ECF-IDS: An Enhanced Cuckoo Filter-based Intrusion Detection System for In-vehicle Network

Sifan Li, Yue Cao, *Senior Member, IEEE,* Hassan Jalil Hadi, Feng Hao, *Senior Member, IEEE,* Faisal Bashir Hussain, and Luan Chen

***Abstract*—With the rapid advancement of vehicle connectivity and intelligent technologies, an increasing number of vehicles are now connected to the Internet. However, these connected vehicles are vulnerable to malicious attacks, posing serious security events. In particular, the in-vehicle controller area network (CAN) bus has witnessed a rise in incidents involving various network attacks, such as denial of service (DoS), fuzzy attacks, and gear attacks. In response, this paper proposes an enhanced cuckoo filter-based intrusion detection system (ECF-IDS) for in-vehicle network. The ECF-IDS builds on an enhanced version of the cuckoo filter. It first utilizes the cuckoo filter to establish two lists (a normal list and an intrusion list) based on the labeled dataset using Car Hacking Dataset (CHD) and can-train-and-test dataset. Then, the input CAN traffic is sequentially compared with these two lists, where the conflicting traffic is further identified using a BERT-based model. The ECF-IDS is experimentally validated using the CHD and can-train-and-test dataset, demonstrating higher detection efficiency, lower resource consumption, and detection success exceeding 99% compared to other algorithms presented in previous studies. Furthermore, we conducted real in-vehicle environment testing on the ECF-IDS model, and its detection performance proved to be excellent.**

***Index Terms*—CAN, IDS, BERT, Cuckoo Filter**

I. Introduction

NOWADAYS, the progression of vehicle intelligence and connectivity is also gaining attention [1]. Integral to this progression is in-vehicle communication, a crucial component within vehicles. The most widely used in-vehicle communication is controller area network (CAN) [2]. The traditional communication technologies, which are based on CAN, enable efficient data transmission, processing, and decision-making [3]. They facilitate communication among in-vehicle sensors, controllers, and electronic control units (ECUs) [4].

The CAN bus plays a crucial role in controlling communication between ECUs in vehicles. As a result, the security of the CAN bus directly influences the overall security of vehicles. However, the CAN bus itself possesses inherent vulnerabilities. First, the absence of encryption technology in CAN communication results in unencrypted data broadcasting [5]. This exposes two major risks: 1) Attackers can effortlessly capture necessary data, facilitating subsequent attacks such as fuzzing and spoofing [6]. 2) The unauthorized access gained by attackers to the owner's private data and associated permissions represents a grave threat. Such access has the potential to compromise the privacy and security of not only the individuals but also their vehicles [7]. Second, the lack of authentication in the CAN bus allows unauthorized devices to join the CAN and broadcast messages to other listening controllers. This inherent vulnerability provides attackers with an exploitable path. By gaining access to the CAN bus, attackers can send deceptive messages through the CAN bus.

Vulnerabilities of the CAN bus have led to a rise in vehicle attacks [8]. In 2016, Savage and his team successfully manipulated the braking system and windshield wipers of a Chevrolet Corvette through a commercial remote message processing control device [9]. This attack demonstrated the potential for aftermarket devices to exploit vulnerabilities in the CAN bus, beyond the control of vehicle manufacturers. Nie et al. achieved remote attacks on a Tesla Model S using wireless and cellular interfaces in the same year. Keen Security Lab of Tencent discovered multiple attack surfaces in BMW vehicles, highlighting that even high-end commercial vehicles cannot completely evade network attacks [10]. In 2012, Palanca et al. targeted an Alfa Romeo Giulietta by applying a distributed denial-of-service (DoS) attack through the onboard diagnostics (OBD) port [11].

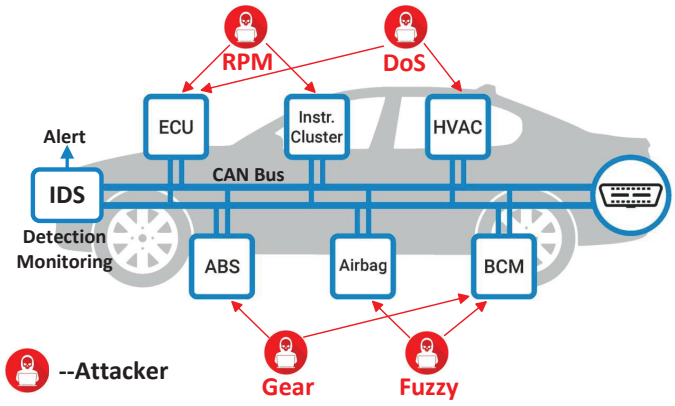

Fig. 1. The Application of In-vehicle IDS

To protect the CAN bus, several mitigation measures have been proposed in past research, such as physically isolating critical ECUs [12], implementing access control strategies

S. Li, Y. Cao (corresponding author), and Hassan Jalil Hadi are with the School of Cyber Science and Engineering, Wuhan University, Wuhan 430000, China. (e-mail: sifan.li@whu.edu.cn, yue.cao@whu.edu.cn, hjh-whu@whu.edu.cn).

F. Hao is with the Department of Computer Science, University of Warwick, CV4 7AL, UK. (e-mail: feng.hao@warwick.ac.uk).

Faisal Bashir Hussain is with the Department of Computer Science at Bahria University, Islamabad, Pakistan. (e-mail: fbashir.buic@bahria.edu.pk).

L. Chen is with ETIS UMR8051, ENSEA, CY Cergy Paris University, CNRS, F-95000, Cergy, France

[13], and employing data encryption [14]. However, these measures primarily aim to minimize the attack surface before an attack occurs, and become ineffective once an attack takes place. To address this issue, deploying an intrusion detection system (IDS) can enable real-time monitoring of in-vehicle traffic and issuing warnings when necessary (see Fig. 1).

In-vehicle IDSs can be classified into the following categories based on detection techniques. 1) Entropy-based detection – This approach identifies attack traffic by comparing the current information entropy with the threshold range of normal states. It offers high detection efficiency and minimal utilization of in-vehicle computational resources [15]. However, it suffers from a tendency of high false-positive rates due to its heavy reliance on threshold calculation and selection. 2) Feature-based detection – This method determines anomalies by comparing already running traffic information in vehicles with existing feature sets [16]. It is simple, practical, provides fast detection, and delivers high accuracy. However, it cannot detect unknown attacks and may exhibit temporal lag. 3) Deep learning-based detection – This approach leverages effective algorithms to automatically extract data features, compare them with the desired traffic for detection, and provide conclusive judgments [17]. It can achieve excellent detection results for both known and unknown attacks, ensuring high accuracy. However, its complexity often demands higher device resources (computational performance).

Considering the above issues, the proposed enhanced cuckoo filter-based intrusion detection system (ECF-IDS) is based on deep learning technology combined with the ECF technology. The main innovations are as follows:

- Based on the cuckoo filter, this paper proposes an enhanced cuckoo filter (ECF). It firstly establishes two storage spaces (a normal list and an intrusion list) based on two public datasets (CHD, can-train-and-test dataset). Then, the CAN traffic is checked against the two lists to determine if the traffic exists. If the results are contradictory (the input traffic exists in two lists) or unknown (the input traffic does not exist in two lists), subsequent judgment is performed using the Bidirectional Encoder Representation from Transformers (BERT) model with generated results also fed back to the two filters.
- This study adopts the BERT model for binary and multi-classification of CAN bus traffic. Considering the environment of in-vehicle IDS, it is important not to consume excessive in-vehicle computing resources. The ECF-IDS not only achieves efficient detection of unknown and known traffic, but also achieves lightweight implementation, so as to save in-vehicle computing resources and reduce detection time.
- Additionally, the testing of the ECF-IDS model is divided into two parts: offline training and online testing. Offline training involves using traditional CHD and the latest can-train-and-test datasets for model training. Subsequently, online testing is performed using the real in-vehicle environment to conduct real-time evaluations of the ECF-IDS model.

The rest of this paper is organized as follows. Section II lists related works about in-vehicle IDSs with comparative summarization. Section III introduces the background of CAN, dataset and in-vehicle attacks. The framework of ECF-IDS is presented in section IV. Section V presents the experiment setup, datasets, evaluation metrics and performance evaluation. Section VI finally concludes this paper.

II. Related Work

The CAN bus protocol has been in existence for over forty years. With the accelerated development of connected and intelligent vehicles, the in-vehicle network has become a prime target for attacks, leading to an increasingly severe security problem [27]. However, the development of in-vehicle IDS is still in its infancy, and there is a lack of related literature and research in this field. In Table I, we summarize the relevant works in the literature on in-vehicle IDS.

An efficient IDS based on the bloom filter was proposed for CAN [18]. Leveraging the effective time-memory tradeoff of the bloom filter, it is capable of testing the periodicity, identifiers, and data fields of message frames in the CAN bus. Furthermore, it has been implemented and tested in a real in-vehicle environment, demonstrating its suitability for in-vehicle bus systems. In another study, Song et al. introduced a Reduced Inception-ResNet model-based IDS [19] for protecting the CAN bus. This model effectively learns the patterns of CAN traffic, resulting in high detection performance, and is particularly notable for its ability to handle large datasets. The CAN-ADF [20] is an ensemble framework that combines rule-based and recurrent neural network (RNN) based models to detect common intrusions on the CAN bus, e.g., DoS, fuzzing, and replay attacks. This framework emphasizes the employment of RNN structures and multi-classification models.

Several studies have focused on utilizing time series prediction (TSP) and spatial-temporal features to detect anomalous traffic in the context of IDS. For example, a hybrid deep learning-based IDS (HyDL-IDS) [21] was introduced. This model integrates convolutional neural network (CNN) and long short-term memory (LSTM) structures to leverage both spatial and temporal representations of traffic in an in-vehicle network (IVN). The CANnolo [2] was proposed to identify anomalies in CAN. It employs LSTM-autoencoders to analyze the time and data sequences of CAN. In addition, Qin [22] proposed an intrusion detection method that utilizes anomaly analysis based on TSP, using an LSTM structure to analyze each data field of CAN messages. This approach achieved outstanding detection performance by effectively capturing the temporal patterns of IVN traffic.

