LUFT-CAN: A lightweight unsupervised learning based intrusion detection system with frequency-time analysis for vehicular CAN bus

LUFT-CAN: A Lightweight Unsupervised Learning Based Intrusion Detection System with Frequency-Time Analysis for Vehicular CAN Bus

Yu Xin[a,b], Xiaohang Wang[a,b,e], Li Lu[a,b], Shuguo Zhuo[a,b], Yingtao Jiang[c], Amit Kumar Singh[d], Kui Ren[a,b], Mei Yang[c], Kaiwei Wu[a,b]

[a]*State Key Laboratory of Blockchain and Data Security, Zhejiang University,*
[b]*Hangzhou High-Tech Zone (Binjiang), Institute of Blockchain and Data Security ,*
[c]*University of Nevada Las Vegas,*
[d]*University of Essex,*
[e]*Corresponding author,*

Abstract

The Controller Area Network (CAN) bus is critical for data transmission among electronic control units (ECUs) in modern vehicles, necessitating robust intrusion detection systems (IDS) for security. However, existing IDS approaches have several limitations. For example, rule based IDS methods depend on proprietary protocol knowledge, while most machine learning approaches rely on supervised training using outdated or limited datasets, hindering their ability to detect emerging threats. Furthermore, deep learning-based IDS models often have high computational complexity, making them unsuitable for resource-constrained vehicular environments. To overcome these challenges, we propose LUFT-CAN, a novel, lightweight, unsupervised IDS that integrates frequency and time domain analysis of CAN traffic. By leveraging spectral characteristics of CAN ID sequences, LUFT-CAN effectively distinguishes between normal and anomalous traffic patterns. A tailored neural network architecture extracts these features, and the system is optimized via quantization-aware training for real-time inference on embedded systems. Experiments performed on datasets collected from modern vehicles, Tesla Model 3 2022 and LeapMotor C10 2024 as well as a public benchmark dataset demonstrate that LUFT-CAN achieves promising F1-scores of 97.1% and 96.7%, significantly outperforming previous approaches. We implemented the proposed IDS on a 2024 LeapMotor C10 test vehicle equipped with a Qualcomm 8295 MCU. The model's inference time is 14.27 seconds per 100,000 frames, demonstrating its effectiveness and efficiency for in-vehicle deployment.

Keywords: Unsupervised Learning Based IDS, Intrusion Detection

Systems, Controller Area Network

1. Introduction

The components and modules of connected and automated vehicles (CAVs) have become increasingly complex. The controller area network (CAN) bus, as the most widely utilized protocol, plays a vital role in serving as the standard interface for connecting a multitude of electronic control units (ECUs). CAN bus is crucial for the reliable and efficient communication within the vehicle's network. It serves as the communication backbone, facilitating the seamless exchange of data between various ECUs. CAN ensures reliable and efficient transmission of critical information, which is essential for the operation of Advanced Driver Assistance Systems (ADAS), telematics, and autonomous functionalities.

Despite several advantages of the CAN bus, the broadcast nature of CAN frames and the absence of security mechanisms, such as encryption and authentication, introduce significant security vulnerabilities. A potential adversary can exploit vulnerabilities through wireless networks to hijack an on-vehicle ECU and sniff CAN traffic. By reversing engineering the data, the attacker can decipher the semantic information of the CAN bus, gaining remote control over vehicle functions. This could lead to severe consequences, including the loss of valuables [1]. More seriously, a thief or attacker may physically access the vehicle and connect to the CAN bus, enabling direct control over doors, windows, or other components, thereby resulting in the risk of accident or vehicle theft [2]. In the mobile remote control of vehicles, attackers can utilize a femtocell device, functioning as a mini-cell tower in low-signal areas, to set up unauthenticated Telnet or https services on the femtocells, enabling remote access to the connected devices and the capability to send arbitrary CAN packets to the CAN bus of the vehicles [3].

To protect in-vehicle networks against cyber attacks, many CAN intrusion detection systems (IDS) have been proposed to monitor CAN traffic to detect malicious data frames, mitigating potential security risks. These methods fall into four categories: rule based IDS methods, traditional machine learning (ML) based IDS methods, supervised deep learning (DL) based IDS methods, and unsupervised deep learning based methods. The rule based IDS methods, e.g. in [4], [5] and [6], utilized clock skew, offset ratio, and bit values respectively, as the primary target to detect the anomalies. The traditional ML based IDS methods in [7], [8] and [9] employed OC-SVM, entropy analysis, and hybrid ML model etc., to classify messages as normal or anomalous. The methods in [10], [11] and [12] utilized supervised DL methods, such as convolutional neural network (CNN) or long short-term memory (LSTM) network, achieving a high detection accuracy. The methods in [13], [14] and [15] designed unsupervised autoencoders to detect unseen

Table 1: Comparison of the Car-Hacking dataset [17] and dataset extracted from Tesla Model 3 2022.

Feature	**Car-Hacking (2020)**	**Tesla Model 3 (2022)**
Identifier (ID)	✓	✓
ID Categories	37	270
Data Length(DLC)	✓	✓
Data Field Size	8 bytes	8 bytes
Control Field	Basic	Enhanced
Extended Identifier	×	✓
Precise Timestamp	×	✓
Driving Data	Minimal	Detailed (GPS, etc.)
ADAS Control Data	×	✓

attacks. Unfortunately, these above methods have the following limitations and challenges:

- Rule based IDS methods depend on database of CAN (DBC) files with the design of the CAN bus protocol, which are often unavailable to third-party IDS vendors. Additionally, rule based IDS methods also have high false positive rate, as they cannot detect previously unseen attacks.

- Most of the ML or DL based IDS methods use supervised methods, which cannot detect unseen anomalies because of limited training dataset. On the other hand, most existing unsupervised methods generally perform well on open-source datasets extracted from old cars before 2020, which tend to be simpler with typically only one or two features, as depicted in Table 1. The old cars have CAN messages with minimal driving data and no ADAS control data. In contrast, modern vehicles including Tesla Model 3 2022 and LeapMotor C10 2024 have more driving functions and specific ADAS, thus have more complicated CAN messages. Therefore, existing methods, such as [10], [14], face challenges in adapting to the evolving CAN traffic in modern automotive systems. When dealing with datasets with a greater number of features from modern vehicles, their performance significantly declines, indicating their unsuitability with modern vehicles, as illustrated in Fig. 1.

- An IDS must be highly computationally efficient in a vehicle, where the System-on-Chip (SoC) has limited computation capability and a very low CPU utilization is required for IDS. However, most of the existing

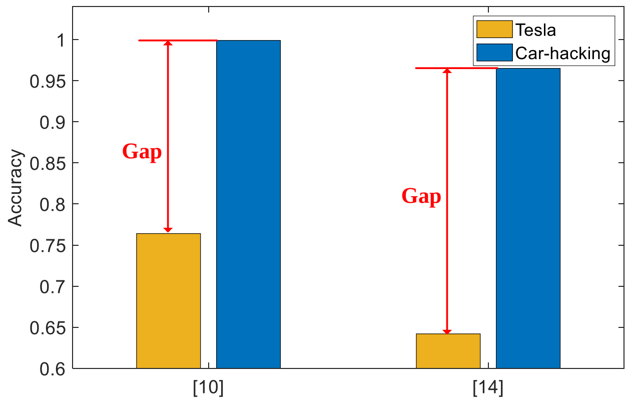


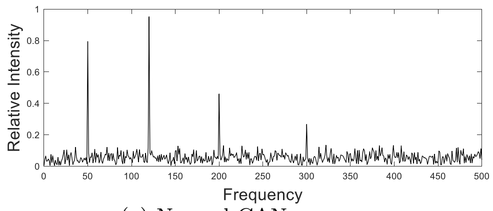
Figure 1: Accuracy gap of existing methods [10] and [14] when applied to the Tesla Model 3 2022 dataset and the Car-Hacking dataset [17].

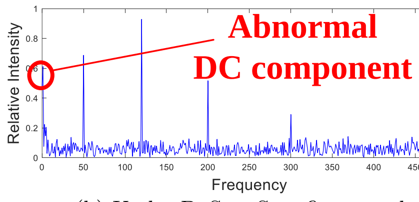
ML or DL based IDS algorithms use highly complicated models with huge number of parameters and thus they are not suitable to run in a vehicle with limited computation resource.

To address the above limitations of existing approaches and challenges that datasets from modern vehicles have more complex functions than the old ones, in this paper, a lightweight unsupervised learning based IDS with frequency-time domain analysis (LUFT-CAN) is proposed. LUFT-CAN is developed with the following key observation.

In this work, we use frequency domain analysis for CAN bus traffic and demonstrate that the frequency domain features of normal and anomalous sequences differ significantly. Our proposed a model integrates FFT and CNN to capture and process CAN ID components across different frequencies, thereby enabling accurate anomaly detection.

