computationally expensive. To improve efficiency, we also tried to use Vanilla Gradient Attribution, i.e., $\phi_i(x) = \frac{\partial f(x)}{\partial x_i}$, during training. However, we still use EG to measure the feature attribution performance before and after training. Due to the discrepancy between those two feature attribution methods, the explanation regularization performance using vanilla gradient should be worse. However, if the degradation is acceptable and the efficiency improvement is significant, it can still be a good alternative.

6.2.2 Experimental results

6.2.2.1 Performance of Gini sparse attribution

In the first experiment, we fine-tuned the MLP model trained on CAN-MIRGU with Gini coefficient as the explanation sparsity regularization term.

Figure 6.4 shows the dynamics of validation accuracy and Gini coefficient during the training process. The left subplot shows that, throughout the process, both validation accuracy and global Gini coefficient steadily increased. At around 80^{th} epoch, the global Gini coefficient reached a plateau at around 0.8, and thus we terminated the training process at 100 epochs.

Notably, at the very beginning of fine-tuning, the validation accuracy is much lower than the test accuracy of the original model. The sudden drop in validation accuracy after reloading a pretrained model typically occurs because the optimizer state (in our case, Adam's first- and second-moment estimates) is not restored. As a result, the initial updates become overly aggressive and temporarily disrupt the pretrained weights. After several iterations, the optimizer statistics stabilize and the model performance returns to its original level. Additionally, in the first epoch, the regularization term exerts a strong pressure on the model to suppress secondary features to increase sparsity. This sudden shift in feature weighting disrupts the established decision boundaries, leading to a temporary drop in accuracy.

The right subplot of Figure 6.4 shows the per-class Gini coefficient over epochs. The Gini coefficient for all classes increased significantly, except for class 2 (fuzzing attack).

Figure 6.5 shows the top-10 important features by EG for each attack before and after fine-tuning. Note that the EG feature importance scores are normalized such that their total sum equals 1.

According to Figure 6.5, all classes except for class 2 (fuzzing attack) reached a sparse explanation. For example, for class 1 (DoS attack), in the left top plot (C1 BEFORE), the importance values are relatively uniform, with the most significant

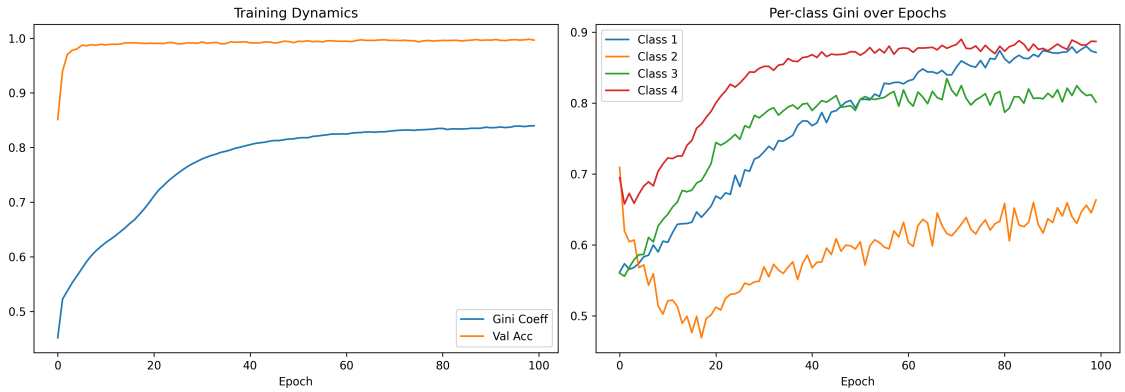


Figure 6.4: Training dynamics during fine-tuning with Gini coefficient as the attribution prior for explanation sparsity

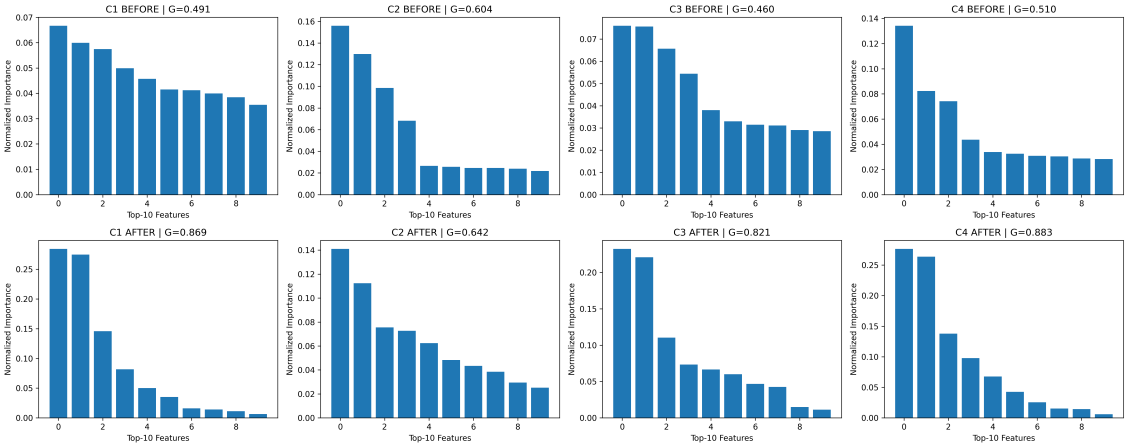


Figure 6.5: Normalized EG feature attribution scores for top-10 features for each attack before and after fine-tuning

feature holding a normalized importance of less than 0.07. In the left bottom plot (C1 AFTER), the importance is heavily concentrated in the first two features, which now reach values exceeding 0.25. Similar changes happened for class 3 (spoofing attack) and class 4 (replay attack).

