A Cloud Collaborative-based Intrusion Detection and Prevention System for IVN

Sifan Li, Yue Cao, *Senior Member, IEEE,* Yu'ang Zhang, Tongxin Liao, Fei Yan, Hai Lin, *Member, IEEE*

***Abstract*—With the increasing intelligence and convenience of modern vehicles, Internet-of-Vehicle (IoV) technology plays a pivotal role in driving these advancements. While IoV enhances user services, it also introduces security threats, particularly intrusions into the In-Vehicle Network (IVN). Attackers can exploit onboard external network interfaces, posing risks to vehicle safety and operation. Besides, deploying an Intrusion Detection System (IDS) on vehicles with limited computational resources exacerbates the computational burden. To address these challenges, this paper proposes a Cloud Collaborative-based Intrusion Detection and Prevention System (CC-IDPS). Firstly, this study analyzes existing in-vehicle intrusion detection datasets and, combined with real in-vehicle/simulation data, constructs the In-vehicle Attack Dataset (IVAD). Secondly, the CC-IDPS employs a sliding window approach for feature extraction from multiple Controller Area Network (CAN) IDs, and leverages Bidirectional Encoder Representation from Transformers (BERT) models for in-vehicle traffic classification. Based on classification results, the CC-IDPS applies treatments to in-vehicle traffic. Subsequently, the proposed $Encode2ID$ algorithm encodes and stores malicious traffic in the database of cloud, facilitating subsequent model training. Experimental results demonstrate that, when applied to the IVAD dataset, the model exhibits lower detection accuracy and F1-score compared to other datasets. Notably, the CC-IDPS outperforms other machine learning and deep learning models in detecting performance on the IVAD dataset.**

***Index Terms*—IVN, CAN, IDPS, BERT, Cloud**

I. Introduction

With the development of intelligent networking and autonomous driving technologies, vehicles have evolved from simple transportation tools into mobile intelligent systems [1]. Modern vehicles not only offer a more intelligent driving experience, but also feature highly interconnected In-Vehicle Networks (IVNs) [2]. These networks provide numerous connectivity interfaces to meet diverse application and service requirements [3]. Coupled with inherent vulnerabilities in IVNs, intelligent vehicles are susceptible to various potential network security loopholes [4]. These security vulnerabilities provide opportunities for cyber attackers. As shown in Fig.1, external attackers exploit remote service vulnerabilities provided by Original Equipment Manufacturers (OEMs) to launch remote attacks on vehicles. Internal attackers can directly target the IVN by physically connecting through the On-Board Diagnostics-II (OBD-II) port. Their goal is to breach IVNs by exploiting potential entry points, aiming to gain control of the vehicle or steal personal information from the owner, resulting in diverse information security concerns. These security threats not only jeopardize passenger safety but also encompass risks of privacy breaches [5].

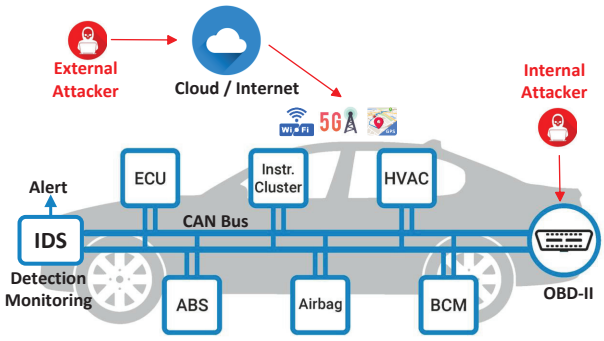


Fig. 1. Illustration of Typical Vehicle Attacks

In recent years, security incidents within the realm of IVNs have garnered significant attention. In 2022, German technology security expert David Colombo highlighted on Twitter a software vulnerability he discovered within Tesla's systems. This vulnerability enabled remote access to 25 Tesla vehicles across 13 countries, allowing for a series of actions including remotely opening doors and windows, starting the vehicle, and disabling the security systems [6]. In 2017, researchers Daan Keuper and Thijs Alkemade from the Dutch cybersecurity firm Computest conducted an investigation on various models of Volkswagen and Audi vehicles, uncovering remote exploitation vulnerabilities within their in-vehicle infotainment systems [7]. Attackers gained access to the infotainment system via the in-vehicle Wi-Fi device, and exploited vulnerabilities within the infotainment system to send Controller Area Network (CAN) messages to the CAN bus, ultimately gaining control over devices such as the central screen, speakers, and microphone.