For Tesla Model 3 2022, Fig. 2 shows the spectrum diagrams of normal and abnormal CAN sequences. In Fig. 2(a), normal CAN frames sequences exhibit periodic patterns corresponding to the three most significant frequency components at 5 ms, 10 ms, and 20 ms, reflecting the regular communication behavior of the CAN bus. In contrast, in Fig. 2(b), the DoS attack introduces an abnormal direct current (DC) component in the low-frequency domain, due to the flooding of noise frames, disrupting the normal periodicity of the sequences. On the other hand, in Fig. 2(c), the frequency spectrum of the fuzzy attack shows a highly irregular pattern, further diverging from the normal sequences. With fuzzy attack, multiple frames are injected to reverse engineer the content of the data frames.


(a) Normal CAN sequence.



(b) Under DoS or Spoofing attacks.

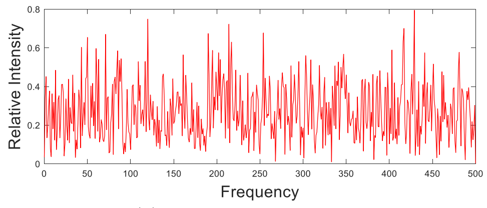

(c) Under Fuzzy attack.

Figure 2: The spectrum diagrams of normal and abnormal CAN sequences. The traffics under DoS/Spoofing/Fuzzy attacks are drastically different from normal traffics.

Another metric to analyze normal and abnormal sequences is cosine similarity that provides a sense of similarity between two sequences. Table 2 shows cosine similarity of ID feature between two sequences of various types with 10K CAN frames. The cosine similarity between two normal sequences are 0.821, but this metric between normal sequences and those under DoS, Spoof, and Fuzzy attacks is 0.207, 0.381, and 0.174 respectively, showing a salient difference between normal CAN sequences and attacks.

Inspired by the above observation, in LUFT-CAN, a Fast Fourier Transform (FFT) [16] module is used to analyze periodicity, global trends, and noise suppression by highlighting low-frequency components and filtering out high-frequency noise in the frequency domain. In the time domain, convolu-

Table 2: Cosine similarity between normal and abnormal sequences on the Tesla dataset.

Categories	Normal	DoS	Spoofing	Fuzzy
Normal	0.821	0.207	0.381	0.416

tional layers are used to capture the short-term features, including the order of sequences and time interval of CAN sequences. To integrate time-domain and frequency-domain analyses, the time series data of CAN bus are segmented and reconstructed according to temporal periods computed through FFT analysis. Data is segmented according to the periodicals and reorganized as 2D tensors where convolution layers can capture both frequency-time domain features. To reduce computation overhead, LUFT-CAN is further compressed to fit the resource constrained vehicular SoCs.

The contributions of this paper are as follows:

- For the first time, we use frequency domain analysis on CAN bus data and found the drastic difference of normal and abnormal sequences. Inspired by this observation, a novel lightweight, unsupervised learning based IDS model, LUFT-CAN, is proposed which can capture both the frequency and temporal features of long time-series CAN frames sequence. This model is the *first one* to our best knowledge using frequency-time domain unsupervised autoencoder to address the limitations of supervised learning based IDS. Additionally, LUFT-CAN is optimized through weight quantization to enhance its efficiency, particularly in terms of execution time.

- To evaluate the performance of our method, we conducted a series of experiments on two modern car datasets, i.e., from Tesla Model 3 2022 and LeapMotor C10 2024, as well as a public dataset [17]. The proposed model achieves an F1-Score of 97.1% and 96.7% on the Tesla and LeapMotor datasets, respectively, which are much higher than state-of-the-art approaches.

- Our proposed method was ported to a LeapMotor C10 2024 test car with a Qualcomm 8295 MCU, and the inference time of LUFT-CAN is 14.27s per 100K frames, which is lightweight enough for real time intrusion detection in real vehicles.

The remainder of this paper is organized as follows. Section 2 reviews the related work. Section 3 presents the proposed method. Section 4 evaluates the experimental results. Section V concludes the paper.

2. Related Work

This section provides an overview of existing works on vehicular network security, including various IDS methods. The CAN IDS methods can be categorized into four types: rule based IDS, ML based IDS, supervised DL based IDS, unsupervised DL based IDS. The characteristics, advantages, and disadvantages of different types of works are summarized in Table 3, detailed as follows:

2.1. Rule Based IDS

Rule based IDS methods rely on predefined rules or signatures to detect anomalies or attacks within a system. The method in [4] improves CAN security by generating unique fingerprints for ECUs based on their clock skew fingerprints, resulting in identification of the responsible compromised ECU. The method in [5] analyzes the offset ratio and temporal differences between request and response messages by broadcasting remote frames on the CAN bus and collecting responses from the sender nodes, thereby identifying the compromised nodes with lower computation power. The method in [6] can examine the bit values in the frame payloads by checking a whitelist quickly, and trigger an alert when the binary sequence's behavior deviates from the whitelist.

While most rule-based IDS approaches rely on DBC files by comparing observed CAN ID frequencies against predefined specifications, there also exist data-driven methods that do not require DBC files. These include approaches that analyze features such as bandwidth utilization [5] or clock periods [4], which are particularly effective in detecting DoS attacks. However, these methods generally fall short in identifying Spoofing or Fuzzy attacks, which do not cause obvious changes in traffic patterns.

Furthermore, some researchers have designed IDS that utilize multiple rules for detection. The method in [18] utilizes information entropy theory to employ a multi-objective optimization algorithm with dual evolutionary selection to enhance the IDS model and effectively balance these conflicting goals.

Rule based IDS methods compare incoming data against known attack patterns or behavior rules, triggering alerts when matches are detected. While this approach is straightforward and effective, one method can only detect one predefined type of attack. Further, it places high demands on data traffic and is challenging to update to address the latest or previously unseen attacks.

Table 3: Comparison of proposed methods and existing methods.

Methods	[4],[5],[6], [18]	[7],[9],[19], [20],[21]	[10],[11],[12], [22],[23],[24], [25],[26],[27].	[13],[14],[15], [28],[29],[30], [31],[32],[33].	**LUFT-CAN**
Categories	Rule Based	Traditional ML	Supervised DL	Unsupervised DL	Unsupervised DL
Models	-	OC-SVM, Entropy, Mixed	CNN, LSTM, GRU	CNN+MLP, LSTM, Transformer	Frequency& Time Analysis
Unseen Attacks Detection	✗	✗	✗	✓	✓
Inference Delay	low	low	high	high	low

2.2. Traditional Machine Learning Based IDS

Traditional machine learning based IDS methods rely on manually selected features and supervised algorithms like decision trees or support vector machine (SVM) to classify traffic as normal or malicious. The method in [19] proposed a lightweight IDS based on time-interval conditional entropy, which measures the conditional entropy of CAN message sequences using inter-message intervals. The methods in [7] and [20] select the data by analyzing the high-dimension entropy extracted from CAN frames and the CAN data frequency based on ML methods. The method in [9] uses SVM to analyze the data features. The method in [21] utilizes two distinctive features, inter-frame space and counter information, and designs IDS model based on random forest and XGBoost algorithms.

These statistical analysis methods and ML models employ fixed feature extraction patterns. However, they fail to capture the characteristics of complex attacks. Additionally, their accuracy is low, and the range of detectable attack types is limited, making them unsuitable for the demands of modern CAN IDS.

2.3. Supervised Deep Learning Based IDS

The method in [10] transforms the CAN ID into 2-dimensional binary matrix and uses a model based on the CNN to extract the spatial features of the CAN frames. The method in [22] utilizes the image network to analyze the 2D feature maps extracted from CAN sequences. The method in [23] develops a supervised constructive ResNet model for classifying multiple CAN bus attacks, addressing the multi-class classification problem by distinguishing normal traffic from various types of attacks. The method in [11] converts a CAN frame into a vector and uses a hybrid method with 1-D CNN and LSTM model to extract the spatial and temporal features of consecutive CAN frames. The method in [24] combines autoencoder and generative adversarial network (GAN) to handle large volumes of labeled data. The method in [25] employs an LSTM in a multitask learning framework to detect multiple anomalies. The method in [12] integrates self-attention mechanism with Gated Recurrent Unit (GRU) model to extract long term dependency from longer CAN frames sequences. The method in [26] employs a sequential CNN approach to automatically extract spatial features from in-vehicle network traffic. The method in [27] designs an auxiliary classifier GAN to generate more abnormal CAN frames sequences to balance the ratio of norms and anomalies to increase the detection accuracy.

The methods above cannot detect previously unseen attacks, as they are limited to defend against the attack types represented in the training dataset.