The BERT model, known for its excellent performance in natural language processing (NLP), can be applied to intrusion detection on the CAN bus. The CAN-BERT [23] is a model that leverages learning arbitration ID sequences for intrusion detection on the CAN bus. It was evaluated using the "Car Hacking: Attack & Defense Challenge 2020" dataset, achieving F1-Scores ranging from 0.81 to 0.99. Another BERT-based IDS model is CANBERT [24], which was tested using the OTIDS dataset and achieved an accuracy rate of over 0.99.

Utilizing a CNN as the base model in combination with other methods or models is also a promising choice for intru-

TABLE I
COMPARISON OF RELATED WORK

Related Work	Type of IDS	Datasets	Method	Real In-vehicle Environment Test
[18]	Bloom filter-based IDS	Private Dataset	Bloom Filter	Yes
[19]	Inception-ResNet-based IDS	CHD	DCNN,Inception-ResNet	Yes
[20]	RNN-based IDS	Private Dataset	RNN, LSTM	Yes
[21]	Hybrid Deep Learning-IDS	CHD	CNN, LSTM	No
[2]	LSTM-based IDS	ReCAN	LSTM (autoencoders)	No
[22]	Anomaly-based IDS	Private Dataset	LSTM	Yes
[23]	BERT-based IDS	Car Hacking: Attack & Defense Challenge 2020	BERT	No
[24]	BERT-based IDS	OTIDS	BERT	No
[25]	Deep Learning IDS	OTIDS	CNN, GRU	No
[26]	Recurrence Plots IDS	CHD,Private Dataset	Recurrence Plots, CNN	Yes

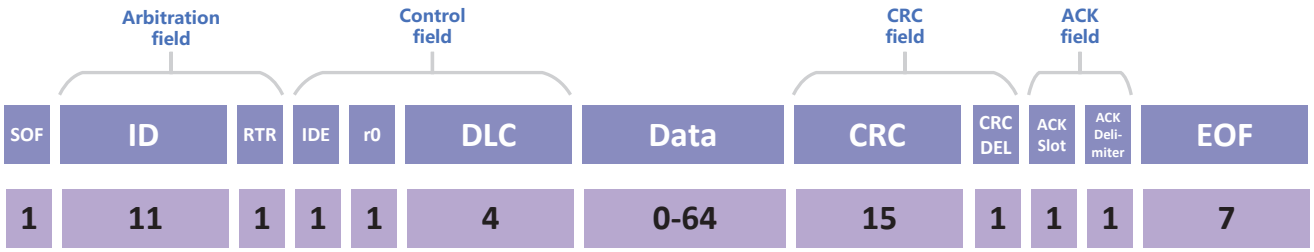

Fig. 2. The Structure of CAN Frame

sion detection. The CANintelligentIDS model [25] integrated CNN and attention-based Gate Recurrent Unit (GRU) to detect attacks. It consists of two types of detection: analyzing data sequences to detect individual attacks and considering real-world scenarios to detect blended attacks in the form of custom vectors. The fusion of recurrence plots with CNN for in-vehicle intrusion detection represents a favorable choice. A Rec-CNN model [26] employed CNN to train images generated by the recurrence plots algorithm using CAN traffic. The recurrence plots algorithm facilitates the Rec-CNN model in capturing the dependencies and correlations between arbitration IDs.

III. BACKGROUND

A. CAN Bus

The main objective of ECF-IDS is to effectively protect the CAN bus from external attacks. Therefore, it is essential to have a prior understanding of CAN bus and its components before establishing the ECF-IDS model. This section will primarily focus on introducing the CAN bus protocol.

The vehicle industry has been flourishing, resulting in an increasing number of ECUs in vehicles. The exchange of signals between these ECUs leads to an expansion in the number of wiring harnesses in vehicles [28]. This poses a growing contradiction between the complex wiring harnesses and the limited wiring space in vehicles. The abundance of wiring harnesses not only reduces the reliability of electrical system, but also increases the overall weight of vehicle.

To address these challenges, the CAN bus is employed to interconnect various ECUs within a vehicle, forming a local area network [29]. This enables information sharing and substantially reduces the complexity of wiring harnesses. In order to mitigate data collision on the bus, the CAN utilizes the concept of priority filtering by assigning higher priority to specific IDs for message transmission. This ensures reliable data transmission even when multiple nodes are simultaneously sending data. Along with the ID, a CAN frame is composed of segments such as start of frame (SOF), data length code (DLC), data, cyclic redundancy check (CRC), acknowledgement (ACK), and end of frame (EOF). The specific structure of a CAN frame is illustrated in Fig. 2.

A CAN data frame consists of seven segments, each with its specific meaning. These segments are defined as follows:

- **SOF:** This segment signals the beginning of a CAN frame and serves as a synchronization marker for all nodes on the bus.
- **ID:** The ID segment carries information about the source and priority of a message. It distinguishes different types of messages and facilitates message filtering.
- **DLC:** The DLC segment indicates the length of data payload in the frame. It specifies the number of bytes included in the CAN message.
- **Data:** The Data segment contains the actual payload or information that is being transmitted. This segment can vary in size, depending on the DLC value specified in the previous segment.
- **CRC:** The CRC segment is used for error detection and ensures the integrity of transmitted data. It enables the receiving ECU nodes to identify and correct potential errors in the received message.

- **ACK:** The ACK segment is sent by the receiving ECU node to acknowledge the successful reception of a valid CAN frame. It confirms the error-free transmission of a message.
- **EOF:** This segment marks the end of a CAN frame and allows the bus to revert to an idle state, ready for the transmission of the next frame.

B. Car Hacking Dataset

The Car Hacking Dataset (CHD) [30] is a comprehensive collection of real-world in-vehicle network data specifically curated for research and development purposes within the domain of in-vehicle IDSs. This dataset encompasses diverse attack scenarios that pertain to the vehicle environment. It includes a range of network traffic data, sensor readings, and control messages, reflecting the interactions within a vehicle's ECUs and various onboard systems.

The dataset provides a valuable resource for studying the security vulnerabilities of modern vehicles and developing robust intrusion detection algorithms. Researchers and practitioners can utilize this dataset to analyze different attack patterns, test intrusion detection approaches, and assess the effectiveness of various defensive mechanisms in mitigating potential threats. The CHD contributes to advancing the field's understanding of car hacking and enhancing the security of in-vehicle networks.

C. Can-train-and-test Dataset

The can-train-and-test dataset [31] contains CAN data from four different vehicles produced by two different manufacturers. It includes equivalent attack captures for each vehicle model, allowing researchers to test the generalization ability of IDS across various vehicle models and manufacturers. The dataset includes replayable .log files, labeled and unlabeled .csv files, and offers nine different types of attacks, ranging from DoS to gear spoofing. Some attacks were conducted during live on-the-road experiments with real vehicles and have known physical impacts. Researchers can use this dataset to benchmark machine learning IDSs and contribute to open-access datasets to address gaps in existing resources.

D. In-Vehicle Attacks

As depicted in Fig. 1, there are four primary attacks posing threats to the in-vehicle network. DoS attacks primarily target the heating ventilation and air conditioning (HVAC) and ECU systems, while fuzzy attacks are primarily directed towards the anti-lock brake system (ABS) and airbag systems. Gear attacks predominantly aim at the ABS and body control module (BCM) systems, while revolutions per minute (RPM) attacks mainly focus on the ECU and instrument cluster.

1) DoS Attack: In this attack, malicious attackers aim to disrupt the normal operation of a vehicle's interior electronic systems. This attack typically entails sending messages without specifying valid CAN IDs or frequently changing the CAN ID, which can severely compromise the integrity of the in-vehicle network. Such actions may result in overloading the internal communication buses or systems, leading to malfunctions or unresponsiveness in critical vehicle components, such as infotainment systems and climate control.

2) Fuzzy Attack: A fuzzy attack involves exploiting uncertainties in the data received by the in-vehicle sensors or control systems. Attackers manipulate sensor inputs to introduce ambiguity and confusion into the vehicle's decision-making processes. Fuzzy attacks can manifest in various ways, one of which is injecting random or ambiguous data, such as random IDs with unpredictable payloads. The idea behind a Fuzzy attack is to take advantage of the lack of precise and unambiguous information in the data flow, which can lead to erratic behavior in the vehicle's control systems. These attacks can potentially compromise the safety and security of the vehicle by creating confusion or misinterpretation of sensor data.

3) Gear Attack: A gear attack targets the vehicle's transmission system. Malicious manipulation of the transmission controls could result in sudden and unexpected gear changes, leading to loss of vehicle control, reduced acceleration, or even mechanical damage.

4) RPM Attack: The RPM attack involves tampering with the engine's RPM data reported by sensors. By providing inaccurate RPM readings, attackers can mislead the vehicle's ECUs, leading to improper fuel injection or timing. This can cause engine performance issues, reduced fuel efficiency, and potentially lead to mechanical damage.

IV. Proposed ECF-IDS Framework

The ECF-IDS combines an ECF algorithm with the BERT model. The main process is illustrated in Fig. 3, where the CAN traffic of target classification is initially inputted. Subsequently, it undergoes a comparison with both a normal list and an intrusion list. The CAN traffic that results in consistent "Yes" or "No" comparisons is then passed into the IDS based on BERT for subsequent detection. The normal traffic identified by the IDS is stored in the normal list, while the intrusion traffic is stored in the intrusion list and subjected to attack classification.