Security issues concerning IVNs require solutions to safeguard vehicle systems from potential network threats [8]. As one of the most effective solutions at present, although traditional machine learning-based Intrusion Detection Systems

This work is supported in part by National Key Research and Development Program of China (2024YFB3108400), Hubei Province Major Science and Technology Innovation Program (2024BAA011), and Wuhan AI Innovation Program (2023010402040020).

S. Li, Y. Cao (corresponding author), Y. Zhang, F. Yan and H. Lin 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, yuang.zhang@whu.edu.cn, yanfei@whu.edu.cn, lin.hai@whu.edu.cn).

T. Liao is with Hubei Key Laboratory of Smart Internet Technology, School of Electronic Information and Communication, Huazhong University of Science and Technology, Wuhan, China, 430074. (e-mail: liaotongxin@hust.edu.cn).

(IDSs) (like [9], [10], [11], [12], [13]) perform well in detecting known Denial-of-Service (DoS) attacks, their performance is unsatisfactory when faced with complex attacks such as fuzzy and spoof attacks [14]. These IDSs are constrained by the limited computational resources within vehicles, thus exhibiting certain limitations. For example, these IDSs are difficult to adapt to dynamically changing network threats and may fail to detect sophisticated attack behaviors. As a result, there is an urgent call for a more comprehensive and innovative security framework to enhance IVN security.

Simultaneously, training models for in-vehicle IDSs rely on datasets. However, due to privacy and confidentiality concerns in vehicle data, the majority of vehicle manufacturers do not publicly disclose in-vehicle intrusion datasets [14]. This limitation results in a scarcity of publicly available datasets, with Hacking and Countermeasure Research Lab (HCRL) datasets being the primary source. Although the HCRL has amassed a considerable volume of data from real in-vehicle environments, each individual dataset is limited in its coverage of attack types and quantity. This limitation makes current datasets insufficient to comprehensively cover the diverse security threats and attack scenarios of IVNs.

To address these issues, this paper aims to construct a new dataset by leveraging both public datasets and data collected from real/simulated environments. Additionally, this paper seeks to explore and design a Cloud Collaborative-based Intrusion Detection and Prevention System (CC-IDPS). By fully harnessing the resources of cloud computing platforms and employing collaborative defense mechanisms, this study aims to develop a novel IDPS to counter the increasingly complex and diverse network threats faced by IVNs. The primary innovations of this paper are as follows:

- To address the limited variety and quantity of attacks in existing datasets, this paper consolidates HCRL's Car Hacking Dataset (CHD) [15], the Survival Analysis Dataset (SAD) [16], and data collected from real in-vehicle/simulation environments. This integration results in the development of an In-Vehicle Attack Dataset (IVAD)[1]. Due to the broad sources of IVAD and its relatively unfruitful features, it presents considerable challenges for subsequent deep learning model detection. To tackle this issue, the paper employs a sliding window mechanism to extract the CAN ID field of each traffic within the corresponding window size, subsequently incorporating these CAN IDs as features of adjacent traffic into the current flow.
- To address the limitations of in-vehicle IDSs due to the restricted computational resources within vehicles, this paper proposes the CC-IDPS. It conducts model training tasks on cloud servers, and subsequently distributes the model to vehicles for detection. The detection model of this system primarily employs Bidirectional Encoder Representation from Transformers (BERT) for classifying the traffic under inspection. Subsequently, based on the classification results, vehicles undertake appropriate actions to defend against the malicious traffic.
- This study enhances the BERT model for IVN intrusion detection by adapting it to process serialized IVN data, effectively transforming features like message IDs and payload contents into a suitable input format. By leveraging the Masked Language Model (MLM), the model captures contextual relationships between adjacent CAN traffic, aiding in the classification of traffic as malicious or benign. Additionally, the activation function of the output layer is adjusted to a sigmoid function, ensuring compatibility with the binary classification objective aimed at detecting malicious traffic.
- Finally, due to the evolving nature of network attacks, the CC-IDPS requires periodic database updates to train new detection models. However, during the process of feeding malicious traffic back to the cloud server, there might be instances of redundant storage. To address this issue, the CC-IDPS involves an $Encode2ID$ algorithm to encode malicious traffic, generating a unique ID for each. As a result of the $Encode2ID$ algorithm, the generated strings occupy less space compared to before, thereby saving storage space and reducing non-repetitive comparison time. Furthermore, the proposed $StoreEventID$ and $VehicleDetection$ algorithms not only update the database to counter novel attacks, but also drop malicious traffic for defensive measures.