Gini coefficients shown in Table 6.4 provide consistent implications for the changes of explanation sparsity. For class 1, 3, and 4, the Gini coefficient improved significantly. For class 2 (fuzzing attack), it almost remained the same.

Table 6.4: Per-class Gini coefficients before and after fine-tuning

	C1 (DoS)	**C2 (Fuzzing)**	**C3 (Spoofing)**	**C4 (Replay)**
Before	0.491	0.604	0.460	0.510
After	0.896	0.642	0.821	0.883

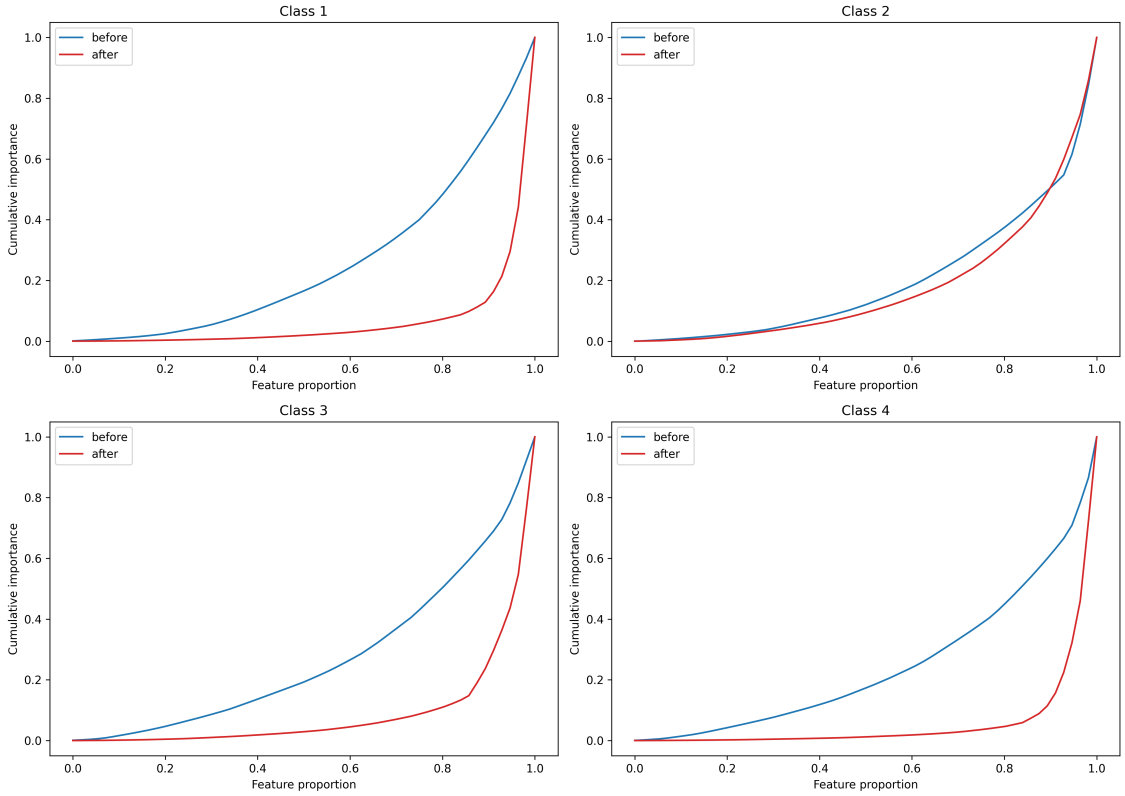


Figure 6.6: Cumulative feature importance before and after fine-tuning

Furthermore, Figure 6.6 shows the cumulative importance of sorted features. As we can see, for all attack classes except for class 2 (fuzzing attack), sparse attribution prior more effectively concentrates importance in the top few features.

In summary, fine-tuning the MLP model trained on the CAN-MIRGU dataset using Gini-sparse attributions enhances the sparsity of explanations for DoS, spoofing, and replay attacks, while preserving the model's high predictive accuracy.

6.2.2.2 Vanilla gradients for training efficiency

In the second experiment, we used vanilla gradient attribution during the training process, while still using EG for measuring the performance.

Table 6.5 shows a performance comparison between Expected Gradients (EG) and Vanilla Gradients (Grad) when utilized as feature attribution priors during the training process. While the model consistently maintained high validation accuracy for both cases, using Vanilla Gradients as the feature attribution prior significantly improved computational efficiency at the cost of explanation sparsity measured by global Gini coefficients.

As evidenced by the results, the Gini coefficient for vanilla gradients plateaued at approximately 0.65, significantly lower than the 0.80 achieved by EG. This suggests that vanilla gradients have a lower performance ceiling when used as a regularization prior.