2.4. Unsupervised Deep Learning Based IDS

Unsupervised learning model requires only the normal data for training, thus can detect unseen attacks and demonstrating greater robustness than supervised counterparts. Normal data is the majority in numbers, while anomalous CAN bus data are scarce and unpredictable, resulting in high labeling costs and imbalance in datasets. Unsupervised methods do not rely on pre-labeled anomalous samples and only depend on the normal samples. Moreover, since they do not require labeled data, they assess the distance with mean square loss or cosine similarity between anomalous and normal states, regardless of whether the anomalies stem from seen or unseen attacks. The method in [13] combines CNN and multilayer perceptron (MLP) to create a hybrid autoencoder to reconstruct the input ID sequence, analyzing anomaly scores with the correlation between the original and reconstructed sequence. The method in [28] deploys a CNN-LSTM model to fit for the one-shot learning to detect unseen attacks. Autoencoder-based CAN IDS works for unsupervised training, but simple CNN or MLP networks are insufficient for extracting complex time series information from CAN data messages. A few works use time series analysis model for CAN IDS. The methods in [14] and [29] employs LSTM to predict future CAN frames sequences for the intrusion detection. The method in [30] employs an LSTM autoencoder to recognize intrusive events from CAN gateways.

However, unsupervised LSTM exhibits low accuracy as it struggles to capture the long-term sequence feature of the entire CAN frames sequences, so the attention mechanism is introduced into model. The method in [31] designs a bidirectional encoder representations from transformers (BERT) model to reconstruct the CAN frames with masked autoencoder method. The method in [32] converts CAN IDs into consisting word tokens and designs a generative pretrained transformers model to learn the normal patterns of CAN ID sequences. The method in [15] combines LSTM with attention mechanism for long term CAN frames prediction. The method in [33] fuses CNN and RNN with attention mechanism to identify driving behavior embedded in CAN sequences.

However, the attention mechanism involves huge computation overhead, which limits its application in recourse constraint vehicular environments. The method in [34] designs a patch-transformer and BiLSTM fused model to predict CAN sequence and uses model distillation method to decrease the computation demand but this method also cannot be implemented on real cars as its overall computation overhead is still too high.

3. Threat Model

This section delineates the threat model adopted throughout our research. We assume that an attacker has the capability to remotely compromise one or more ECUs through a wireless interface, such as a telematics port [35], or physically via the On-Board Diagnostics II (OBD-II) port [36]. There are three types of intrusion attacks against CAN bus [17], introduced as follows:

3.1. Denial of Service (DoS) Attack

In this attack, the attacker deliberately disrupts normal operations by overwhelming the network with an excessive number of CAN messages. This is achieved by exploiting the priority mechanism inherent in the CAN protocol, where messages with lower numeric IDs are granted higher priority during arbitration. By injecting messages with the lowest possible identifier (ID 0x000), an adversary ensures that these messages consistently win the arbitration process, thereby effectively monopolizing the bus. As a consequence, legitimate messages from other nodes are delayed or entirely blocked, leading to a significant degradation in system performance.

3.2. Fuzzy Attack

This attack is characterized by the injection of data that exhibits random identifier sequences. The adversary generates CAN messages with arbitrary IDs and corresponding data frames, without knowing any pre-defined protocol or structure. The randomness in the CAN IDs, ranging from 0x000 to 0x7FF, which covers the entire spectrum of possible identifiers, is intentionally employed to obscure the inherent semantics of the messages. Then, the attacker aims to facilitate reverse engineering of the communication protocol and to induce confusion among the receiving nodes. This attack can lead to an overall degradation of system performance, and may even allow the attacker to glean sensitive information about the underlying architecture or operational parameters of CAN network.

3.3. Spoofing Attack

In this attack, the adversary first intercepts and monitors valid messages transmitted by a specific ECU. By analyzing these messages, the attacker gathers critical information about the transmission frequency and the format of legitimate messages. The next phase of the attack involves the imitation of the compromised ECU. The attacker reproduces the intercepted messages at the same or a higher frequency, thereby misleading other ECUs into believing that these messages originate from the authentic source. This replication can induce abnormal behaviors within the network. For instance, it might cause nonauthorized control of vehicle functions, such as activating lights or other systems directly through the injected CAN messages.

All the attack mechanisms mentioned above are based on message injection. The message injection attack is affected by the frequency at which the ECU sends CAN frames and the interval between adjacent CAN messages. Based on such foundings, we propose an unsupervised model which can analyze the frequency-time features of CAN frames and extract the normal scenarios of the entire CAN sequence.

4. The Proposed LUFT-CAN Model

4.1. Overview

According to the observation in Fig. 1, LUFT-CAN extracts the frequency and time domain features that differentiate the benign status from abnormal ones. LUFT-CAN has two phases: model training and anomaly detection, as shown in Algorithm 1. In the training phase, the model iteratively learns a representation of normal data by reconstructing each input sample from the training dataset and minimizing the reconstruction error calculated by the mean squared error (MSE). Once training is complete, a threshold is established based on the reconstruction loss of the training data, serving as a benchmark for anomaly detection. During the anomaly detection phase, each sample from the test dataset is processed by the trained model to generate a reconstructed output. The algorithm calculates an anomaly score by measuring the discrepancy between the test sample and its reconstruction. If this anomaly score exceeds the predefined threshold, the sample is classified as an anomaly. Otherwise, it is considered normal. The advantages of the proposed IDS method are as follows:

- To capture characteristics of the time and frequency domains, the CAN sequence is segmented according to the frequency components. Convolutional layers are used to extract features of these segments, followed by a layer to concatenate those features. In this manner, the differences between benign states and those under attack in both frequency and time domains can be captured for anomaly detection.
- To improve the computational efficiency, a model compression method is proposed with weight quantization to make the model lightweight.

4.2. Data Preprocessing

Fig. 3 illustrates the fields within the CAN bus data frame. The Arbitration ID (CAN ID) serves as a unique marker to denote the type or category of each data frame. Data frames with distinct CAN IDs are transmitted and interpreted by Electronic Control Units (ECUs) with specialized functions. The payload, encompassing 64 bits (equivalent to 8 bytes, ranging from DATA [0] to DATA [7]), is structured such that each individual bit or a concatenation of multiple bits corresponds to a specific function or status present in the actual vehicle. The data preprocessing step works as follows:

Algorithm 1: The proposed IDS method.

```
Input  : Training-loader $X_{train}$, Testing-loader $X_{test}$
Output: Intrusion detection result
// Model Training Phase:
1  while $Model$ $has$ $not$ $been$ $trained$ do
2      for $s \in X_{train}$ do
3          $\hat{s} = Model(s)$;
4          $Loss = MSE(s, \hat{s})$;
5          $Loss.backward()$;
6      end
7  end
8  Calculate Threshold $\tau$: by $Loss$ of $X_{train}$;
// Anomaly Detection Phase:  1 represents anomalies, 0
    represents norms
9  for $s_{test} \in X_{test}$ do
10     $\hat{s}_{test} = Model(s_{test})$;
11     AnomalyScore $R(s_{test}) = MSE(s_{test}, \hat{s}_{test})$;
12     if $R(s_{test}) > \tau$ then
13         $result = 1$;
14     end
15     else
16         $result = 0$;
17     end
18 end
```

4.2.1. Format Convertion

The proposed IDS model handles CAN frames a time-series sequence. In the CAN data frames, the ID field varies over time, and the time series is formed with respect to the ID. The payloads of different CAN IDs have different functional definitions and adjacent CAN frames typically have different IDs. There is no inherent relationship between the payloads of adjacent frames, nor do they contain meaningful temporal information. Therefore, including payloads as model input is useless, which is also the case as in previous works [13], [14]. Consequently, we confine our analysis to the ID field of the captured CAN data frames by converting from hexadecimal to decimal.

4.2.2. Timestamp Processing

Although there is no timestamp attribute in CAN frames, we can get the timestamps from the CAN sniffer device when collecting the data. Upon analyzing the entire dataflow of the CAN bus, it can be observed that the time interval between adjacent data frames is time varying. Hence, we incorporate

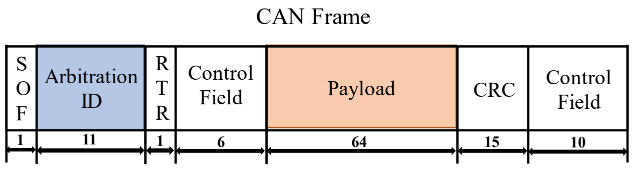


Figure 3: The standard structure of CAN frame.

an analysis of timestamps during preprocessing by adding the time interval between adjacent data frames as a new feature, and the time intervals between same ID frames is also extracted as auxiliary features for analysis. A consecutive CAN frames sequence is defined as follows:

$$\{F_{d_1,q_1,1}, F_{d_2,q_2,2}, F_{d_3,q_3,3}, ..., F_{d_m,q_m,m}\} \tag{1}$$

where F is a CAN frame, m is the frame index in the sequence, d_m is the CAN ID of the frame, q_m is the occurrence index of ID d_m. The feature of each frame can be expressed as a tuple:

$$< d_m,\ ts_{d_m,q_m,m} - ts_{d_m,q_{m-1}},\ ts_{d_m,m} - ts_{d_{m-1},m-1} > \tag{2}$$

where ts donates the timestamp of a CAN frame. This tuple serves as the input to the model, as illustrated by the Input s in Fig. 4(a).