The remaining structure of this paper is as follows: Section VII enumerates related works on in-vehicle IDSs. Section II introduces the in-vehicle CAN bus and its associated attacks. The specific construction process of the IVAD dataset is outlined in Section III. The proposed CC-IDPS framework and experimental evaluations are presented in Section IV and V, respectively. Section VIII provides a conclusion for this paper.

II. Background

A. In-Vehicle CAN Bus

The CAN is a crucial network architecture used for communication and data transmission among various Electronic Control Units (ECUs) within the internal control area of intelligent vehicle [17]. The CAN bus plays a pivotal role in modern vehicles, facilitating real-time data exchange and collaboration among different internal vehicle systems. Furthermore, the layout design of the CAN bus holds critical significance in modern vehicles, determining the communication method and data exchange pathways among the vehicle's internal systems.

1) The network topology of CAN bus: Common layouts for the CAN bus currently include star, bus, and hybrid topologies. In a star topology, individual control units are directly connected to the main control unit via a hub or switch. The bus topology involves connecting various control units through a single bus, facilitating data transmission on the bus. The hybrid topology combines aspects of both star and bus topologies, employing a specific structure to balance system stability and data transmission efficiency.

2) Main Control Unit (MCU): In the CAN bus, there are typically one or multiple MCUs responsible for coordinating and managing the entire CAN bus system. The MCUs usually

[1]https://github.com/yuecao871441562/IVAD

serve as the central processor of vehicle, tasked with controlling data flow, allocating network resources, and managing the priority of data transmission [18].

3) Data Transmission Protocol: The CAN bus employs a specific communication protocol, known as the CAN protocol, for data transmission and communication. The CAN protocol operates at two different speeds: standard CAN (typically at 250 kbit/s) and high-speed CAN (typically at 500 kbit/s or higher rates). These protocols define the format of message transmission, the structure of data packets, and transmission rates, ensuring reliable data transmission on the bus [19].

4) ECU: In the CAN bus, each control unit is referred as a CAN node or ECU. These nodes can encompass various control units for different vehicle systems, such as the engine control unit, brake control unit, air condition control unit, among others. Each ECU is capable of sending and receiving data via the CAN bus to facilitate communication with other systems.

5) Message ID (CAN ID): Data transmission on the CAN bus is identified and differentiated through message IDs. Each data packet sent to the CAN bus contains a unique ID that signifies the type of data packet and its sender. Such a design ensures that data transmitted between different systems on the bus can be accurately identified and processed.

6) Data Frame Structure: Data transmission on the CAN bus relies on data frame structures, categorized into standard frames and extended frames. Standard frames comprise an 11-bit message ID, while extended frames utilize a 29-bit message ID, allowing for a larger address space for message differentiation. The data frame structure also consists of data fields, control fields, and Cyclic Redundancy Check (CRC) verification fields, ensuring data integrity and accuracy.

B. In-Vehicle Attack

As depicted in Fig. 2, there are nine primary attacks posing threats to the IVN. DoS and randomized DoS attacks primarily target the Heating Ventilation and Air Conditioning (HVAC) and ECU systems, while fuzzy and intermittent masquerade 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. Malfunction attacks primarily target ABS and BCM systems, which is the same as gear spoof attacks. Replay attacks have a broad attack surface, affecting messages flowing through the CAN bus.