A. Enhanced Cuckoo Filter

In the domain of ECF-IDS, the ECF can be considered as an upgraded version of cuckoo filters [32]. It not only provides the ability to delete elements, but also achieves higher query efficiency and space utilization. Here, the ECF consists of two hash functions to store the labeled traffic. For a given input of labeled CAN traffic t, the fingerprint information f_t and hash values $h1$ and $h2$ are computed as follows:

$$f_t = fingerprint(t) \tag{1}$$

$$h1 = Hash(t) \tag{2}$$

$$h2 = h1 \oplus Hash(f_t) \tag{3}$$

Where $fingerprint()$ is the fingerprint function, $Hash()$ is the hash function, and $\oplus$ represents exclusive or (XOR) operation.

As depicted in Algorithm 1 and Algorithm 2, the ECF has two lists: a normal list (TN) and an intrusion list (TI). It first computes the fingerprint information for each element (c_i is the $i-th$ element in TN, and d_j is the $j-th$ element in TI)

Algorithm 1 Establish Normal List

Input: CAN normal traffic set (TN)
Output: $normal\_list$

1: **for** c_i in TN **do**
2: $fc_i = fingerprint(c_i)$
3: **if** fc_i in $normal\_list$ **then**
4: Continue
5: **else**
6: hc_i1 = $Hash(c_i)$
7: **if** hc_i1 has available slots **then**
8: add fc_i in $normal\_list$
9: $n\_filtersize$+1
10: **else**
11: $hc_i2 = hc_i1 \oplus Hash(fc_i)$
12: **if** hc_i2 has available slots **then**
13: add fc_i in $normal\_list$
14: $n\_filtersize$+1
15: **else**
16: **while** $n\_filtersize < \sigma 1$ **do**
17: remove one element fc_s of hc_i1's slots from $normal\_list$
18: $hc_s2 = hc_s1 \oplus Hash(fc_s)$
19: **if** hc_s2 has available slots **then**
20: add fc_s in $normal\_list$
21: $n\_filtersize$+1
22: break
23: **else**
24: return 0
25: **end if**
26: **end while**
27: **if** $n\_filtersize \geq \sigma 1$ **then**
28: expand $\sigma 1$
29: **end if**
30: **end if**
31: **end if**
32: **end if**
33: **end for**
34: **return** $normal\_list$

Algorithm 2 Establish Intrusion List

Input: CAN intrusion traffic set (TI)
Output: $intrusion\_list$

1: **for** d_j in TI **do**
2: $fd_j = fingerprint(d_j)$
3: **if** fd_j in $intrusion\_list$ **then**
4: Continue
5: **else**
6: hd_j1 = $Hash(d_j)$
7: **if** hd_j1 has available slots **then**
8: add fd_j in $intrusion\_list$
9: $i\_filtersize$+1
10: **else**
11: $hd_j2 = hd_j1 \oplus Hash(fd_j)$
12: **if** hd_j2 has available slots **then**
13: add fd_j in $intrusion\_list$
14: $i\_filtersize$+1
15: **else**
16: **while** $i\_filtersize < \sigma 2$ **do**
17: remove one element fd_s of hd_j1's slots from $intrusion\_list$
18: $hd_s2 = hd_s1 \oplus Hash(fd_s)$
19: **if** hd_s2 has available slots **then**
20: add fd_s in $intrusion\_list$
21: $i\_filtersize$+1
22: break
23: **else**
24: return 0
25: **end if**
26: **end while**
27: **if** $i\_filtersize \geq \sigma 2$ **then**
28: expand $\sigma 2$
29: **end if**
30: **end if**
31: **end if**
32: **end if**
33: **end for**
34: **return** $intrusion\_list$

(line 2). It then searches for the presence of this fingerprint information within the normal list or intrusion list (line 3). If it exists, the next traffic is processed (line 3-4). If it does not exist, the element undergoes a hashing operation to calculate the first storage position hc_i1 (or hd_j1) (line 6). This position contains four vacant slots, and if any of these slots is available, the fingerprint information of that element is stored (line 7-8). The filter size of the normal or intrusion list ($n\_filtersize$ or $i\_filtersize$) is increased by 1 (line 9). However, if all four slots are occupied, an XOR operation is performed between the fingerprint and the first hash coordinate to calculate the second coordinate hc_i2 (or hd_j2) (line 11).

1) Case 1: If there is an available slot at the second coordinate, the fingerprint information of the element is stored (line 12-13). The filter size of the normal list or intrusion list ($n\_filtersize$ or $i\_filtersize$) is increased by 1(line 14).

2) Case 2: If there is still no available slot, one of the fingerprints in a randomly chosen occupied slot (fc_s or fd_s) is evicted to make room for the new element (line 17). The evicted fingerprint then calculates its second coordinate hc_s2 (or hd_s2), and the process is repeated to check for available slots (line 18-24). This continues until a threshold $\sigma 1$ (or $\sigma 2$) is reached, at which point the cuckoo filter undergoes expansion (line 27-28). The construction of the intrusion list (Algorithm 2) follows a similar process, but the aforementioned algorithm is applied to the intrusion list instead of the normal list. Finally, both the normal list and intrusion list are obtained.

After obtaining the two lists, the ECF subsequently processes the set of CAN traffic to be detected, denoted as T, as described in Algorithm 3. The input traffic t_i (the $i-th$ element in T) is initially processed to compute its fingerprint information (line 2). Subsequently, it undergoes matching with the normal list, and regardless of the outcome, a matching process with the intrusion list follows (line 3-4). The traffic that does not match with both the normal list and the intrusion list (line 13), as well as traffic that matches with both lists (line

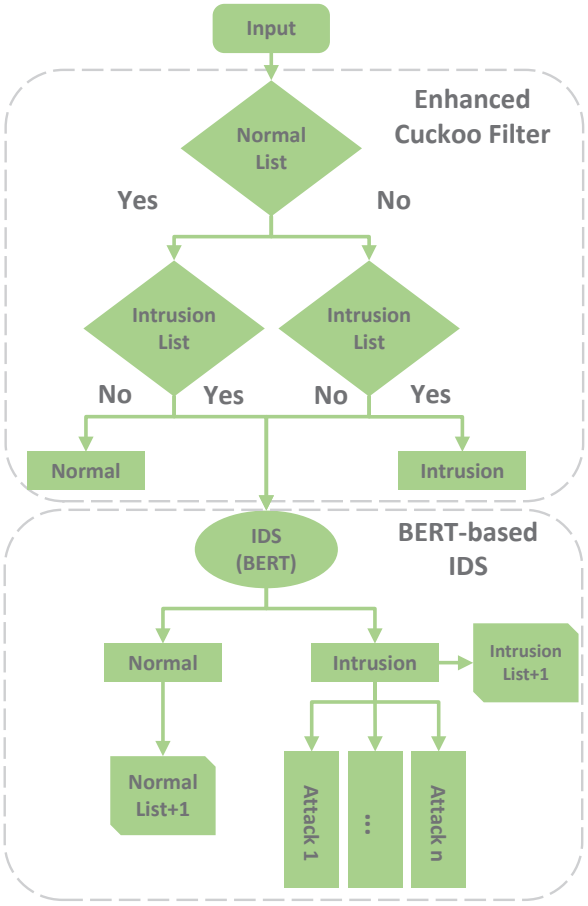

Fig. 3. The Process of ECF-IDS

5), is sent to the subsequent IDS based on BERT. The traffic that matches with either one of the lists is allocated with the corresponding label of "normal" or "intrusion" (line 7, 11).

Compared to the traditional cuckoo filter, the ECF offers several advantages. Firstly, it prohibits the existence of duplicate traffic data within the filter, thereby saving storage space. Secondly, it employs a dual-filter combination approach to establish both a normal list and an intrusion list, enhancing the accuracy of query results. Additionally, based on the judgment results provided by the subsequent IDS using BERT, the filter is updated and data is inserted accordingly.

B. BERT-based IDS

BERT, as one of the commonly used models in NLP, has the ability to capture contextual relationships and perform subsequent tasks [33], [34]. It can learn the potential associations between adjacent CAN traffic to aid in the judgment and classification of traffic. To capture these contextual relationships, the BERT is a pre-trained language representation model that employs the MLM to generate deep bidirectional language representations. As illustrated in Fig. 4, for the CAN traffic set t, where $t(i)$ represents the ith specific CAN traffic within the set, the traffic $t(i)$ is prefixed with [CLS], while [SEP] is inserted as a delimiter between each

Algorithm 3 Double Lists Filtering

Input: CAN traffic set T (to be detected)
Output: output "normal"/"intrusion"

```
1:  for t_i in T do
2:      f_i = fingerprint(t_i)
3:      if f_i in normal_list then
4:          if f_i in intrusion_list then
5:              feed t_i into BERT-IDS
6:          else
7:              output "normal"
8:          end if
9:      else
10:         if f_i in intrusion_list then
11:             output "intrusion"
12:         else
13:             feed t_i into BERT-IDS
14:         end if
15:     end if
16: end for
```

subsequent traffic. The remaining traffic is transformed into corresponding token representations. Subsequently, a learnable segment embedding is added for each token representation to indicate whether it belongs to CAN traffic $t(i)$ or $t(i+1)$. Finally, the corresponding position embedding is appended, completing the input pre-processing for BERT.