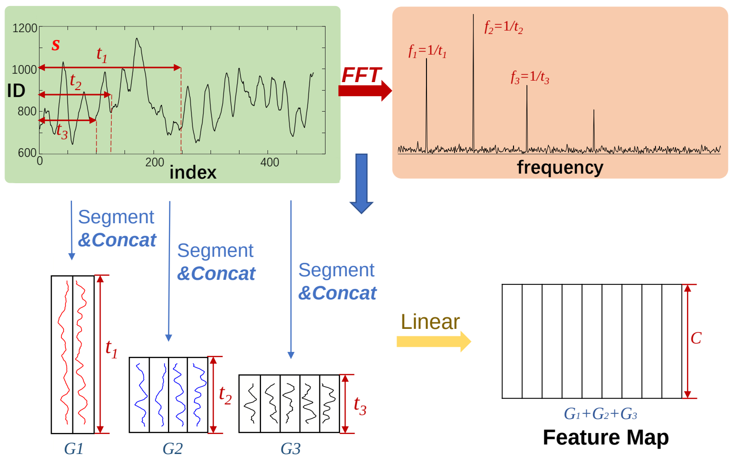
The time intervals between frames with the same CAN ID are fixed, which allows for predictable patterns that can be utilized as features in the model. By incorporating intervals between frames as features, it enables the model to more effectively distinguish between normal and anomalous behavior.

4.2.3. Data Normalization

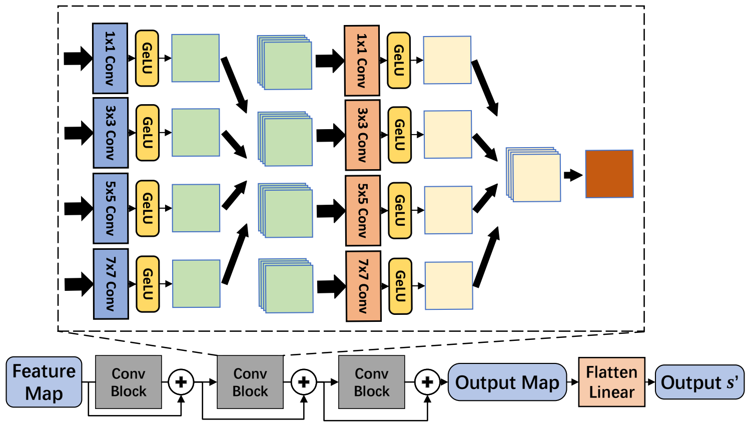
We use the reversible instance-wise normalization (RevIN) module [39] when preprocessing the data. The RevIN module can normalize the traffic data to a mean of 0 and a standard deviation of 1, and the inverse reversible instance-wise normalization (iRevIN) module can restore the normalized data. Initially, the RevIN technique denoises non-stationarities from the input traffic data, thereby mitigating intrinsic distribution shift issues inherent in time series data. As a result, the CAN dataflow becomes more concentrated and more analyzable for the DL model.

4.3. The Proposed LUFT-CAN Architecture

The overall architecture of the proposed IDS model is illustrated in Fig. 4. To simultaneously capture both temporal and frequency-domain features, the time series data including CAN ID and time intervals which are one dimensional are segmented according to their periodicals and these segments



(a) Model architecture of segmentation module.



(b) Model architecture of feature expansion and refactoring module.

Figure 4: Overview of our proposed model architecture.

are reorganized as a 2D matrix as shown in Fig. 4(a). This makes data suitable to convolutional layers for feature extraction. The FFT [16] module is used to analyze the frequency components, extracting periodicity, global trends and perform noise filtering of CAN sequence. In the frequency domain, the signal trends are captured by low-frequency components, while high-frequency components correspond to noise and thus low pass filtering can remove noises by selecting the top-k frequency components with the highest intensity.

The time domain features encompass short-term trends, such as local variations and abrupt changes, which reflect dynamic fluctuations within short time windows, and long-term dependencies, including continuous trends and evolving seasonal patterns. In CAN data sequences, the short-term trends are more important than long-term dependencies because the ordering of different data frames and the intervals between frames in the CAN sequences are much more critical than the overall variations in their quantity and intensity. Therefore, the convolutional neural network is selected as backbone layers instead of transformer with attention mechanism which focus more on long-term dependencies [37].

Overall, the preprocessed CAN bus data is fed into the FFT module for frequency domain analysis of the CAN sequence. Then, the sequence is segmented into multiple groups according to the frequency components, resulting in the formation of several feature maps [38]. These feature maps are then processed by the feature expansion module to generate high-dimensional feature maps in order to extract and represent more comprehensive temporal features. Finally, the feature maps are mapped back to a predicted sequence through a flattening layer and a linear layer. The modules of the model are as follows:

4.3.1. Sequence Segmentation Module

As shown in Fig. 4(a), this module segments the CAN frames sequence into distinct groups of segments according to different frequency components. A 2-D feature map is later formed upon these segments, which captures the frequency-time domain features.

Initially, the CAN frames sequence $\boldsymbol{s}$ of length l is input into the module. The top-k frequency components are obtained by FFT, as follows:

$$\{f_1, f_2, ..., f_i, ..., f_k\} = topk\left(FFT\left(\boldsymbol{s}\right)\right) \tag{3}$$

where f_i denotes the i-th frequency component, and the number of k is a hyperparameter of this model. A corresponding sequence of periods $\{t_1, t_2, \ldots, t_k\}$ is formed by:

$$t_i = \frac{1}{f_i},\ i \in \{1, 2, ..., k\} \tag{4}$$

Then the CAN frames sequence is segmented into k groups $\{G_1, G_2, ..., G_k\}$ according to the periods in the sequence $\{t_1, t_2, \ldots, t_k\}$, each corresponding to a unique feature vector group. The sequence lengths for these groups are as follows:

$$l_i = \frac{l}{t_i},\ i \in \{1, 2, ..., k\} \tag{5}$$

where l_i denotes the length of the vector group G_i , divided by t_i . We apply three distinct linear layers with the number of input channels l_i and the number of output channels being c, as follows:

$$\hat{\boldsymbol{s}}_{i,j} = Linear_i\left(\boldsymbol{s}_{i,j}\right), i \in \{1, 2, ..., k\},\ j \in \{1, 2, ..., l_i\} \tag{6}$$

where $s_{i,j}$ denotes the raw vector of G_i , $\hat{s}_{i,j}$ denotes the linearized vector of G_i . In this manner, the vector length of $\{G_1, G_2, ..., G_k\}$ becomes c. The one-dimensional feature vectors are concatenated in parallel, resulting a two-dimensional feature maps $\{\boldsymbol{M}_1, \boldsymbol{M}_2, ..., \boldsymbol{M}_k\}$, each with dimensions of $c \times t_i$, as follows:

$$\boldsymbol{M_i} = concat\left(\hat{\boldsymbol{s}}_{i,1}, \hat{\boldsymbol{s}}_{i,2}, ..., \hat{\boldsymbol{s}}_{i,l_i}\right), i \in \{1, 2, ..., k\}\,,\ \{\hat{\boldsymbol{s}}_{i,1}, \hat{\boldsymbol{s}}_{i,2}, ..., \hat{\boldsymbol{s}}_{i,l_i}\} \in G_i \quad (7)$$

Eqn. (7) is the reorganized 2D tensor $\boldsymbol{M}_i$transformed from the 1D CAN data sequences G_i as in Eqn. (3).