However, when a moderate sparsity level (Gini $\approx$ 0.65) is sufficient, vanilla gradients offer a substantial efficiency advantage, requiring only 17 seconds compared to 5 minutes and 30 seconds for EG. Consequently, for applications with lower sparsity requirements, training with vanilla gradients proves to be a significantly more computationally efficient alternative.

Table 6.5: Comparison of training attributes (EG vs. Grad) across different epochs

train_attr	# of epochs	Gini_before	Gini_after	val_acc	time
EG	20	0.485	0.671	0.991	5m30s
EG	40	0.494	0.751	0.993	11m23s
EG	60	0.493	0.789	0.993	17m03s
EG	100	0.516	0.803	0.996	29m36s
Grad	20	0.476	0.609	0.990	7s
Grad	50	0.478	0.657	0.990	17s
Grad	100	0.488	0.610	0.994	32s
Grad	150	0.494	0.595	0.997	48s

7

Conclusion

7.1 Conclusion and Discussion

In this thesis, we constructed a feature engineering and deep learning analysis pipeline for CAN intrusion detection based on the CAN-MIRGU and CT&T datasets. We used the raw CAN fields together with explicitly extracted temporal statistical features as inputs to an MLP model, applied several XAI methods to analyze the basis of the model decisions under different attacks and test conditions, and utilized several EGL methods to improve the models' explainability based on the XAI results.

7.1.1 Feature engineering and MLP-based analysis

The experimental results show that extracting features explicitly before using a relatively simple model is a suitable design for explainability analysis. Through features such as weight, jitter, Hamming distance, payload entropy, and CAN ID entropy, the model decisions can be related to CAN traffic properties such as changes in message frequency, timing regularity, and payload distribution. For example, in several DoS, spoofing, and replay scenarios, the model pays attention to weight or jitter features within short time windows. This is consistent with the attack mechanism, since injected messages disturb the normal transmission pattern. For the purpose of this thesis, the advantage of the MLP is not only that it can perform classification, but also that its inputs and decision basis can be more directly linked to domain meanings through XAI.

At the same time, explicit feature engineering also limits which attack behaviors the model can capture. The global CAN ID entropy feature was designed for fuzzing attacks, but its aggregated statistical form is not sensitive enough to local message changes. Therefore, it does not consistently become an important feature for the model. In the RPM spoofing case, the model relies on Hamming distance within a short time window. However, this feature changes clearly only after several consecutive attack messages have appeared, which leads to missed detections at the beginning of the attack. This shows that explicit features improve the readability of explanations, but their window sizes, updating methods, and local or global scopes still need to be designed according to specific attack mechanisms.

7.1.2 Autoencoder and AE-p-value based analysis

As a complementary approach to the supervised MLP classifiers, we introduce an autoencoder trained in an unsupervised manner on the CAN-MIRGU dataset and investigate a model-specific explanation method based on AE-p-values.

First, the feature attribution results provide insights into both the model's decision-making logic and the characteristics of the dataset. For DoS attacks, the model places substantial emphasis on payload-related features. While this is effective for the CAN-MIRGU dataset, which contains only zero-payload DoS attack samples, such behavior is undesirable for generalization to non-zero-payload DoS scenarios. In the case of fuzzing attacks, the model primarily focuses on payload entropy and ID entropy, which is consistent with prior knowledge regarding the nature of fuzzing behavior. Further analysis reveals that the model mainly learns entropy decreases because the CAN-MIRGU dataset contains fuzzing attack samples with fixed payloads. Nevertheless, the model is still expected to detect fuzzing attacks with random payloads that cause entropy increases, since both increases and decreases in entropy represent deviations from normal CAN traffic patterns and can therefore be identified by the autoencoder.

Second, we investigate attack clustering using AE-p-values. The results show that clustering attacks in the representation space defined by p-values and their variants achieves higher clustering quality than clustering directly in the raw input feature space. Moreover, analysis of the best clustering results provides additional insight into the distribution of attacks within the explainability space. In particular, DoS and fuzzing attacks can be clearly separated from other samples, with only limited overlap between them. In contrast, replay and spoofing attacks remain heavily mixed in the p-value space, motivating future improvements in feature engineering to better distinguish between these two attack types.

7.1.3 Dataset bias, shortcut learning, and generalization

Another finding is that high classification performance alone does not prove that a model has learned robust attack patterns. The MLP achieves high F1-scores on CAN-MIRGU, but the XAI results show that the model relies on temporal statistical features, payload entropy, and specific payload bytes for some classes. These features are strongly discriminative in the current dataset, but they may also be related to the fixed construction of some attack samples. A comparison with the DoS results on CT&T shows that, when the attack payload is no longer fixed, the model relies more on message weight within short time windows. This feature is closer to the basic mechanism of DoS attacks, namely high-frequency injection. Therefore, shortcut learning is not determined only by the model architecture or the training procedure. The design of payload values, injection frequencies, and target IDs in the dataset also directly affects the decision logic learned by the model.