1) Pre-training: As shown in Fig. 5, the BERT model consists of two main steps: pre-training and fine-tuning. During the pre-training phase, the MLM model is employed to randomly replace 15% of the pre-processed tokens with a masked token ([MASK]). These tokens are then used as input into the encoder layer to predict the original tokens. As illustrated in Fig. 6, the encoder layer consists of two components: multi-head attention mechanism [35] and feed-forward neural network [36]. The pre-processed token $Y(i)$ is fed into the encoder layer. Since BERT-base contains L layers of encoders, $\mathbf{Y}^{(i,l)}$ represents the $i-th$ token input to the $l-th$ layer of encoder. After each sub-layer, layer normalization is applied with a residual connection incorporated. Thus, a residual connection is utilized around each sub-layer followed by layer normalization as follows:

$$\mathbf{H}^{(i,l)} = h(\mathbf{Y}^{(i,l)}) + nor\left(\mathbf{Y}^{(i,l)} + h(\mathbf{Y}^{(i,l)})\right) \tag{4}$$

$$\mathbf{F}^{(i,l)} = f(\mathbf{H}^{(i,l)}) + nor\left(\mathbf{H}^{(i,l)} + f(\mathbf{H}^{(i,l)})\right) \tag{5}$$

$$\mathbf{Y}^{(i,l+1)} = \mathbf{F}^{(i,l)}, \forall l < L \tag{6}$$

where $\mathbf{H}^{(i,l)}$ represents the output of the first sub-layer for the $l-th$ encoder layer, while $\mathbf{F}^{(i,l)}$ represents the output of the second sub-layer for the $l-th$ encoder layer. The function h refers to the multi-head attention function, while the function of f corresponds to the position-wise feed-forward function. Additionally, the function of nor denotes the layer normalization function.

For the multi-head attention function h, the query Q, key K, and value V are computed based on the input $\mathbf{Y}^{(i,l)}$ of the

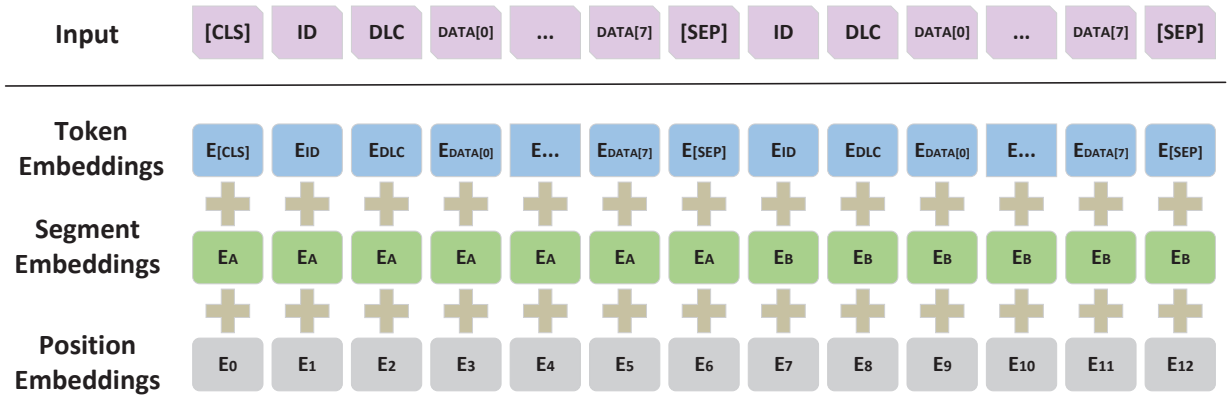

Fig. 4. The Embedding of CAN

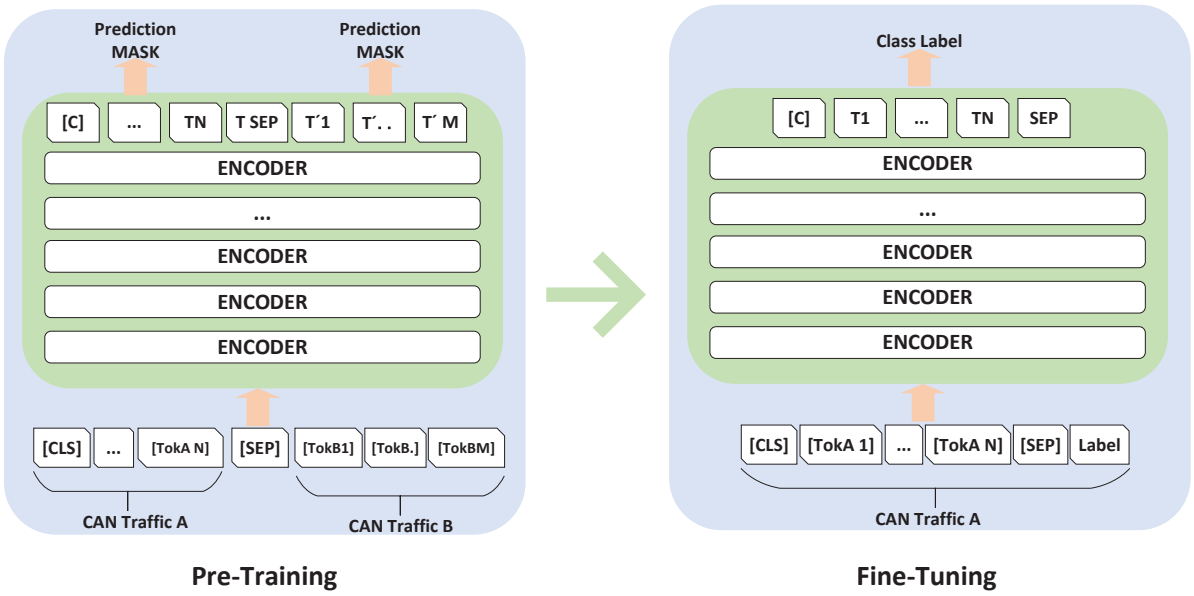

Fig. 5. The Main Process of BERT

$i-th$ CAN traffic sequence in the $l-th$ layer of the encoder. The equations are as follows:

$$\mathbf{Q}^{(i,l)} = \mathbf{Y}^{(i,l)}\mathbf{W}^{(Q)} \tag{7}$$

$$\mathbf{K}^{(i,l)} = \mathbf{Y}^{(i,l)}\mathbf{W}^{(K)} \tag{8}$$

$$\mathbf{V}^{(i,l)} = \mathbf{Y}^{(i,l)}\mathbf{W}^{(V)} \tag{9}$$

where W corresponds to the respective trainable weight matrices. The attention function is then computed based on Q, K, and V as follows:

$$\text{Attn}\left(\mathbf{Q}^{(i,l)}, \mathbf{K}^{(i,l)}, \mathbf{V}^{(i,l)}\right) = \sigma\left(\frac{\mathbf{Q}^{(i,l)}\mathbf{K}^{(i,l)T}}{\sqrt{d}}\right)\mathbf{V}^{(i,l)} \tag{10}$$

where σ refers to the softmax function, T denotes the transposed matrix, and d represents the dimension of $\mathbf{Q}^{(i,l)}, \mathbf{K}^{(i,l)}, \mathbf{V}^{(i,l)}$ vectors. Finally, the attention results from multiple heads are transformed through a linear transformation to obtain the final output $h\left(\mathbf{Y}^{(i,l)}\right)$:

$$head^{(i,l)} = Attn\left(\mathbf{Q}^{(i,l)}, \mathbf{K}^{(i,l)}, \mathbf{V}^{(i,l)}\right) \tag{11}$$

$$h(\mathbf{Y}^{(i,l)}) = linear(head^{(i,l)}) \tag{12}$$

where $linear$ refers to the linear transformation function.

The inspiration for the design of a feed-forward network comes from the fully connected layers in CNN. However, in contrast to fully connected layers, positional feed-forward networks adopt a special structure. In pre-training, a positional feed-forward network consists of two fully connected layers, which are connected through a non-linear activation function (specifically the ReLU activation function), capturing local features in the sequence. Specifically, for an input element

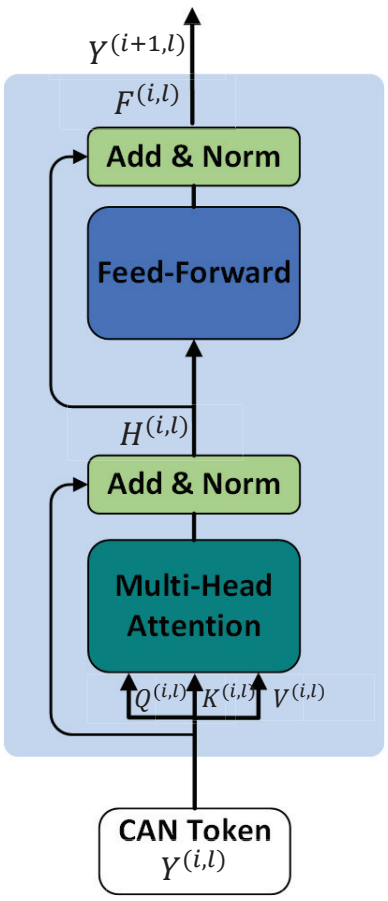

Fig. 6. The Structure of Encoder Layer

$\mathbf{h}^{(i,l)}$, the positional feed-forward network can be represented by the following formula:

$$FFN(X) = max(0, \mathbf{h}^{(i,l)} \cdot W_1 + b_1) \cdot W_2 + b_2 \quad (13)$$

where W_1 and b_1 are the weight matrix and bias vector of the first fully connected layer, while W_2 and b_2 are the weight matrix and bias vector of the second fully connected layer. The "·" symbol denotes matrix multiplication, and $max\,(0, \cdot)$ represents the element-wise ReLU activation function. After these steps, $\mathbf{F}^{(i,L)}$ is passed through a linear layer followed by the softmax function to activate, and the optimal result is selected as the prediction ($PredictionMask$) for the given [MASK]:

$$PredictionMask = \sigma[linear(\mathbf{F}^{(i,L)})] \quad (14)$$

2) Fine-tuning: Fine-tuning involves using the pre-trained BERT model as initial parameters. Then, in order to adapt the ECF-IDS model, additional training is performed on the labeled classification task. As the CAN traffic in pre-training does not contain labels, this process falls under unsupervised training. However, during fine-tuning, the input CAN traffic is labeled, making it a supervised training process. The main steps of fine-tuning are outlined as follows:

- **Adding Task-Specific Layers:** In order to adapt the BERT model for specific tasks, it is necessary to add a layer that is specific to the prediction classification task on top of the pre-trained BERT model. This classification task can be binary (normal, intrusion) or multi-class (normal, attack 1, attack 2, ...). Therefore, this layer is a fully connected layer used to extract task-related features from the output of BERT.
- **Parameter Initialization:** After adding the task-specific layers, the parameters of these layers need to be initialized using the pre-trained BERT model from earlier.
- **Fine-Tuning Training:** The entire model is trained using the task dataset. In each training step, the task data is fed into the model, and the loss between the model's output and true labels is calculated. The model parameters are then updated using back-propagation and an optimization algorithm to minimize the loss.
- **Model Evaluation:** After the completion of Fine-Tuning training, the tuned model is evaluated using a test set. Task-specific evaluation metrics such as accuracy, F1-Score, etc., are used to measure the performance of the model.