4.3.2. Feature Expansion Module

As shown in Fig. 4(b), this module is designed to generate high-dimensional features from the two-dimensional feature maps for richer transformations through nonlinear activation functions, enabling the model to learn more complex mappings. This module consists of several blocks of convolutional neural network layers. As illustrated in Fig. 3(b), the first layer has four convolution channels with kernel sizes of $1 \times 1, 3 \times 3, 5 \times 5, 7 \times 7$, respectively, each with a stride of 1, while padding is applied to maintain the spatial dimensions of the feature maps. For activating the features, GeLU activation is applied following the first convolutional layer for the retention of positive features and the discarding of negative features, that is:

$$\boldsymbol{M}_{i,p\times p} = GeLU\left(Conv_{p\times p}\left(\boldsymbol{M}_i\right)\right), i \in \{1, 2, ..., k\}\,,\ p \in \{1, 3, 5, 7\} \quad (8)$$

where $\boldsymbol{M}_{i,p\times p}$ donates the output of convolutional layer with the kernel size of $p \times p$. The GeLU function is as follows:

$$\text{GeLU}\left(\boldsymbol{x}\right) = \boldsymbol{x} \cdot \frac{1}{2}\left(1 + \tan\text{h}\left(\sqrt{\frac{2}{\pi}}\left(\boldsymbol{x} + 0.044715\boldsymbol{x}^3\right)\right)\right) \quad (9)$$

where $\boldsymbol{x}$ represents each input element of GeLU function. The resulting feature maps from each channel are concatenated along the channel dimension, as follows:

$$\hat{\boldsymbol{M}}_i = concat\left(\boldsymbol{M}_{i,1\times1}, \boldsymbol{M}_{i,3\times3}, \boldsymbol{M}_{i,5\times5}, \boldsymbol{M}_{i,7\times7}\right), i \in \{1, 2, ..., k\} \quad (10)$$

The output channel count of the first convolutional layer is four times of the input channel count. The result is passed to the second layer which mirrors the structure of the first layer: it has an input channel number that is four times of the original, and the number of output channels is identical to the original channels, thus forming an inverted residual structure to enhance feature representation. The final output of the convolutional layers is also activated using GeLU and serves as the output of the convolutional block and is connected by a residual module. The entire feature expansion module is as follows:

$$\hat{\boldsymbol{M}} = \boldsymbol{M} + ConvLayer2\left(ConvLayer1\left(\boldsymbol{M}\right)\right) \tag{11}$$

where $\hat{\boldsymbol{M}}$ denotes the output matrix of feature expansion module, $ConvLayer$ includes the convolutional layers and GeLU activation function. The number of convolutional blocks is a hyperparameter of this model.

4.3.3. Data Refactoring Module

This module is designed to convert the high-dimension feature maps into the predicted sequence. This module consists of a flattening layer and a fully connected layer, as follows:

$$\hat{\boldsymbol{s}} = Linear\left(Flatten\left(\hat{\boldsymbol{M}}\right)\right) \tag{12}$$

where $\hat{\boldsymbol{s}}$ denotes the output vector of this module. The flattening layer converts the high-dimensional feature map into a one-dimensional feature vector, while the fully connected layer aligns the length of this feature vector with that of the input vector, producing the predicted sequence. The predicted sequence output by this module serves as the final output of the model.

4.4. Training and Threshold

In the training phase, we choose mean squared error (MSE) as the loss function and reconstruction error as follows:

$$Loss = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2 \tag{13}$$

where y_i , $\hat{y}_i$ are the elements in raw and reconstructed sequence $\boldsymbol{s}$, $\hat{\boldsymbol{s}}$ respectively, n represents the length of the sequence. During training, the model is iteratively optimized to minimize the MSE between each input $\boldsymbol{s}$ and its reconstructed output $\hat{\boldsymbol{s}}$. The model's reconstruction error on the validation set is used to evaluate its fitting performance.

We use the Adam optimizer with a learning rate of 0.0001 and adopt a warm-up scheduler. We apply the early stopping mechanism during the training phase to minimize overfitting and ensure that the model generalizes well to unseen data. By monitoring the model's performance on a validation set, we interrupt the training process when the performance stops improving for a specified number of epochs, thereby avoiding unnecessary computation and preventing the model from becoming too tailored to the training data.

This helps to achieve a balance between underfitting and overfitting, improving the model's overall robustness. If the training loss keeps decreasing but validation loss starts increasing, as follows:

$$\begin{gathered} Loss_{\text{train}}\left(t\right) < Loss_{\text{train}}\left(t-1\right) \\ Loss_{\text{val}}\left(t\right) > Loss_{\text{val}}\left(t-1\right) \end{gathered} \tag{14}$$

where t represents the training epoch, it is considered that the model has been fitted. Because of the early stopping strategy, the number of training epochs is not fixed, with an average number being 100.

Once training convergence is reached, a threshold is derived based on the distribution of reconstruction errors over the training set, following an existing selection method [40]. Specifically, the MSE values for all training samples are analyzed, and a suitable percentile or statistical criterion is chosen to establish the threshold. This threshold then serves as the criterion for distinguishing normal from anomalous inputs in subsequent inference stages.

Initially, the model computes an anomaly score $R\left(\boldsymbol{s}\right)$ for each sample $\boldsymbol{s}$ based on MSE. The first phase involves analyzing the distribution of these scores on the training set X_{Train}. A candidate threshold is defined by selecting a specific quantile of the training set score distribution. For instance, if the 95th percentile is chosen, the preliminary threshold is given by:

$$\tau_{(train)} = \text{Quantile}_{(0.95)}\left\{R\left(\boldsymbol{s}\right) \mid \boldsymbol{s} \in X_{train}\right) \tag{15}$$

which represents the upper bound of anomaly scores for the majority of normal instances as learned during training. In our experiments, the approximate proportion of anomalies is used as the chosen percentile, with 95th percentile typically being selected.

In the second phase, the validation set is used to refine this threshold. The anomaly scores for the validation samples are computed, and the candidate threshold is adjusted to optimize a performance metric such as the F1-score. This can be achieved through a grid search or an iterative process where multiple candidate thresholds are evaluated against the validation set X_{Valid}. The optimal threshold τ is then selected as the value that best balances the trade-off between false positives and false negatives, ensuring that the model accurately identifies anomalies without excessive misclassification. Thus, the final threshold can be represented as:

$$\tau = arg\max_{\tau'} \text{F1} - \text{score}\left(\tau';\, X_{Valid}\right) \tag{16}$$

where τ' varies over a range of candidate thresholds in the training set X_{Train}. This two-step approach, combining the statistical properties of the training set with performance-driven calibration on the validation set, ensures that the anomaly threshold is both statistically sound and practically effective, performing well on the test set X_{Test}.

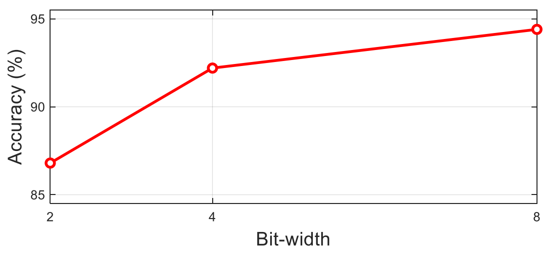


Figure 5: Accuracy comparison for 2-bit, 4-bit, and 8-bit quantization on the Tesla dataset.

4.5. Model Compression

To improve the computational efficiency, Quantization-Aware Training (QAT) strategy [41] is used to compress CNN layers after the model has been trained. QAT involves simulating the effects of quantization during the training phase, allowing the model to learn under conditions that mimic lower precision. This model implements 8-bit quantization, which can be expressed mathematically as:

$$\omega_{i,j,quantized} = round\left(\frac{\omega_{i,j}}{2^{b-1}}\right) \tag{17}$$

where $\omega_{i,j}$ and $\omega_{i,j,quantized}$ denote the weights of the proposed method and the quantized model respectively, b is the bit-width of the quantized weights, which equals to 8 in our proposed method. It is evident that reducing the precision of model parameters can lead to a degradation in model accuracy. Therefore, we adopt the Quantization-Aware Training (QAT) approach to mitigate this loss. Specifically, for the trained LUFT-CAN model, we insert fake quantization nodes to simulate parameter quantization. Then we use the supervised fine-tuning on the quantized-aware model using the full dataset, allowing the original model parameters to adapt to the quantization-induced errors. This procedure significantly reduces the accuracy loss when the final quantized model is obtained. Fig. 5 illustrates the accuracy across different quantization precisions. Notably, 8-bit quantization achieves highest accuracy and thus is adopted in our method.

5. Evaluation

In this section, we evaluated the performance of the proposed IDS method on two datasets extracted from modern cars, Tesla Model 3 2022 and Leap-Motor C10 2024, and a public Car-Hacking dataset [17].

5.1. Experimental Setup

Our proposed IDS method is developed using Python with the Pytorch [42] library. The experiments were conducted on a personal computer equipped with an Intel i7-13700K CPU and an NVIDIA RTX 4060 GPU with 8 GB memory, running Windows 11. To evaluate the inference efficiency in the computing resource constrained environment, we used a Raspberry Pi 4B with a BCM2711 CPU (4 Cortex-A72 cores at 1.8 GHz) and 8 GB RAM, running Ubuntu 22.04.

There are two hyperparameters in our proposed IDS. The model employs a top-k selection mechanism in the segmentation module, by choosing the periods with the highest top-k intensity, and k=3, ensuring that only the three most relevant elements are retained at each selection step. Additionally, the proposed IDS model incorporates three convolutional blocks, facilitating hierarchical feature extraction and enhancing the model's capacity to capture complex spatial representations. This configuration balances computational efficiency and representational power, optimizing performance for the intrusion detection on the real cars.