However, attention to specific payload bytes does not always mean that the model uses an unreasonable shortcut. In CAN payloads, particular bytes and their values

may represent real vehicle states, such as speed, engine speed, or gear position. For some spoofing attacks, payload features may therefore have a valid domain meaning. The important distinction is whether the model uses these bytes because they represent the control meaning changed by the attack, or because a specific attack script repeatedly uses a fixed value pattern in the current dataset. The former may be a useful detection basis, while the latter may fail when the attack implementation changes. Therefore, payload features should neither be simply removed nor directly accepted as reliable evidence without further validation. Their validity should be judged together with attack semantics and performance in different scenarios.

The results on CT&T further show that model generalization depends strongly on the match between the training distribution and the test environment. The model still has some detection ability for known attacks on a known vehicle, but its performance decreases clearly for unknown attacks, and it fails to detect attacks from unknown vehicles. However, not all unknown attacks fail. Some unseen spoofing attacks can still be detected because their injection behavior is similar to that of attacks in the training set. In contrast, fuzzing attacks whose statistical patterns are not represented during training cannot be identified. This suggests that the detection of a new attack depends more on whether similar behavior patterns have been learned than on whether the attack category has appeared in the training data. For practical deployment, cross-vehicle validation, calibration for the target environment, and transfer training when necessary should be considered.

These results also show that shortcut learning is not simply present or absent, and it does not always immediately lead to incorrect predictions. In some spoofing scenarios, the model combines message weight with payload-related features and achieves good performance. However, in the interval spoofing cases, where the injection behavior is similar but the payload distributions are different, a similar dependency causes one attack to be detected successfully while the other is completely missed. Therefore, the role of XAI is not simply to label a feature as correct or incorrect. It is to examine whether the features used by the model have reasonable meanings, and whether the same decision basis remains reliable when the payload, attack implementation, or vehicle environment changes.

7.1.4 Roles and limitations of XAI Methods

Different XAI methods are also suitable for different purposes in this process. SHAP is more suitable for detailed offline analysis, but it requires more computation time. Integrated Gradients has better efficiency and stability, and is suitable for observing general trends over a larger number of samples. LIME is more suitable for analyzing individual misclassified samples. In addition, when the model predicts almost all inputs as benign in cross-vehicle tests, the explanations themselves may become degenerated and may no longer provide reliable behavior analysis. Therefore, XAI results should be interpreted together with model performance, test distribution, and the background of each attack, rather than being treated alone as evidence of model reliability.

7.1.5 EGL methods to improve explainability

We apply explanation-guided learning (EGL) methods to enhance the explainability and robustness of the target models.

First, we employ Right for the Right Reasons (RRR) as an explanation-supervision approach to align the models' decision-making rationale with domain knowledge. In particular, the models are penalized for relying on non-jitter and non-weight features when detecting DoS attacks, encouraging them to focus on more semantically meaningful features and thereby improving their generalization capability across different payload distributions.

For the MLP model trained on the CAN-MIRGU dataset, RRR supervision reduces the model's reliance on unintended shortcut features by 43%. For the MLP model trained on the CT&T dataset, whose training set contains only zero-payload DoS attack samples, the original model achieves an F1-score of 0 on non-zero-payload DoS attacks in a dedicated test set. However, after applying RRR supervision during training, the model acquires the ability to generalize to non-zero-payload DoS attacks, achieving an F1-score of 0.61 on the test set, albeit with a slight degradation in overall classification performance across all classes.

Second, we employ the Gini coefficient as an attribution prior for explanation regularization to improve the sparsity of explanations generated by the target MLP model trained on the CAN-MIRGU dataset. The results show a substantial improvement in overall explanation sparsity without any degradation in detection performance.

For individual attack classes, the per-class Gini coefficient improves by 82%, 78%, and 73% for DoS, spoofing, and replay attacks, respectively, while remaining nearly unchanged for fuzzing attacks. These findings indicate that incorporating the Gini coefficient as an attribution prior in deep learning–based CAN intrusion detection systems can effectively enhance explanation sparsity while preserving predictive performance.

7.1.6 Implications for CAN IDS evaluation

Based on the above analysis, future evaluation of CAN intrusion detection systems (IDSs) should follow a comprehensive process that combines predictive performance assessment, explanation analysis, and explainability-oriented model improvement.

First, standard evaluation metrics such as accuracy, recall, and F1-score can be used to verify the model's basic detection capability on the target test set. Subsequently, explainable artificial intelligence (XAI) methods can be employed to determine whether the model primarily relies on features that are semantically related to attack mechanisms or on dataset-specific artifacts and fixed patterns.

The identified decision rationale should then be further evaluated under distribution shifts, including variations in payload implementations, unseen attack types,

and cross-vehicle scenarios. The issues revealed through this analysis can guide improvements in dataset coverage, feature engineering, model architecture, and training strategies. For multi-class attack recognition tasks, separating attack detection from attack-type classification may also be beneficial, as this can reduce the likelihood that the model bases its decisions on incidental differences between attack categories.

Furthermore, insights obtained from XAI analysis can be leveraged to apply explanation-guided learning (EGL) methods that improve model explainability. Such approaches can align the model's decision-making rationale with domain-specific prior knowledge while also enhancing desirable explanation properties, such as sparsity.