Regarding binary and multi-class tasks, the key difference lies in the final output layer. Binary tasks utilize a sigmoid layer, while multi-class tasks utilize a linear layer followed by the softmax function. The sigmoid function used in binary classification is shown below:

$$s(x) = \frac{1}{1 + e^{-x}} \quad (15)$$

where e represents the base of natural logarithm, and x denotes the input value. $s(x)$ has the following characteristics: when x approaches negative infinity, $s(x)$ approaches 0; When x approaches positive infinity, $s(x)$ approaches 1; When $x = 0, s(x) = 1/2$. The representation for the linear layer + softmax in multi-class tasks is similar to the mentioned Eq. (14) earlier, with the only difference being the output.

V. Performance Evaluation

In order to evaluate the performance of ECF-IDS, the hardware and software environment is described in Table II. The choice of the AMD Threadripper and RTX 4090 in the experimental setup is primarily driven by practical considerations during the offline model training phase. Our ECF-IDS model is indeed initially trained offline on a high-performance server infrastructure due to its computational intensity and resource requirements. However, it is essential to emphasize that the deployment of the ECF-IDS within the vehicle does not involve the same hardware configuration used during training. Once the ECF-IDS model is trained and optimized, it is intended for deployment on hardware and systems that are more appropriate and representative of in-vehicle network controllers.

TABLE II
Experiment Setup

Hardware	Software
AMD Ryzen Threadripper PRO 5995WX Nvidia RTX4090(24GB) 64GB(RAM)	Jupyterlab 3.4.4 Python 3.10.6

A. Datasets

The ECF-IDS utilizes the widely-used dataset CHD [30] and can-train-and-test. The CHD was constructed by logging CAN traffic through the OBD-II port of a real vehicle. The logging was done during the execution of message injection attacks. The dataset consists of normal traffic and four types of attack traffic (DoS attack, Fuzzy attack, RPM gauze attack, and Drive Gear attack). 12 attributes of CHD are presented in Table III.

TABLE III
THE ATTRIBUTES OF CHD

Attribute	Explaination
Timestamp	recorded time (s)
CAN ID	identifier of CAN message in HEX (ex. 043f)
DLC	number of data bytes, from 0 to 8
DATA[0-7]	data value (byte)
Flag	T or R, T represents injected message while R represents normal message

The can-train-and-test [31] was collected from on-road data vary four different vehicles and six different drivers to create a diverse CAN dataset for machine learning. This dataset accessed CAN traffic data through the OBD-II port of each vehicle. It utilized a Korlan USB2CAN cable to connect to the OBD-II port, which converted raw OBD-II data into USB data, enabling communication with the OBD-II port through Linux's SocketCAN subsystem and can-utils utilities. In Table IV, 4 attributes of can-train-and-test are presented.

TABLE IV
THE ATTRIBUTES OF CAN-TRAIN-AND-TEST

Attribute	Explaination
timestamp	recorded time (s)
arbitration_id	identifier of CAN message in HEX (ex. 043f)
data_field	data value (byte)
attack	1 or 0, 1 represents injected message while 0 represents normal message

The normal list used in the ECF-IDS is constructed based on the normal traffic file (normal_run_data.txt) from the CHD and can-train-and-test datasets. After removing duplicate samples, a total of 306,767 CAN traffic samples are added to the normal list. The intrusion list is constructed based on the attack traffic (Flag=T or attack=1) from the remaining attack files. After removing duplicate samples, a total of 491,850 CAN traffic samples are added to the intrusion list. Subsequently, the ECF-IDS processes the CHD and can-train-and-test datasets, with the training set and test set accounting for 70% and 30%, respectively. The 70% training data helps the model become proficient, and the 30% testing data evaluates its adaptability to real in-vehicle conditions.

B. Evaluation Metrics

To facilitate the comparison of performance between ECF-IDS and other algorithms, this paper primarily employs the following performance metrics: accuracy, precision, recall, F1-Score, loss, and detection time. These metrics effectively measure the prediction accuracy of IDS. Their calculation formulas are as follows:

$$Accuracy = \frac{TRP + TRN}{TRP + TRN + FAP + FAN} \tag{16}$$

$$Precision = \frac{TRP}{TRP + FAP} \tag{17}$$

$$Recall = \frac{TRP}{TRP + FAN} \tag{18}$$

$$F1 - Score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{19}$$

where TRP indicates the number of normal traffic samples that are correctly classified as normal traffic. FAN indicates the number of normal traffic samples that are misclassified as intrusion traffic. FAP indicates the number of intrusion traffic samples that are misclassified as normal traffic. TRN indicates the number of intrusion traffic samples that are correctly classified as intrusion traffic. The loss is a metric used to measure the discrepancy between predicted values and true values. The loss calculation formulas of binary and multi-class tasks are different, and the calculation formula for binary classification using cross entropy is as follows:

$$BL = \frac{1}{n_b} \sum_{i}^{n_b} BL_i \tag{20}$$

$$BL_i = -\left[y_i \log\left(p_i\right) + \left(1 - y_i\right) \log\left(1 - p_i\right)\right] \tag{21}$$

where BL is the loss of binary classification, BL_i is the loss of the $i-th$ binary classification sample. n_b is the total number of binary classification samples. y_i represents the label of the $i-th$ binary classification sample, where the positive class is denoted as 1 and the negative class is denoted as 0. p_i represents the probability of the ith binary classification sample being predicted as the positive class.

The calculation formula for the loss in multi-class classification is as follows:

$$ML = \frac{1}{n_m} \sum_{i}^{n_m} ML_i \tag{22}$$

$$ML_i = -\sum_{c}^{M} y_{ic} \log p_{ic} \tag{23}$$

where ML is the loss of multi-classification, ML_i is the loss of the ith multi-classification sample. n_m is the total number of multi-classification samples. M represents the number of classes. p_{ic} represents the predicted probability of the observed sample belonging to class c. For multi-class samples, if the true class of the sample is equal to c, y_{ic} is set to 1. Otherwise, it is set to 0. The detection time is an important metric for evaluating the efficiency of an IDS, and its calculation formula is as follows:

$$DetectionTime = \frac{AllTime}{len(data)} \tag{24}$$

where $AllTime$ represents the total time taken by the IDS to detect all samples, and $len(data)$ represents the total number of all samples.

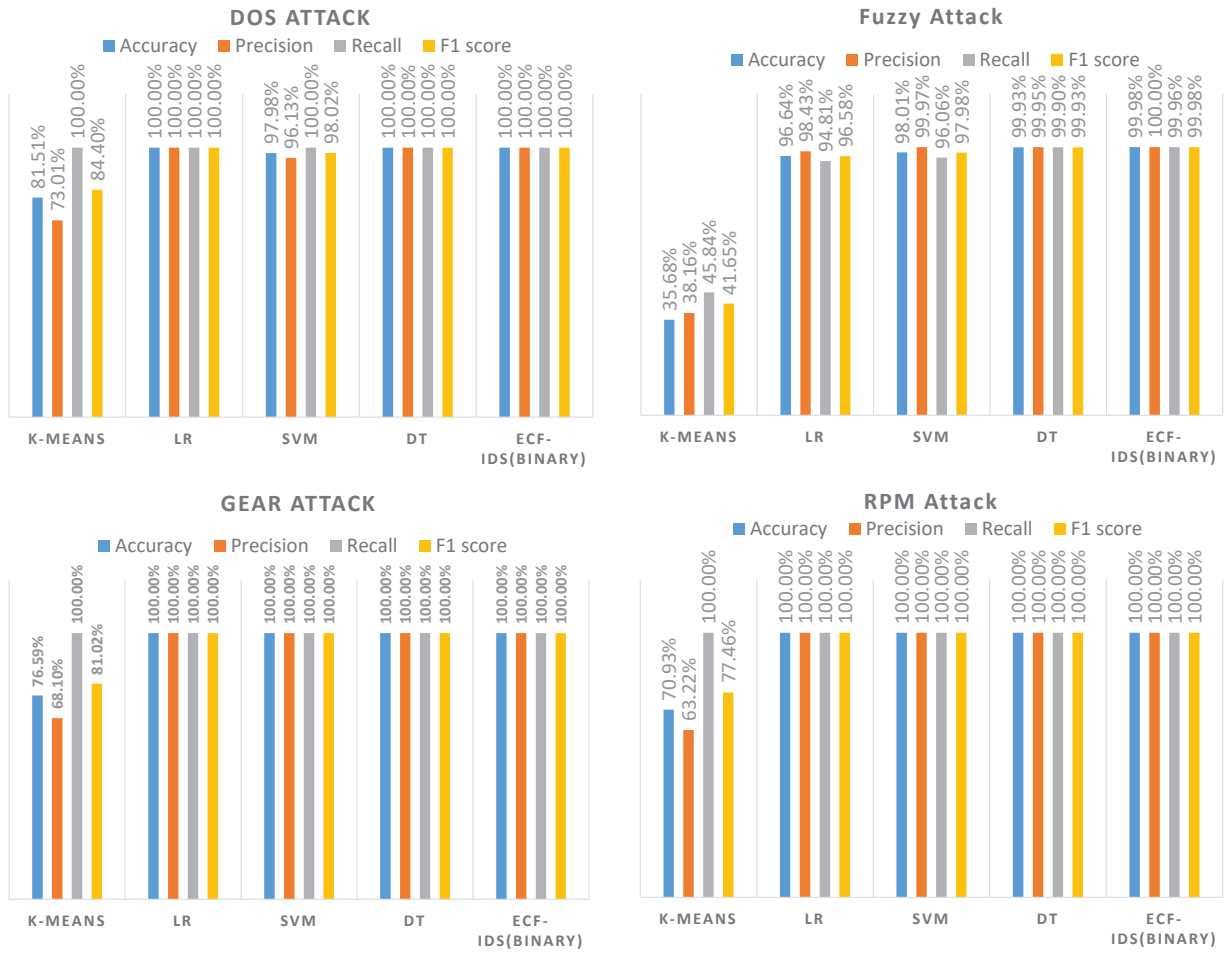

Fig. 7. Comparisons of Traditional Algorithms on Four Attacks Binary Classification (CHD)