To compare, four recent IDS works, [10], [13], [14], [26] were selected. The method in [10] applies a DCNN network for network traffic patterns and identify malicious traffic with the spatial features. The method in [13] designs an unsupervised IDS with CNN layers and MLP autoencoders which is the best unsupervised learning based IDS on [17] so far. The method in [14] employs an LSTM model to identify anomalous messages within the CAN bus. The method in [26] employs a sequential CNN approach to extract spatial-temporal hybrid features from CAN frames. For each dataset and each method for comparison, retraining and hyperparameter optimization were performed using the corresponding dataset. Since the methods for comparison have not released their source code, we reproduced them based on our understanding of their original papers. The experimental results were obtained with parameters carefully tuned to the best of our ability. To ensure a fair comparison, we employed EarlyStopping to prevent overfitting and performed grid search hyperparameter tuning [45]. The search space included variations in learning rate (1e-3, 1e-4), batch size (16, 32, 64, 128), and training epochs (5, 10, 20). Optimal settings were determined via validation accuracy.

We also ported LUFT-CAN to a LeapMotor C10 2024 test car with Qualcomm 8295 MCU. The IDS model was trained on the personal computer with GPU support and then deployed on a central electric control unit (ECU) or microcontroller unit (MCU), which connects to the CAN bus and is capable of reading real-time CAN frames. The operating system in this car provides APIs designed for reading the real-time CAN bus data, enabling a continuous sequence of single-frame messages to be read by simply calling this API, without the need for segmentation. Each incoming frame is processed by a preprocessor, which reads the data and stores it in a buffer. Once 100 frames are accumulated in the buffer, they are subsequently fed into the IDS model for inference. This process ensures that the model can handle large amounts of real-time data effectively, allowing for continuous monitoring of the CAN

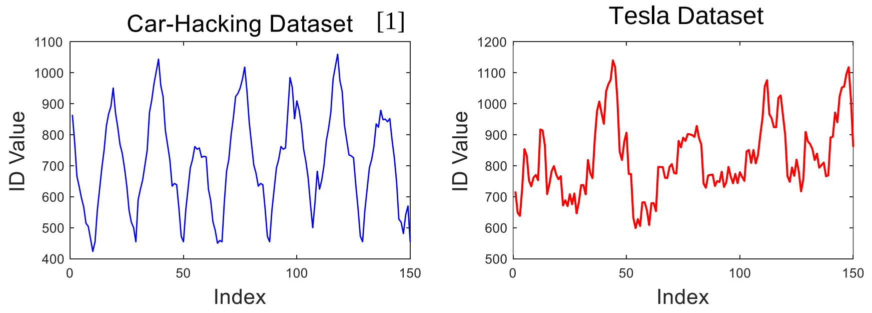


Figure 6: Difference between the Car-Hacking dataset [17] and the Tesla dataset.

bus traffic and the detection of potential anomalies.

The training time of LUFT-CAN model is around 10 seconds for each epoch with a 100K training set; and thus for a specific car, it can be trained within 12 epochs, i.e., 2 minutes in total, and detect threats later on. For different cars, LUFT-CAN needs to be retrained to capture the distinct internal frequency-time features of CAN bus for each vehicle which takes 2 minutes and thus is very efficient. Since our model is lightweight enough, the approach of using transfer learning with another vehicle's dataset does not significantly reduce the training time compared to retraining.

5.2. Modern Cars Datasets

We generated two CAN datasets from a Tesla Model 3 2022 and a LeapMotor C10 2024. Through data reverse engineering, we launched three different types of network attacks on the vehicles' CAN bus systems: DoS, Spoofing and Fuzzy. Among these, two types of Spoofing attacks were implemented: manipulating the vehicle's turn signal and horn, which are easy to identify on real cars. We monitored and collected traffic on the CAN bus and reverse engineer the CAN bus data fields according to [44], generating a dataset containing both benign and malicious data. Compared to existing datasets, such as the Car-Hacking dataset [17] whose data were collected from old cars before 2020, the data collected from modern vehicles after 2022 has more frequency components, as illustrated in Fig. 6.

Our datasets include over 200 unique CAN IDs and a larger number of frequency components. In contrast, the existing public dataset [17] collects data from older vehicles before 2020 with fewer functionalities, including only about 40 CAN IDs, which makes it outdated and less suitable for modern vehicle systems. For the Tesla dataset, the training set consists of 1M normal frames, while each of the four attack types in the test set includes 500K samples with injection attacks. It provides a diverse range of normal and attack data to train and evaluate the IDS model. For the LeapMotor dataset, the training set comprises 500K normal frames, and each of the two attack types

Table 4: Experimental results on the Tesla dataset.

Attack	Model	Accuracy	Recall	F1-score	AUC	FPR
DoS	DCNN[10]	0.764	0.621	0.797	0.668	0.285
	HistCAN[13]	0.348	0.252	0.516	0.356	0.539
	LSTM[14]	0.642	0.352	0.573	0.518	0.316
	Hybrid[26]	0.666	0.402	0.648	0.489	0.423
	LUFT-CAN	**0.973**	**0.972**	**0.986**	**0.963**	**0.046**
Spoof1	DCNN[10]	0.844	0.331	0.559	0.487	0.358
	HistCAN[13]	0.847	0.839	0.917	0.867	0.106
	LSTM[14]	0.796	0.640	0.802	0.712	0.217
	Hybrid[26]	0.673	0.351	0.597	0.462	0.426
	LUFT-CAN	**0.956**	**0.954**	**0.977**	**0.945**	**0.063**
Spoof2	DCNN[10]	0.755	0.643	0.811	0.684	0.276
	HistCAN[13]	0.276	0.070	0.432	0.090	0.890
	LSTM[14]	0.677	0.384	0.593	0.567	0.251
	Hybrid[26]	0.681	0.350	0.576	0.504	0.342
	LUFT-CAN	**0.961**	**0.960**	**0.980**	**0.950**	**0.059**
Fuzzy	DCNN[10]	0.821	0.353	0.589	0.483	0.386
	HistCAN[13]	0.791	0.760	0.884	0.735	0.221
	LSTM[14]	0.306	0.468	0.421	0.594	0.851
	Hybrid[26]	0.744	0.468	0.678	0.589	0.290
	LUFT-CAN	**0.873**	**0.866**	**0.932**	**0.882**	**0.102**

in the test set contains 200K frames with injection attacks since LeapMotor C10 has an more complicated CAN bus data fields, only two attacks were successfully implemented. In our experiments, we found that 500K normal CAN frames sequences was enough for the model convergence, fewer data would cause underfitting and more data would cost more training time and more computation resources. We chose 100K attacked data as test data. We released the Tesla dataset in [43].

We also choose a public dataset named Car-Hacking dataset [17]. This dataset is from old cars before 2020 which has no ADAS system, fewer functions and fewer car features, resulting in fewer categories of CAN messages and fewer frequency components than modern car datasets. So our method and existing methods show different results compared with the modern car datasets, as illustrated in Section 5.3.

Each vehicle manufacturer defines CAN bus data differently across vehi-

Table 5: Experimental results on the LeapMotor dataset.

Attack	**Model**	**Accuracy**	**Recall**	**F1-score**	**AUC**	**FPR**
DoS	DCNN[10]	0.838	0.761	0.910	0.610	0.541
	HistCAN[13]	0.348	0.281	0.516	0.430	0.421
	LSTM[14]	0.048	0.010	0.091	0.060	0.999
	Hybrid[26]	0.658	0.480	0.711	0.541	0.399
	LUFT-CAN	**0.937**	**0.923**	**0.961**	**0.931**	**0.060**
Spoof	DCNN[10]	0.674	0.340	0.619	0.401	0.537
	HistCAN[13]	0.716	0.650	0.835	0.616	0.419
	LSTM[14]	0.796	0.636	0.802	0.696	0.243
	Hybrid[26]	0.662	0.254	0.597	0.277	0.701
	LUFT-CAN	**0.960**	**0.957**	**0.979**	**0.942**	**0.074**

cle models, and the functions corresponding to different data, the frequency and order of CAN IDs, as well as the resulting spectral features after preprocessing. Therefore, retraining is performed for different datasets while comparing the accuracies of different methods [10], [13], [14], [26].