7.2 Ethics and Sustainability

This thesis follows academic ethics in terms of data privacy and responsible data use. Although the public datasets used in this thesis do not contain real driver identities, CAN bus data may still indirectly reflect vehicle operating states and driving patterns. Therefore, the scope of this thesis is limited to deep-learning-based intrusion detection systems (DL-based IDS) and explainable artificial intelligence (XAI) methods. It does not try to recover personal information or track specific vehicles. In future in-vehicle deployment, IDS data collection should follow the principle of data minimization. Whenever possible, data should be processed locally, and sensitive vehicle behavior data should not be uploaded to the cloud unless strictly necessary.

In-vehicle networks are typical safety-critical systems, and IDS decisions may affect both vehicle safety and passenger safety. A false negative prediction may allow a real attack to remain undetected, while a false positive prediction may cause unnecessary responses that disturb normal driving. The ethical value of this thesis lies in the fact that it does not propose a production-ready IDS. Instead, it uses XAI methods to examine model decision logic and help engineers identify whether a model relies on shortcut learning, such as depending on specific payload bytes rather than the essential characteristics of an attack. This reduces the risk of applying black-box models directly in safety-critical contexts. The experimental results in this thesis should therefore be understood only as research analysis. Any system for real deployment would require more strict validation across vehicle models, real driving conditions, and hardware constraints.

This thesis also involves dual-use risk in cybersecurity. Explaining the decision logic of CAN IDS can help defenders identify system weaknesses and improve model generalization. However, at the same time, such information could also be used by attackers to design more sophisticated strategies. For this reason, when discussing attack types, dataset properties, and model explanations, this thesis focuses on improving the transparency and reliability of systems, rather than providing attack guidance. The model weaknesses and shortcuts identified in this thesis are treated as security risks and directions for future improvement, not as attack strategies.

Regarding sustainability and computational cost, electronic control units in real vehicles have limited resources. Complex deep learning models or real-time XAI methods may be unsuitable for long-term online operation. For this reason, this thesis uses relatively lightweight model structures that are more compatible with resource-constrained in-vehicle environments. Although XAI methods and explanation-guided learning (EGL) can be computationally expensive during training and explanation, these steps are mainly performed offline during development and debugging. The sustainability rationale of this thesis is to improve model reliability through offline analysis, so that simpler, lower-cost models can be used during actual deployment.

From a broader sustainability perspective, an in-vehicle IDS that can detect attacks early may help prevent physical vehicle damage, traffic accidents, and unnecessary maintenance, thereby reducing material and resource waste. However, these potential safety and environmental benefits must be weighed against the additional hardware requirements, model training costs, and operational energy consumption introduced by such a system.

7.3 Future Work

Several paths can be explored as a continuation of our work.

First, different model architectures and Explainable AI (XAI) techniques can be systematically evaluated. In this project, we primarily focus on multilayer perceptrons (MLPs), which offer a simple architecture while achieving competitive accuracy in CAN intrusion detection. However, as CAN attack strategies continue to evolve, more expressive models, such as transformer-based architectures, may be required to maintain adequate detection performance. As model complexity increases, obtaining reliable explanations becomes more challenging. Consequently, additional XAI methods tailored to these advanced models, such as attention-based saliency for transformers, should also be considered.

Second, more advanced feature engineering strategies for CAN intrusion detection systems (IDSs) can also be explored. In our current work, the extracted features mainly consist of general temporal statistics, which are relatively generic and not specifically designed for the unique characteristics of CAN intrusion datasets. Although these features can capture basic traffic dynamics, they may fail to fully represent the protocol-specific behaviors and attack patterns inherent in in-vehicle CAN communications. Future work could therefore investigate domain-specific feature representations tailored to CAN networks. For example, features related to payload bit-level transitions, arbitration behavior, and ECU communication dependencies may provide more discriminative information for identifying sophisticated attacks.

Third, self-explanatory IDS models can also be explored. Structures like Graph Neural Networks (GNNs) naturally provide relational reasoning capabilities that align well with the graph-structured dependencies in network traffic, CAN communication, and system provenance data. By modeling interactions among entities such as

ECUs, CAN IDs, hosts, or processes, GNN-based IDS frameworks can simultaneously perform intrusion detection and identify the critical nodes, edges, or message flows contributing to the detection result. Other directions including prototype-based and concept-based models are also promising.

Bibliography

[1] Robert Bosch GmbH, Stuttgart, Germany. *CAN Specification Version 2.0*, 1991.

[2] Sampath Rajapaksha, Garikayi Madzudzo, Harsha Kalutarage, Andrei Petrovski, and M Omar Al-Kadri. CAN-MIRGU: A comprehensive CAN bus attack dataset from moving vehicles for intrusion detection system evaluation. In *Symposium on Vehicles Security and Privacy. Internet Society*, 2024.

[3] Syed Yusuf, Arshad Khan, and Riad Souissi. Vehicle-to-everything (V2X) in the autonomous vehicles domain: A technical review of communication, sensor, and AI technologies for road user safety. *Transportation Research Interdisciplinary Perspectives*, 23:100980, 2024.