C. ECF-IDS Experimental Result

The k-means, logistic regression (LR), support vector machine (SVM), and decision tree (DT) algorithms are compared with ECF-IDS. They utilize the same training and testing datasets, and their binary classification results for the four types of attacks are shown in Fig. 7. When faced with the four attacks, k-means performs the worst, primarily due to its tendency to converge to local optima instead of global optima, and its poor performance in clustering high-dimensional data. LR and DT demonstrate excellent performance against DoS, gear, and RPM attacks, comparable to ECF-IDS. However, under fuzzy attacks, the ECF-IDS exhibits the best performance, achieving accuracy, precision, recall, and F1-Score all above 99.96%.

The four attack datasets are merged with the normal dataset for binary classification experiments. As shown in Fig. 8, the performance of four traditional algorithms decreases compared to the previous individual attack scenarios, while only ECF-IDS maintains its performance at a high level. The corresponding ROC comparison graph (Fig. 9) confirms that ECF-IDS achieves the highest area under curve (AUC). Fig. 10 illustrates the variation of loss during the training process as a function of iteration (with each iteration processing 32 data instances), revealing a convergence to stability near iteration 500, starting from an initial value of 0.7. The final testing loss for binary classification on ECF-IDS is 0.0027.

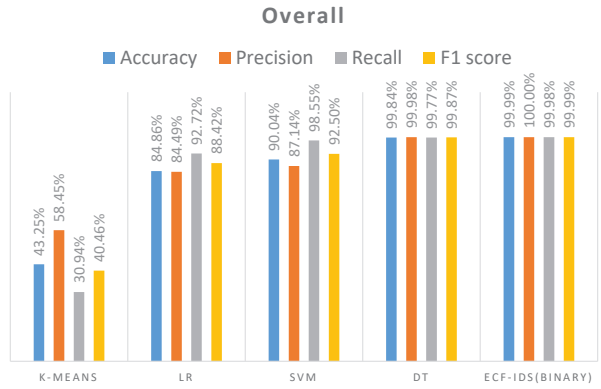

Fig. 8. Comparisons of Traditional Algorithms on Overall Attacks Binary Classification (CHD)

Multi-classification involves merging the four attack datasets with the normal dataset, requiring not only binary

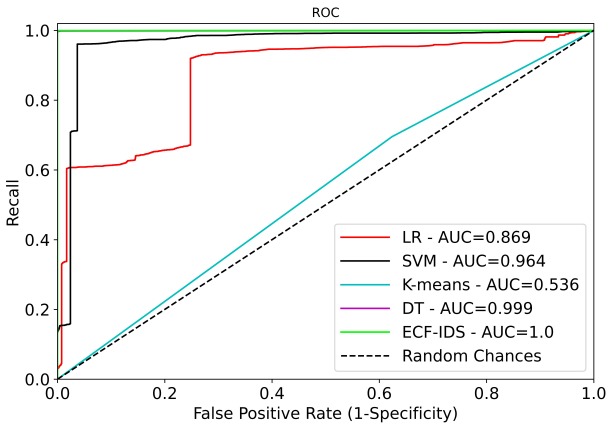

Fig. 9. ROC Curve on Overall Attacks (CHD)

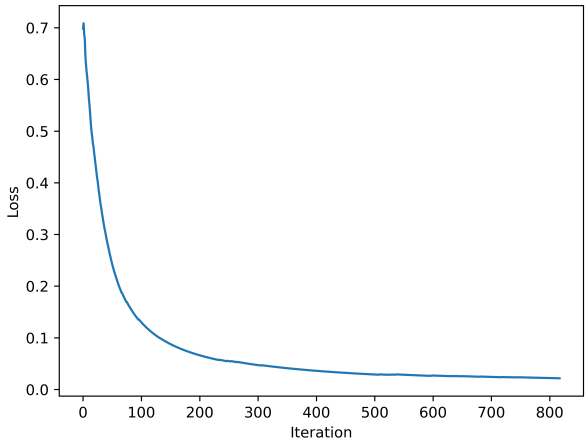

Fig. 10. The Binary Loss of Training on Overall Attacks (CHD)

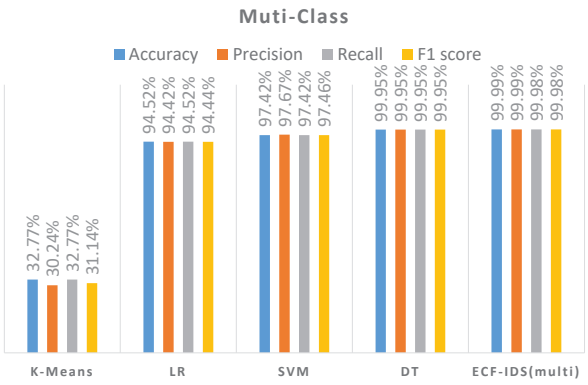

Fig. 11. Comparisons of Traditional Algorithms on Overall Attacks Multi-Classification Classification (CHD)

detection of normal and intrusion instances, but also precise classification of attacks within intrusions. As depicted in Fig. 11, the ECF-IDS outperforms the other four traditional algorithms by a wide margin. This superior performance can be attributed to the unsupervised training during the pre-training phase of the BERT model, which involves masking 15% of the data. This process provides essential parameter initialization for subsequent fine-tuning, enabling effective detection and classification of CAN traffic. As shown in Fig. 12, the training loss decreases with iterations, gradually reaching stability at around 50 iterations (with each iteration processing 32 instances). For the multi-classification task, the ECF-IDS achieves a final testing loss of 0.00035.

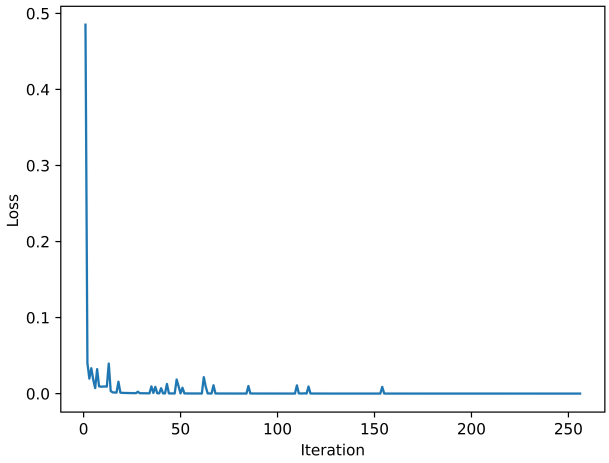

Fig. 12. The Multi-Classification Loss of Training on Overall Attacks (CHD)

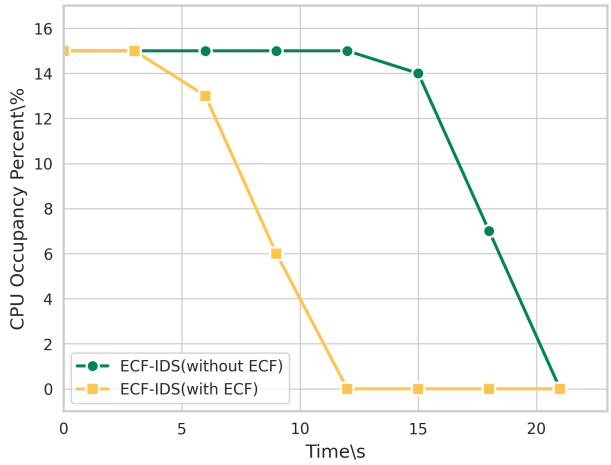

Fig. 13. The Comparison of CPU Occupancy Between ECF-IDS(with ECF) and ECF-IDS (without ECF)

For the ECF-IDS, the inclusion of ECF not only has no negative impact on the performance of the original BERT model, but also significantly reduces both the CPU consumption and detection time in vehicle applications. As illustrated in Fig. 13, the ECF-IDS (with ECF) consumes CPU resources until approximately 12 sec, while the ECF-IDS (without ECF) continues until 21 sec. Furthermore, the ECF-IDS (with ECF) achieves a detection time of 0.9 ms per traffic instance, compared to 2 ms without ECF. The addition of ECF results in an approximate 50% improvement in IDS performance.