5.3. Experimental Results

Tables 4 and 5 present results for the Tesla Model 2 2022 and LeapMotor C10 2024 datasets respectively. The Tesla dataset includes four attacks, DoS, Spoof1, Spoof2 and Fuzzy. The LeapMotor dataset includes two attacks, DoS and Spoof. Accuracy, Recall, F1-score, ROC-AUC (Receiver Operating Characteristic-Area Under Curve) and FPR (False-Positive Rate) are the evaluation metrics. Our proposed method outperforms the other methods on each attack, with accuracy of 97.3%, 95.6%, 96.1%, 87.3%, Recall of 97.2%, 95.4%, 96.0%, 86.6%, F1-score of 98.6%, 99.7%, 98.0%, 93.2% and ROC-AUC of 0.963, 0.945, 0.950, 0.882 respectively for the four types of attacks with the Tesla dataset. For the LeapMotor dataset, LUFT-CAN yields accuracy of 93.7%, 96.0%, Recall of 92.3%, 95.7%, F1-score of 96.1%, 97.9%, ROC-AUC of 0.931, 0.942 respectively. These results significantly surpass the performance of the IDS methods in [10], [13], [14], [26]. For the FPR, LUFT-CAN gets 4.6%, 6.3%, 5.9%, 10.2% on the Tesla dataset and 6.0%, 7.4% on the LeapMotor dataset, and they are much lower than four previous IDS methods. The existing IDS methods [10], [13], [14], [26] perform poorly on modern vehicle datasets, because they fail to capture the frequency-time features. The two modern datasets contain a significantly larger number of CAN IDs and control functions, reflecting much more complicated system structure, and previous methods in [10], [13], [14], [26] cannot handle them properly.

Table 6: Experimental results on the Car-Hacking dataset [17].

Attack Type	**Model**	**Accuracy**	**F1-score**
DoS	DCNN[10]	**0.999**	**0.997**
	HistCAN[13]	0.992	0.997
	LSTM[14]	0.965	0.942
	Hybrid[26]	0.982	0.947
	LUFT-CAN	0.991	0.993
Gear	DCNN[10]	**0.999**	**0.998**
	HistCAN[13]	0.999	0.997
	LSTM[14]	0.990	0.988
	Hybrid[26]	0.979	0.996
	LUFT-CAN	0.992	0.989
RPM	DCNN[10]	**0.999**	**0.999**
	HistCAN[13]	0.997	0.996
	LSTM[14]	0.991	0.984
	Hybrid[26]	0.988	0.957
	LUFT-CAN	0.994	0.991
Fuzzy	DCNN[10]	**0.999**	**0.998**
	HistCAN[13]	0.998	0.997
	LSTM[14]	0.943	0.961
	Hybrid[26]	0.991	0.966
	LUFT-CAN	0.976	0.981

Table 6 shows results of the proposed IDS model in comparison with four previous IDS methods on the public Car-Hacking dataset [17]. Our proposed method achieves 99.1%, 99.2%, 99.4%, 97.6% accuracy for the four attacks, and it is approximately the same as those of existing works. The method in [13] can extract the temporal features much more easily with the dataset from old cars. The method in [10] is a supervised learning based model, with higher efficacy on feature extraction on datasets with simple structure. However, with more complicated datasets extracted from modern cars like the Tesla and LeapMotor datasets, these previous works perform poorly as indicated in Tables 4 and 5.

5.4. Execution Evaluation

We evaluated the inference time of LUFT-CAN on the Tesla dataset with 10K CAN frames and the compression version on a PC with an NVIDIA GeForce 4060 GPU and a Raspberry Pi 4B, as presented in Table 7. For comparison, we evaluated the inference time of other existing methods on a

Table 7: Experiment results of inference time on the Tesla dataset.

Environment	Accuracy	Inference Time
On PC with GPU	0.941	1.125s
On Raspberry Pi	0.941	6.266s
On PC with GPU after QAT	0.937	0.967s
On Raspberry Pi after QAT	0.937	5.891s

Table 8: Experiment results of inference time for different methods on PC with GPU.

Method	Inference Time
DCNN[10]	3.107s
HistCAN[13]	1.684s
LSTM[14]	7.192s
Hybrid[26]	5.214s
LUFT-CAN	0.967s

PC with GPU, as shown in Table 8. The previous work were implemented in the same experimental platform as our method. Table 8 shows that LUFT-CAN has the lowest inference time among these five IDS methods. With the proposed 8-bit quantization method, the execution time of the compressed LUFT-CAN is decreased by 14.0% over the original model on PC with GPU, and this number is around 6% on Raspberry Pi with almost the same accuracy, demonstrating that our quantization method has good performance on model compression.

In Raspberry Pi 4B to the power consumption when running our algorithm is 4.972 W. For comparison, the power consumptions of the other four methods [10], [13], [14], [26] are 4.982W, 5.023W, 5.066W, and 4.933W respectively, which are higher than ours. In LeapMotor C10 testbed, our algorithm consumes 41.6W of power while the testbed standby power is 38.2W. This can demonstrated that our method have lower energy consumption. Considering the extremely limited computational resources of the Raspberry Pi, the increase in power consumption caused by more complex algorithms is also limited.

5.5. In the Wild Evaluation on a Real Car

We also evaluated the inference time and CPU utilization in a LeapMotor C10 2024 test car, with Qualcomm 8255 as the MCU. Our method captures CAN frames through an API function based on the operating system of the T-Box in this car and collects real-time data from the vehicle's CAN bus.

Our method contains a data queue to store CAN frames, wherein every 100 stored CAN frames are input into the model for analysis. The inference time is 14.27s/100K frames when the MCU is fully used. With the real CAN bus transmission rate, the MCU utilization is 6.42% in average. Therefore, the proposed LUFT-CAN is suitable for efficient detection of threats in real cars.

6. Conclusion

This paper proposed a novel unsupervised learning based IDS designed specifically for vehicular CAN bus. The proposed method effectively captures and integrates both temporal and frequency domain features from CAN frames sequences for anomaly detection. By segmenting CAN frames according to frequency components, we generated feature vectors that are subsequently concatenated to form a 2D tensor feature map, enabling fusion of temporal and frequency information. To further ensure suitability for resource-constraint automotive environments, the model is compressed to improve computational efficiency. Experimental results demonstrated that the proposed method achieves over 94% accuracy on datasets from the Tesla Model 3 2022 and the LeapMotor C10 2024, outperforming state-of-the-art approaches. Consequently, the proposed IDS method holds significant promise for enhancing security in smart vehicles.

Acknowledgment

This work was supported in part by the key R&D programme of Zhejiang Province No. 2024C01012, in part by the National Key Research and Development Program of China No. 2023YFB4404404, in part by the National Natural Science Foundation of China under Grants 92373205 and 62374146, in part by Ant Group Research Fund, in part by the Key Technologies R&D Program of Jiangsu (Prospective and Key Technologies for Industry) under Grant BE2023005-2, and in part by CIE-Smartchip research fund No. 2023-004.

References

[1] Andy Greenberg, *Hackers remotely kill a jeep on the highway — With me in it - WIRED*, url: https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/, 2015.

[2] Eduard Kovacs, *Thieves use CAN injection hack to steal cars - Security Week*, url: https://www.securityweek.com/thieves-use-can-injection-hack-to-steal-cars/, 2023.

[3] Miller, Charlie and Valasek, Chris, *Remote exploitation of an unaltered passenger vehicle*, Black Hat USA, no. S 91, pp. 1–91, 2015.

[4] Zhao, Yilin and Xun, Yijie and Liu, Jiajia, *ClockIDS: A real-time vehicle intrusion detection system based on clock skew*, IEEE Internet of Things Journal, vol. 9, no. 17, pp. 15593–15606, 2022.

[5] Lee, Hyunsung and Jeong, Seong Hoon and Kim, Huy Kang, *OTIDS: A novel intrusion detection system for in-vehicle network by using remote frame*, IEEE Conference on Privacy, Security and Trust (PST), pp. 57–5709, 2017.

[6] Guiqi, Zhang and Qi, L and Chenhong, C and others, *Bit scanner: Anomaly detection for in-vehicle CAN bus using binary sequence whitelisting*, Computers & Security, vol. 134, 2023.

[7] Müter, Michael and Asaj, Naim, *Entropy-based anomaly detection for in-vehicle networks*, IEEE Intelligent Vehicles Symposium (IV), pp. 1110–1115, 2011.

[8] Song, Hyun Min and Kim, Ha Rang and Kim, Huy Kang, *Intrusion detection system based on the analysis of time intervals of CAN messages for in-vehicle network*, IEEE International Conference on Information Networking (ICOIN), pp. 63–68, 2016.

[9] Tanksale, Vinayak, *Intrusion detection for controller area network using support vector machines*, IEEE International Conference on Mobile Ad Hoc and Sensor Systems Workshops (MASSW), pp. 121–126, 2019.

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

[11] Cheng, Pengzhou and Han, Mu and Li, Aoxue and Zhang, Fengwei, *STC-IDS: Spatial–temporal correlation feature analyzing based intrusion detection system for intelligent connected vehicles*, International Journal of Intelligent Systems, vol. 37, no. 11, pp. 9532–9561, 2022.