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

[5] Aida Ben Chehida Douss, Ryma Abassi, and Damien Sauveron. State-of-the-art survey of in-vehicle protocols and automotive ethernet security and vulnerabilities. *Mathematical Biosciences and Engineering*, 20(9):17057–17095, 2023.

[6] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, and Stefan Savage. Experimental security analysis of a modern automobile. In *IEEE Symposium on Security and Privacy*, pages 447–462, 2010.

[7] Mehmet Bozdal, Mohammad Samie, Sohaib Aslam, and Ian Jennions. Evaluation of CAN bus security challenges. *Sensors*, 20(8):2364, 2020.

[8] Francesco Pascale, Ennio Andrea Adinolfi, Simone Coppola, and Emanuele Santonicola. Cybersecurity in automotive: An intrusion detection system in connected vehicles. *Electronics*, 10(15):1765, 2021.

[9] Pedro Garcia-Teodoro, Jesus Diaz-Verdejo, Gabriel Macia-Fernandez, and Enrique Vazquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*, 28(1-2):18–28, 2009.

[10] Giacinto Kim, Seungmin Lee, and Sehun Kim. Long short term memory recurrent neural network classifier for intrusion detection. *IEEE International Conference on Platform Technology and Service*, 2016.

[11] Ahmad Javaid, Quamar Niyaz, Weiqing Sun, and Mansoor Alam. A deep learning approach for network intrusion detection system. *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies*, 2016.

[12] Md Liakat Ali, Kutub Thakur, Suzanna Schmeelk, Joan Debello, and Denise Dragos. Deep learning vs. machine learning for intrusion detection in computer networks: A comparative study. *Applied Sciences*, 15(4):1903, 2025.

[13] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267:1–38, 2019.

[14] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information fusion*, 58:82–115, 2020.

[15] Yuyang Gao, Siyi Gu, Junji Jiang, Sungsoo Ray Hong, Dazhou Yu, and Liang Zhao. Going beyond XAI: A systematic survey for explanation-guided learning. *ACM Computing Surveys*, 56(7), 2024.

[16] Yisroel Mirsky, Tomer Doitshman, Yuval Werbeloff, Alon Nosrati, and Yuval Elovici. Kitsune: An ensemble of autoencoders for online network intrusion detection. In *Proceedings of the Network and Distributed System Security (NDSS) Symposium*, 2018.

[17] Brooke Lampe and Weizhi Meng. can-train-and-test: A curated CAN dataset for automotive intrusion detection. *Computers & Security*, 140:103777, 2024.

[18] Karl Henrik Johansson, Martin Törngren, and Lars Nielsen. Vehicle applications of controller area network. In *Handbook of networked and embedded control systems*, pages 741–765. Springer, 2005.

[19] Kyong-Tak Cho and Kang G. Shin. Fingerprinting electronic control units for vehicle intrusion detection. In *USENIX Security Symposium*, 2016.

[20] Adrian Taylor, Sylvain Leblanc, and Nathalie Japkowicz. Anomaly Detection in Automobile Control Network Data with Long Short-Term Memory Networks. In *2016 IEEE 3rd International Conference on Data Science and Advanced Analytics (DSAA)*, pages 130–139, Los Alamitos, CA, USA, October 2016. IEEE Computer Society.

[21] Sampath Rajapaksha, Harsha Kalutarage, M. Omar Al-Kadri, Andrei Petrovski, and Garikayi Madzudzo. Beyond vanilla: Improved autoencoder-based

ensemble in-vehicle intrusion detection system. *Journal of Information Security and Applications*, 77:103570, 2023.

[22] Miki E. Verma, Robert A. Bridges, Michael D. Iannacone, Samuel C. Hollifield, Pablo Moriano, Steven C. Hespeler, Bill Kay, and Frank L. Combs. A comprehensive guide to can ids data and introduction of the ROAD dataset. *PLOS ONE*, 19(1):1–32, 01 2024.

[23] Anna L. Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, 2016.

[24] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. Intrusion detection system: A comprehensive review. *Journal of network and computer applications*, 36(1):16–24, 2013.

[25] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE Symposium on Security and Privacy*, pages 305–316, 2010.

[26] Yang Xin, Lingshuang Kong, Zhi Liu, Yuling Chen, Yanmiao Li, Hongliang Zhu, Mingcheng Gao, Haixia Hou, and Chunhua Wang. Machine learning and deep learning methods for cybersecurity. *IEEE Access*, 6:35365–35381, 2018.

[27] Vincent Zibi Mohale and Ibidun Christiana Obagbuwa. A systematic review on the integration of explainable artificial intelligence in intrusion detection systems to enhancing transparency and interpretability in cybersecurity. *Frontiers in Artificial Intelligence*, Volume 8 - 2025, 2025.

[28] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

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

[30] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[31] Anshu Gangwar and Sandeep Sahu. A survey on anomaly and signature based intrusion detection system (IDS). *International Journal of Engineering Research and Applications*, 4(4):67–72, 2014.

[32] Vaishnavi Shanmugam, Roozbeh Razavi-Far, and Ehsan Hallaji. Addressing class imbalance in intrusion detection: A comprehensive evaluation of machine learning approaches. *Electronics*, 14(1):69, 2024.