TABLE V
COMPARISONS OF OTHER ALGORITHMS ON THE CHD DATASET (THE DATA OF HYDL, RESNET, CANBERT AND GIDS IS OBTAINED FROM THE CORRESPONDING ARTICLE UNDER THE CHD)

DoS	Accuracy	Precision	Recall	F1-Score
HyDL [21]	100%	100%	100%	100%
ResNet [19]	99.97%	100%	99.89%	99.95%
CANBERT [24]	100%	100%	100%	100%
GIDS [30]	97.9%	96.8%	99.6%	97.9%
ECF-IDS	**100%**	**100%**	**100%**	**100%**
Gear	Accuracy	Precision	Recall	F1-Score
HyDL [21]	100%	100%	100%	100%
ResNet [19]	99.95%	99.99%	99.89%	99.94%
CANBERT [24]	100%	100%	100%	100%
GIDS [30]	96.2%	98.1%	96.5%	97.3%
ECF-IDS	**100%**	**100%**	**100%**	**100%**
Fuzzy	Accuracy	Precision	Recall	F1-Score
HyDL [21]	99.98%	99.98%	99.88%	99.93%
ResNet [19]	99.82%	99.95%	99.65%	99.80%
CANBERT [24]	100%	100%	100%	100%
GIDS [30]	98%	97.3%	99.5%	98.3%
ECF-IDS	**99.98%**	**100%**	**99.96%**	**99.98%**
RPM	Accuracy	Precision	Recall	F1-Score
HyDL [21]	100%	100%	100%	100%
ResNet [19]	99.97%	99.99%	99.94%	99.96%
CANBERT [24]	100%	100%	100%	100%
GIDS [30]	98%	98.3%	99%	98.6%
ECF-IDS	**100%**	**100%**	**100%**	**100%**

TABLE VI
COMPARISONS OF DETECTION TIME (THE DETECTION TIME IN THIS TABLE IS SIMULATED AT THE SAME PLATFORM.)

Method	Detection Time(ms/traffic)
HyDL [21]	1.03
ResNet [19]	1.56
CANBERT [24]	2
GIDS [30]	3
ECF-IDS	**0.9**

TABLE VII
STORAGE SPACE FOR TWO FILTERS

	Normal	Malicious
Storage Space (bytes/1000000 traffic)	80	80

TABLE VIII
COMPARISONS OF OTHER ALGORITHMS ON THE CAN-TRAIN-AND-TEST DATASET (THE FOLLOWING DATA IS OBTAINED FROM THE CORRESPONDING ARTICLE UNDER THE SUB-DATASET 3 OF CAN-TRAIN-AND-TEST)

All data	Accuracy	Precision	Recall	F1-Score
Gaussian Naive Bayes [31]	98.05%	96.32%	98.05%	97.18%
K-Nearest Neighbor [31]	99.12%	99.83%	52.74%	69.02%
LR [31]	98.60%	98.51%	98.60%	98.23%
SVM [31]	98.96%	99.94%	44.29%	61.38%
DT [31]	99.63%	97.79%	82.13%	89.28%
Extra Tree [31]	99.52%	99.86%	74.32%	85.22%
Random Forest [31]	99.63%	99.98%	80.20%	89.00%
Multi-Layer Perceptron [31]	99.30%	99.30%	99.30%	99.22%
ECF-IDS	**99.96%**	**99.91%**	**99.84%**	**99.86%**

Compared to other algorithms proposed in previous literature, this paper also provides a comparison. However, the literature on the CHD dataset is currently limited. To control variables and ensure the use of the same dataset for comparison, we compare ECF-IDS with ResNet [19], GIDS [30], HyDL-IDS [21], and CANBERT [24], as presented in Table V. These five methods, tested on the CHD dataset, exhibit satisfactory performance. However, in the face of DoS, gear, fuzzy, and RPM attacks, the GIDS exhibits comparatively lower performance among the five models, with its F1-score dropping below 98% when dealing with DoS and gear attacks. In contrast, the CANBERT and ECF-IDS demonstrate superior performance. The CANBERT achieves a perfect score of 100% across all four metrics when confronted with the four attack types (note that results are rounded to the nearest integer). Meanwhile, the ECF-IDS falls just short of achieving 100% accuracy, recall, and F1-score when dealing with fuzzy attacks, but performs flawlessly in correctly identifying all other attack types.

To compare the efficiency of these approaches, we refer to the performance summary and comparison of current in-vehicle IDSs conducted by Wang et al. [37]. As depicted

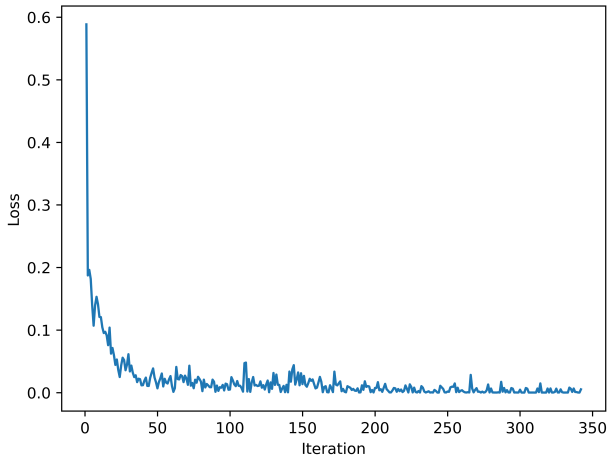

Fig. 14. The Binary Loss of Training on Overall Attacks (can-train-and-test)

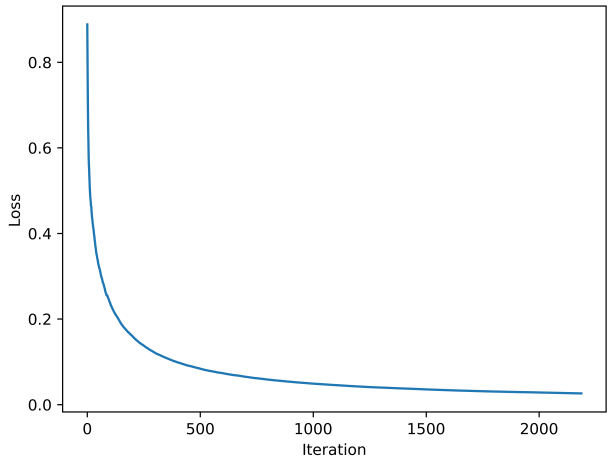

Fig. 15. The Multi-Classification Loss of Training on Overall Attacks (can-train-and-test)

in Table VI, it can be observed that the ECF-IDS performs well in detecting intrusions, and it also exhibits the highest efficiency among the five methods. Besides, Table VII displays the storage space occupied by the normal and malicious filters established by the ECF-IDS. The storage space occupied by the corresponding filters for normal and malicious filters (both configured with a maximum capacity of 1,000,000) is 80 bytes, which is highly suitable for the limited in-vehicle environment.

The can-train-and-test dataset was only made publicly available in 2023, and as of now, literature utilizing this dataset is limited to its own reference, denoted as [31]. Consequently, no other publicly available literature or works have employed this dataset. In light of this, this paper leverages the can-train-and-test dataset as a substitute for real-world scenarios to enhance the practical applicability of ECF-IDS. To facilitate meaningful comparisons with models that also utilize the can-train-and-test dataset, we have selected the Gaussian Naive Bayes [31], K-Nearest Neighbor (KNN) [31], LR [31], SVM [31], DT [31], Extra Tree [31], Random Forest [31], and Multi-Layer Perceptron (MLP) [31] models cited in the dataset literature [31] for benchmarking against the ECF-IDS. The specific sub-dataset chosen for comparison in this paper is sub-dataset 3, and the comparative results are presented in Table VIII.

From the results, it is evident that all models achieve reasonably high accuracy and precision levels. However, there is significant disparity in recall and F1-Score outcomes. On the one hand, SVM and KNN exhibit the worst performance, primarily due to their limited capability in handling imbalanced and non-linear data, resulting in substantial classification bias. On the other hand, LR and MLP demonstrate superior performance, achieving F1-Scores of 98.23% and 99.22%, respectively. Nevertheless, they still fall short in comparison to the ECF-IDS, which outperforms other models across all four metrics. This indicates that ECF-IDS not only exhibits strong capabilities in handling imbalanced data but also displays remarkable stability in both multi-class and binary classification tasks. This assertion is further substantiated by Fig. 14 and Fig. 15, which depict the gradual and stable reduction of training loss with iterations.

D. ECF-IDS Real Vehicle Test

Fig. 16. Real Vehicle Test of ECF-IDS Model

To validate the real in-vehicle environment detection capabilities of the ECF-IDS model, we conducted practical testing by connecting it to a test vehicle. The actual testing environment, as depicted in Fig. 16, involved using a laptop equipped with the ECF-IDS model connected to the OBD-II port inside the vehicle via the CANalyst-II interface. Additionally, we injected DoS, fuzzy, and replay attack data into the CAN bus through the CANalyst-II interface. The specific descriptions of the attacks are as follows:

- **DoS attack:** 100,000 blank messages with both CAN ID and DATA set to 0 were injected every 0.1 milliseconds.
- **Fuzzy attack:** Messages of totally random CAN ID and DATA values were injected every 0.1 milliseconds.
- **Replay attack:** Replay the data packets captured in the previous 10 seconds were injected every 20 seconds.

Subsequently, we employed the ECF-IDS model to perform real-time detection of the CAN message data from the test vehicle (as detailed in Table IX, representing the composition of the real test dataset). As shown in Table X, regarding DoS

TABLE IX
THE COMPOSITION OF THE REAL TEST DATASET

Dataset	Amount
Normal	431495
DoS Attack	99952
Fuzzy Attack	30000

TABLE X
THE PERFORMANCE OF REAL VEHICLE TEST

Attack Type	Accuracy	Precision	Recall	F1-Score
DoS attack	100%	100%	100%	100%
Fuzzy attack	99.99%	100%	99.99%	99.99%
Replay attack	100%	100%	99.99%	99.99%
Normal	100%	100%	100%	100%

attacks and normal traffic detection in real vehicle testing, the ECF-IDS achieved a detection accuracy of 100%. It also demonstrated a performance of 99.99% against fuzzy and replay attacks. On average, throughout the entire process of real vehicle testing, the accuracy of ECF-IDS reached 99.999%, with an F1-Score of 99.997%.