[12] Javed, Abdul Rehman and Ur Rehman, Saif and Khan, Mohib Ullah and Alazab, Mamoun and Reddy, Thippa, *CANintelliIDS: Detecting in-vehicle intrusion attacks on a controller area network using CNN and attention-based GRU*, IEEE Transactions on Network Science and Engineering, vol. 8, no. 2, pp. 1456–1466, 2021.

[13] Shuguo Zhuo, Nuo Li, Kui Ren, *HistCAN: A real-time can ids with enhanced historical traffic learning capability*, Symposium on Vehicle Security and Privacy, 2024.

[14] Taylor, Adrian and Leblanc, Sylvain and Japkowicz, Nathalie, *Anomaly detection in automobile control network data with long short-term memory networks*, IEEE international Conference on Data Science and Advanced Analytics (DSAA), pp. 130–139, 2016.

[15] Sun, Heng and Chen, Miaomiao and Weng, Jian and Liu, Zhiquan and Geng, Guanggang, *Anomaly detection for in-vehicle network using CNN-LSTM with attention mechanism*, IEEE Transactions on Vehicular Technology, vol. 70, no. 10, pp. 10880–10893, 2021.

[16] Cooley, James W and Tukey, John W, *An algorithm for the machine calculation of complex Fourier series*, Mathematics of Computation, vol. 19, no. 90, pp. 297–301, 1965.

[17] HCRL, *Car-Hacking Dataset for the intrusion detection*, url: https://ocslab.hksecurity.net/Datasets/car-hacking-dataset, 2020.

[18] Zhang, Jiangjiang and Gong, Bei and Waqas, Muhammad and Tu, Shanshan and Chen, Sheng, *Many-objective optimization based intrusion detection for in-vehicle network security*, IEEE Transactions on Intelligent Transportation Systems, vol. 24, no. 12, pp. 15051–15065, 2023.

[19] Yu, Zhangwei and Liu, Yan and Xie, Guoqi and Li, Renfa and Liu, Siming and Yang, Laurence T, *TCE-IDS: Time interval conditional entropy-based intrusion detection system for automotive controller area networks*, IEEE Transactions on Industrial Informatics, vol. 19, no. 2, pp. 1185–1195, 2022.

[20] Olufowobi, Habeeb and Young, Clinton and Zambreno, Joseph and Bloom, Gedare, *SAIDuCANT: Specification-based automotive intrusion detection using controller area network (CAN) timing*, IEEE Transactions on Vehicular Technology, vol. 69, no. 2, pp. 1484–1494, 2019.

[21] Jeong, Yeonseon and Kim, Hyunghoon and Lee, Seyoung and Choi, Wonsuk and Lee, Dong Hoon and Jo, Hyo Jin, *In-vehicle network intrusion detection system using CAN frame-aware features*, IEEE Transactions on Intelligent Transportation Systems, vol. 25, no. 5, pp. 3843–3853, 2023.

[22] Gao, Sheng and Zhang, Linchuan and He, Lei and Deng, Xiaoyang and Yin, Huilin and Zhang, Hao, *Attack detection for intelligent vehicles via can-bus: A lightweight image network approach*, IEEE Transactions on Vehicular Technology, vol. 72, no. 12, pp. 16624–16636, 2023.

[23] Hoang, Thien-Nu and Kim, Daehee, *Supervised contrastive ResNet and transfer learning for the in-vehicle intrusion detection system*, Expert Systems with Applications, vol. 238, pp. 122181, 2024.

[24] Hoang, Thien-Nu and Kim, Daehee, *Detecting in-vehicle intrusion via semi-supervised learning-based convolutional adversarial autoencoders*, Vehicular Communications, vol. 38, pp. 100520, 2022.

[25] Balaji, Prashanth and Ghaderi, Majid and Zhang, Hongwen, *CANLite: Anomaly detection in controller area networks with multitask learning*, IEEE Vehicular Technology Conference (VTC2022-Spring), pp. 1–5, 2022.

[26] Lo, Wei and Alqahtani, Hamed and Thakur, Kutub and Almadhor, Ahmad and Chander, Subhash and Kumar, Gulshan, *A hybrid deep learning based intrusion detection system using spatial-temporal representation of in-vehicle network traffic*, Vehicular Communications, vol. 35, pp. 100471, 2022.

[27] Zhao, Qingling and Chen, Mingqiang and Gu, Zonghua and Luan, Siyu and Zeng, Haibo and Chakrabory, Samarjit, *CAN bus intrusion detection based on auxiliary classifier GAN and out-of-distribution detection*, ACM Transactions on Embedded Computing Systems (TECS), vol. 21, no. 4, pp. 1–30, 2022.

[28] Tariq, Shahroz and Lee, Sangyup and Woo, Simon S, *CANTransfer: Transfer learning based intrusion detection on a controller area network using convolutional LSTM network*, ACM Symposium on Applied Computing, pp. 1048–1055, 2020.

[29] Qin, Hongmao and Yan, Mengru and Ji, Haojie, *Application of controller area network (CAN) bus anomaly detection based on time series prediction*, Vehicular Communications, vol. 27, pp. 100291, 2021.

[30] Ashraf, Javed and Bakhshi, Asim D and Moustafa, Nour and Khurshid, Hasnat and Javed, Abdullah and Beheshti, Amin, *Novel deep learning-enabled LSTM autoencoder architecture for discovering anomalous events from intelligent transportation systems*, IEEE Transactions on Intelligent Transportation Systems, vol. 22, no. 7, pp. 4507–4518, 2020.

[31] Nwafor, Ebelechukwu and Olufowobi, Habeeb, *Canbert: A language-based intrusion detection model for in-vehicle networks*, IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 294–299, 2022.

[32] Nam, Minki and Park, Seungyoung and Kim, Duk Soo, *Intrusion detection method using bi-directional GPT for in-vehicle controller area networks*, IEEE Access, vol. 9, pp. 124931–124944, 2021.

[33] Zhang, Jun and Wu, Zhongcheng and Li, Fang and Xie, Chengjun and Ren, Tingting and Chen, Jie and Liu, Liu, *A deep learning framework for driving behavior identification on in-vehicle CAN-BUS sensor data*, Sensors, vol. 19, no. 6, pp. 1356, 2019.

[34] Cheng, Pengzhou and Wu, Zongru and Liu, Gongshen, *MKF-ADS: A multi-knowledge fused anomaly detection system for automotive*, arXiv preprint arXiv:2403.04293, 2024.

[35] Miller, Charlie and Valasek, Chris, *Remote exploitation of an unaltered passenger vehicle*, Black Hat USA, vol. 2015, no. S 91, pp. 1–91, 2015.

[36] Moore, Michael R and Bridges, Robert A and Combs, Frank L and Starr, Michael S and Prowell, Stacy J, *Modeling inter-signal arrival times for accurate detection of can bus signal injection attacks: a data-driven approach to in-vehicle intrusion detection*, ACM Conference on Cyber and Information Security Research, pp. 1–4, 2017.

[37] Li, Shanda and Chen, Xiangning and He, Di and Hsieh, Cho-Jui, *Can vision transformers perform convolution?*, arXiv preprint arXiv:2111.01353, 2021.

[38] Wu, Haixu and Hu, Tengge and Liu, Yong and Zhou, Hang and Wang, Jianmin and Long, Mingsheng, *TimesNet: Temporal 2D-variation modeling for general time series analysis*, International Conference on Learning Representations, 2023.

[39] Kim, Taesung and Kim, Jinhee and Tae, Yunwon and Park, Cheonbok and Choi, Jang-Ho and Choo, Jaegul, *Reversible instance normalization for accurate time-series forecasting against distribution shift*, International Conference on Learning Representations, 2021.

[40] Xu, Jiehui and Wu, Haixu and Wang, Jianmin and Long, Mingsheng, *Anomaly Transformer: Time series anomaly detection with association discrepancy*, International Conference on Learning Representations, 2022.

[41] Jacob, Benoit and Kligys, Skirmantas and Chen, Bo and Zhu, Menglong and Tang, Matthew and Howard, Andrew and Adam, Hartwig and Kalenichenko, Dmitry, *Quantization and training of neural networks for efficient integer-arithmetic-only inference*, IEEE Conference on Computer Vision and Pattern Recognition, pp. 2704–2713, 2018.

[42] Paszke, A, *Pytorch: An imperative style, high-performance deep learning library*, arXiv preprint arXiv:1912.01703, 2019.

[43] *Tesla Car Dataset. url: https://github.com/fcas-LAB/Tesla*, 2024.

[44] Cai, Yunlang and Shi, Hanxue and Wang, Xiaohang and Shen, Haoting and Lu, Li and Ren, Kui, *On Bit-level Reverse Engineering of Vehicular CAN Bus*, Design Automation Conference, 2025.

[45] Liashchynskyi P, Liashchynskyi P *Grid search, random search, genetic algorithm: a big comparison for NAS.*, arXiv preprint arXiv:1912.06059, 2019.