[33] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[34] Youngrok Song, Sangwon Hyun, and Yun-Gyung Cheong. Analysis of autoencoders for network intrusion detection. *Sensors*, 21(13):4294, 2021.

[35] Subash Neupane, Jesse Ables, William Anderson, Sudip Mittal, Shahram Rahimi, Ioana Banicescu, and Maria Seale. Explainable intrusion detection systems (X-IDS): A survey of current methods, challenges, and opportunities. *IEEE Access*, 10:112392–112415, 2022.

[36] Amina Adadi and Mohammed Berrada. Peeking inside the black-box: A survey on explainable artificial intelligence (XAI). *IEEE Access*, 2018.

[37] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Comput. Surv.*, 51(5), August 2018.

[38] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144, 2016.

[39] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. Curran Associates, Inc., 2017.

[40] Lloyd S. Shapley. A value for n-person games. In Harold William Kuhn and Albert William Tucker, editors, *Contributions to the Theory of Games (AM-28), Volume II*, volume 28 of *Annals of Mathematics Studies*, pages 307–317. Princeton University Press, Princeton, NJ, 1953.

[41] Liat Antwarg, Ronnie Mindlin Miller, Bracha Shapira, and Lior Rokach. Explaining anomalies detected by autoencoders using Shapley additive explanations. *Expert Systems with Applications*, 186:115736, 2021.

[42] Maonan Wang, Kangfeng Zheng, Yanqing Yang, and Xiujuan Wang. An explainable machine learning framework for intrusion detection systems. *IEEE Access*, 8:73127–73141, 2020.

[43] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR, 2017.

[44] Maxime Lanvin, Pierre-François Gimenez, Yufei Han, Frédéric Majorczyk, Ludovic Mé, and Eric Totel. Towards understanding alerts raised by unsupervised network intrusion detection systems. In *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, pages 135–150, 2023.

[45] Arthur S Jacobs, Roman Beltiukov, Walter Willinger, Ronaldo A Ferreira, Arpit Gupta, and Lisandro Z Granville. AI/ML and network security: The

emperor has no clothes. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1537–1551, 2022.

[46] Andrew Slavin Ross, Michael C. Hughes, and Finale Doshi-Velez. Right for the right reasons: Training differentiable models by constraining their explanations. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 2662–2670, 2017.

[47] 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.

[48] Flora Amato, Luigi Coppolino, Francesco Mercaldo, Francesco Moscato, Roberto Nardone, and Antonella Santone. Can-bus attack detection with deep learning. *IEEE Transactions on Intelligent Transportation Systems*, 22(8):5081–5090, 2021.

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

[50] Muneeb Hassan Khan, Abdul Rehman Javed, Zafar Iqbal, Muhammad Asim, and Ali Ismail Awad. DivaCAN: Detecting in-vehicle intrusion attacks on a controller area network using ensemble learning. *Computers & Security*, 139:103712, 2024.

[51] Yan Meng, Jiachun Li, Fazhong Liu, Shaofeng Li, Haotian Hu, and Haojin Zhu. GB-IDS: An intrusion detection system for CAN bus based on graph analysis. In *2023 IEEE/CIC International Conference on Communications in China (ICCC)*, pages 1–6, 2023.

[52] Hyunsung Lee, Seong Hoon Jeong, and Huy Kang Kim. OTIDS: A novel intrusion detection system for in-vehicle network by using remote frame. In *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, volume 00, pages 57–5709, Aug 2017.

[53] Xue Zhou, Guihe Qin, Yanhua Liang, Jiaru Song, Wanning Liu, and Qingxin Liu. CGTS: graph transformer-based anomaly detection in controller area networks. *Cybersecurity*, 8(1):62, Aug 2025.

[54] Md Mahbub Hasan, Sanchita Ghose, and Krishna Chandra Roy. CAN-GraphiT: A graph-based IDS for CAN networks using transformer. *IEEE Access*, 13:130015–130028, 2025.

[55] Robert Frenken, Sidra Ghayour Bhatti, Hanqin Zhang, and Qadeer Ahmed. KD-GAT: Combining knowledge distillation and graph attention transformer for a controller area network intrusion detection system. In *2025 IEEE 28th*

International Conference on Intelligent Transportation Systems (ITSC), pages 3721–3726, 2025.

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

[57] Hareem Kibriya, Ayesha Siddiqa, Saad Alahmari, Wazir Zada Khan, Saad Nasser Altamimi, and Atta ur Rehman Khan. A hybrid deep learning and residual connection-based architecture for intrusion detection in autonomous vehicles. *PLOS ONE*, 21:1–24, 03 2026.

[58] Chathurika S. Wickramasinghe, Daniel L. Marino, Harindra S. Mavikumbure, Victor Cobilean, Timothy D. Pennington, Benny J. Varghese, Craig Rieger, and Milos Manic. RX-ADS: Interpretable anomaly detection using adversarial ML for electric vehicle CAN data. *IEEE Transactions on Intelligent Transportation Systems*, 24(12):14051–14063, 2023.

[59] Rakib Hossain Sajib, Md. Rokon Mia, Prodip Sarker, Abdullah Al Noman, and Md Arifur Rahman. CANGuard: A spatio-temporal CNN-GRU-Attention hybrid architecture for intrusion detection in in-vehicle CAN networks. pages 1010–1015, 12 2025.