VI. CONCLUSION

Improvements and adaptations were made to the existing cuckoo filter, resulting in an enhanced version called ECF. The ECF offers three main advantages over the original cuckoo filter: 1) The ECF avoids inserting duplicate elements, so as to save storage space and eliminate the limitation on the maximum number of times a single element can be inserted. 2) The ECF combines two filters to address the disadvantage of low detection accuracy in a single filter. 3) The ECF fine-tunes the dual filters based on the subsequent judgments made by the BERT model.

Upon the integration of the BERT model, we proposed the ECF-IDS for the in-vehicle network. The ECF-IDS was trained and tested on the CHD and can-train-and-test datasets, with strong performance demonstrated in both binary and multi-classification tasks, over four traditional algorithms (k-means, LR, SVM and DT). Furthermore, when compared with other in-vehicle IDSs in the literature, the ECF-IDS not only exhibited superior effectiveness but also showcased its advantages in terms of efficiency and lightweight design. During the real in-vehicle testing process, the ECF-IDS model also demonstrated excellent detection performance. Given the limited computational resources in vehicles, a lightweight IDS is essential, and the ECF-IDS is precisely tailored to meet this demand. However, due to the limited resources of GPUs in a vehicle's environment, the training process of the ECF-IDS model can only be completed offline, necessitating regular updates of the detection models to address emerging attacks. In future work, we aim to create a distributed in-vehicle IDS that can perform model training in the cloud and deliver model updates in real-time.

ACKNOWLEDGEMENT

This work is partially supported by ECARX (Hubei) Technology Co., Ltd. for providing the real vehicle testing environment and assistance.

The research is supported in part by EU HORIZON-TMA-MSCA-SE project TRACE-V2X under grant (101131204) and Wuhan Industrial base Innovation Program (2023010402010783).

REFERENCES

[1] M. Elhattab, M. Khabbaz, N. Al-Dahabreh, R. Atallah, and C. Assi, "Leveraging real-world data sets for qoe enhancement in public electric vehicles charging networks," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2023.

[2] S. Longari, D. H. N. Valcarcel, M. Zago, M. Carminati, and S. Zanero, "Cannolo: An anomaly detection system based on lstm autoencoders for controller area network," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1913–1924, 2020.

[3] H. Zhou, B. Liu, T. H. Luan, F. Hou, L. Gui, Y. Li, Q. Yu, and X. Shen, "Chaincluster: Engineering a cooperative content distribution framework for highway vehicular communications," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 6, pp. 2644–2657, 2014.

[4] K. Sharma, B. Butler, and B. Jennings, "Graph-based heuristic solution for placing distributed video processing applications on moving vehicle clusters," *IEEE Transactions on Network and Service Management*, vol. 19, no. 3, pp. 3076–3089, 2022.

[5] Y. Cao, S. Li, C. Lv, D. Wang, H. Sun, J. Jiang, F. Meng, L. Xu, and X. Cheng, "Towards cyber security for low-carbon transportation: Overview, challenges and future directions," *Renewable and Sustainable Energy Reviews*, vol. 183, p. 113401, 2023.

[6] S. Li, Y. Cao, S. Liu, Y. Lai, Y. Zhu, and N. Ahmad, "Hda-ids: A hybrid dos attacks intrusion detection system for iot by using semi-supervised cl-gan," *Expert Systems with Applications*, p. 122198, 2023.

[7] M. A. El-Zawawy, A. Brighente, and M. Conti, "Authenticating drone-assisted internet of vehicles using elliptic curve cryptography and blockchain," *IEEE Transactions on Network and Service Management*, vol. 20, no. 2, pp. 1775–1789, 2023.

[8] L. Zhao, Z. Yin, K. Yu, X. Tang, L. Xu, Z. Guo, and P. Nehra, "A fuzzy logic-based intelligent multiattribute routing scheme for two-layered sd-vns," *IEEE Transactions on Network and Service Management*, vol. 19, no. 4, pp. 4189–4200, 2022.

[9] A. Greenberg, "Hackers cut a corvette's brakes via a common car gadget," https://www.wired.com/2015/08/hackers-cut-corvettes-brakes-via-common-car-gadget/, 2015.

[10] K. S. Lab, "Experimental security assessment of bmw cars: A summary report."

[11] A. Palanca, E. Evenchick, F. Maggi, and S. Zanero, "A stealth, selective, link-layer denial-of-service attack against automotive networks," in *Detection of Intrusions and Malware, and Vulnerability Assessment: 14th International Conference, DIMVA 2017, Bonn, Germany, July 6-7, 2017, Proceedings 14*. Springer, 2017, pp. 185–206.

[12] S. Sinha and R. West, "Towards an integrated vehicle management system in driveos," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 5s, pp. 1–24, 2021.

[13] M. Gupta, F. M. Awaysheh, J. Benson, M. Alazab, F. Patwa, and R. Sandhu, "An attribute-based access control for cloud enabled industrial smart vehicles," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 6, pp. 4288–4297, 2020.

[14] M. S. Rathore, M. Poongodi, P. Saurabh, U. K. Lilhore, S. Bourouis, W. Alhakami, J. Osamor, and M. Hamdi, "A novel trust-based security and privacy model for internet of vehicles using encryption and steganography," *Computers and Electrical Engineering*, vol. 102, p. 108205, 2022.

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

[16] G. Qin, Y. Zhou, J. Yan, J. Chen, B. Rao, and P. Li, "Intrusion detection framework for in-vehicle network combining time features and data features," in *2022 IEEE/CIC International Conference on Communications in China (ICCC)*. IEEE, 2022, pp. 985–990.

[17] A. Miglani and N. Kumar, "Deep learning models for traffic flow prediction in autonomous vehicles: A review, solutions, and challenges," *Vehicular Communications*, vol. 20, p. 100184, 2019.

[18] B. Groza and P.-S. Murvay, "Efficient intrusion detection with bloom filtering in controller area networks," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 4, pp. 1037–1051, 2018.

[19] H. M. Song, J. Woo, and H. K. Kim, "In-vehicle network intrusion detection using deep convolutional neural network," *Vehicular Communications*, vol. 21, p. 100198, 2020.

[20] S. Tariq, S. Lee, H. K. Kim, and S. S. Woo, "Can-adf: The controller area network attack detection framework," *Computers & Security*, vol. 94, p. 101857, 2020.

[21] W. Lo, H. Alqahtani, K. Thakur, A. Almadhor, S. Chander, and G. Kumar, "A hybrid deep learning based intrusion detection system using spatial-temporal representation of in-vehicle network traffic," *Vehicular Communications*, vol. 35, p. 100471, 2022.

[22] H. Qin, M. Yan, and H. Ji, "Application of controller area network (can) bus anomaly detection based on time series prediction," *Vehicular Communications*, vol. 27, p. 100291, 2021.

[23] N. Alkhatib, M. Mushtaq, H. Ghauch, and J.-L. Danger, "Can-bert do it? controller area network intrusion detection system based on bert language model," in *2022 IEEE/ACS 19th International Conference on Computer Systems and Applications (AICCSA)*. IEEE, 2022, pp. 1–8.

[24] E. Nwafor and H. Olufowobi, "Canbert: A language-based intrusion detection model for in-vehicle networks," in *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2022, pp. 294–299.

[25] A. R. Javed, S. Ur Rehman, M. U. Khan, M. Alazab, and T. Reddy, "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.

[26] A. K. Desta, S. Ohira, I. Arai, and K. Fujikawa, "Rec-cnn: In-vehicle networks intrusion detection using convolutional neural networks trained on recurrence plots," *Vehicular Communications*, vol. 35, p. 100470, 2022.

[27] J. Liu, S. Zhang, W. Sun, and Y. Shi, "In-vehicle network attacks and countermeasures: Challenges and future directions," *IEEE Network*, vol. 31, no. 5, pp. 50–58, 2017.

[28] D. Zhang, Z. Chen, J. Ren, N. Zhang, M. K. Awad, H. Zhou, and X. S. Shen, "Energy-harvesting-aided spectrum sensing and data transmission in heterogeneous cognitive radio sensor network," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 1, pp. 831–843, 2017.

[29] S. Boumiza and R. Braham, "An anomaly detector for can bus networks in autonomous cars based on neural networks," in *2019 international conference on wireless and mobile computing, networking and communications (WiMob)*. IEEE, 2019, pp. 1–6.

[30] E. Seo, H. M. Song, and H. K. Kim, "Gids: Gan based intrusion detection system for in-vehicle network," in *2018 16th Annual Conference on Privacy, Security and Trust (PST)*. IEEE, 2018, pp. 1–6.

[31] B. Lampe and W. Meng, "can-train-and-test: A curated can dataset for automotive intrusion detection," *arXiv preprint arXiv:2308.04972*, 2023.

[32] B. Fan, D. G. Andersen, and M. Kaminsky, "Cuckoo filter: Better than bloom," *USENIX Programming*, vol. 38, no. 4, pp. 36–40, 2013.

[33] J. D. M.-W. C. Kenton and L. K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of naacL-HLT*, vol. 1, 2019, p. 2.

[34] Z. Zhang, Y. Wu, H. Zhao, Z. Li, S. Zhang, X. Zhou, and X. Zhou, "Semantics-aware bert for language understanding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 05, 2020, pp. 9628–9635.

[35] J. Li, X. Wang, Z. Tu, and M. R. Lyu, "On the diversity of multi-head attention," *Neurocomputing*, vol. 454, pp. 14–24, 2021.

[36] G. Bebis and M. Georgiopoulos, "Feed-forward neural networks," *Ieee Potentials*, vol. 13, no. 4, pp. 27–31, 1994.

[37] K. Wang, A. Zhang, H. Sun, and B. Wang, "Analysis of recent deep-learning-based intrusion detection methods for in-vehicle network," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 2, pp. 1843–1854, 2022.