[60] Sampath Rajapaksha, Harsha Kalutarage, M. Omar Al-Kadri, Andrei Petrovski, Garikayi Madzudzo, and Madeline Cheah. AI-Based intrusion detection systems for in-vehicle networks: A survey. *ACM Comput. Surv.*, 55(11), February 2023.

[61] Euclides Carlos Pinto Neto, Hamideh Taslimasa, Sajjad Dadkhah, Shahrear Iqbal, Pulei Xiong, Taufiq Rahman, and Ali A Ghorbani. CICIoV2024: Advancing realistic IDS approaches against DoS and spoofing attack in IoV CAN bus. *Internet of Things*, 26:101209, 2024.

[62] Yuan Zhang, Jiaru Song, Yongxiong Sun, Zhanheng Gao, Zhe Hu, and Minghui Sun. Federated two-stage transformer-based network for intrusion detection in non-IID data of controller area networks. *Cybersecurity*, 8, 05 2025.

[63] Alexander Warnecke, Daniel Arp, Christian Wressnegger, and Konrad Rieck. Evaluating explanation methods for deep learning in security. pages 158–174, 09 2020.

[64] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? In *International conference on database theory*, pages 217–235. Springer, 1999.

[65] Ricardo JGB Campello, Davoud Moulavi, Arthur Zimek, and Jörg Sander. Hierarchical density estimates for data clustering, visualization, and outlier detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 10(1):1–51, 2015.

[66] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. In *ICML Workshop on Visualization for Deep Learning*, 2017.

A

Appendix 1

A.1 Efficient AE-p-value computation with GPU acceleration

The authors of the AE-p-value paper [44] provide an implementation of AE p-value computation. However, its performance is limited because it relies solely on CPU computation, which is significantly slower than GPU-based parallelization for massive matrix computation. Therefore, we implemented the AE p-value computation using GPU.

In our implementation, we used CuPy for GPU acceleration when dealing with massive matrices. General steps are as follows:

1. Calculate the discrete empirical distribution $\hat{\mathbb{P}} \in \mathbb{R}^{M \times F}$, where M is the size of the training data, and F is the number of features. This is the forward propagation of the autoencoder, thus the GPU support is provided by TensorFlow. Notably, the distribution only needs to be calculated once for a single model, and different inputs can rely on the same empirical distribution.

2. Calculate the reconstruction error matrix $\mathbf{E} \in \mathbb{R}^{N \times F}$, where N is the size of all input data, and F is the number of features. Same as step 1, the GPU support is provided by TensorFlow.

3. Sort the $\hat{\mathbb{P}}$ matrix. Since we have to find the percentile of the input reconstruction error e_i within the normal data r_i, we sort every column of $\hat{\mathbb{P}}$, and then we can use binary search to find $\#\{r_i > e_i\}$.

4. Calculating the p-values: $p_i = \frac{\#\{r_i > e_i\}}{\#\{r_i\}}$. As stated in step 3, we can use binary search to find $\#\{r_i > e_i\}$, and $\#\{r_i\} = M$ is a constant.

The time and space complexity analysis is as follows:

1. Calculate $\hat{\mathbb{P}}$ and $\mathbf{E}$: $O(M + N)$ time complexity for M normal data vectors and N input data vectors. Consider batch size B, the space complexity in GPU memory is $O(B)$.

2. Sort the $\hat{\mathbb{P}}$ matrix: The time complexity takes $O(M \times \log M)$ for quick sorting a single feature dimension, and $O(F \times M \times \log M)$ for the whole $\hat{\mathbb{P}}$ matrix. Consider batch size B (number of features for a single sort operation), the space complexity in GPU memory is $O(B \times F)$.

3. Calculating the p-values: $p_i = \frac{\#\{r_i > e_i\}}{\#\{r_i\}}$: Finding $\#\{r_i > e_i\}$ for an input batch using binary search costs $O(N \times F \times \log M)$, and $\#\{r_i\} = M$ is a constant, thus the overall time complexity is $O(N \times F \times \log M)$. The space complexity in GPU memory is $O(B \times F + M \times F)$, where $B \times F$ is for the e_i batch, and $M \times F$ is for the whole r_i matrix.

In summary, we can split the process into 2 phases:

1. Forward propagation: calculating $\hat{\mathbb{P}}$ and $\mathbf{E}$, with time complexity of $O(M+N)$

2. p-value calculation: all other operations, with time complexity of $O((M + N) \times F \times \log M)$

For forward propagation, both implementations reply on the interfaces provided by TensorFlow, thus their performance is close to each other. However, for the p-value calculation, the difference is huge. The results of experiments for both the original CPU implementation and our GPU implementation in the p-value calculation phase are shown in Tab A.1.

Table A.1: Performance comparison for p-value computation

Implementation	Time for a batch (4096 samples)
CPU (original)	0.6ms * 4096 $\approx$ 2457ms
GPU (our implmentation)	$\approx$ 60ms

In summary, we implemented an efficient AE-p-values calculation script that utilizes GPUs, and compared to the original version, the improvement is significant.