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 IVN. 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

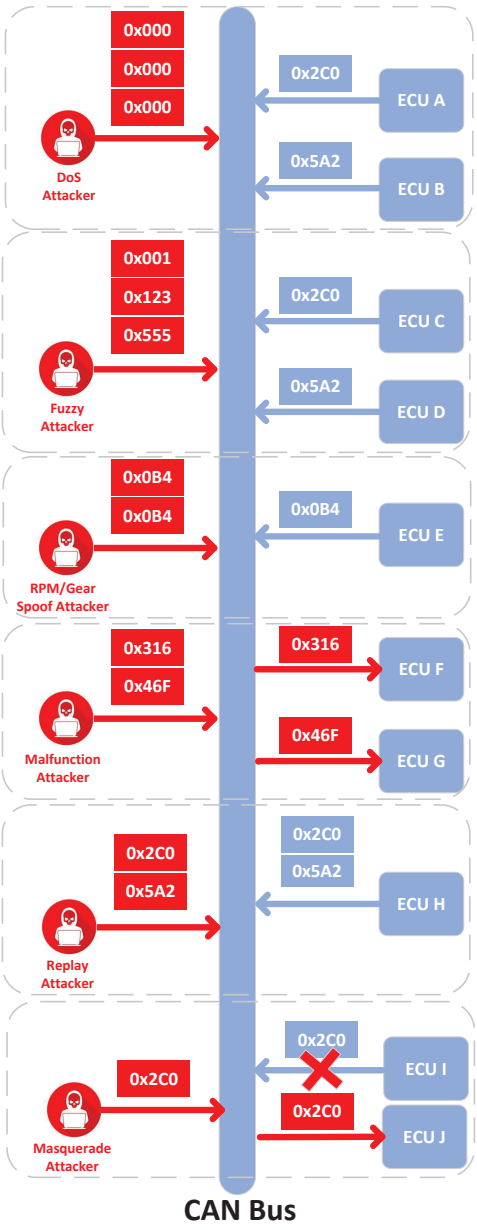


Fig. 2. In-vehicle Attacks

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 exploit 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 security of the vehicle by creating confusion or misinterpretation of sensor data.

3) Gear Spoof Attack: Attackers can manipulate data from vehicle sensors or ECUs, deceiving the control systems and displaying incorrect gear lever positions, leading to inaccurate shifting or speed indications. Such attacks may result in the

display of false gear lever positions, misleading drivers, and potentially causing accidents or discomfort during driving.

4) RPM Spoof Attack: Attackers manipulate data from vehicle sensors or ECUs to transmit false tachometer information to the control system, misleading the actual engine speed. This action might result in the vehicle displaying incorrect engine speed information, disrupting the ability of driver to make accurate driving judgments and potentially creating safety hazards.

5) Malfunction Attack: Attackers maliciously manipulate commands or data within control units, leading to malevolent manipulation of the control system, which can result in system malfunctions or incorrect operations. Malfunction attacks can potentially disable critical vehicle systems such as engine start/stop functions, brake system malfunctions, posing a threat to both vehicle and passenger safety.

6) Replay Attack: Attackers intercept previously valid communication data packets and resend them to the system to deceive it into performing the same operations. This action may result in the system executing repeated instructions or data packets multiple times, potentially causing system state confusion or unpredictable behavior.

7) Intermittent Masquerade Attack: An Intermittent Masquerade Attack in IVNs involves an attacker occasionally impersonating a legitimate node at irregular intervals to avoid detection. The attacker varies the message IDs and timing to mimic normal traffic patterns, making the attack sporadic and challenging to identify.

8) Combination Attack: A combination attack in IVNs involves blending two or more attack types, such as DoS, replay, fuzzy, and spoof, to create a more sophisticated threat. By mixing these attacks, the goal is to confuse the IDS and exploit multiple vulnerabilities simultaneously.

9) Randomized DoS Attack: Instead of using a constant flooding technique, the attacker sends bursts of data packets at random intervals with varying payload sizes and message IDs to overwhelm the network or specific ECUs. This unpredictability makes it harder to detect using simple rate-based or pattern-matching algorithms.

C. In-Vehicle Dataset

1) Car Hacking Dataset (CHD): The CHD [15] from the HCRL provides valuable data for automotive cybersecurity research, specifically for detecting and mitigating attacks on IVNs. The CHD [15] includes various types of message injection attacks, such as DoS, fuzzy, and spoofing attacks targeting the RPM and gear system. These attacks were conducted on a real vehicle, with CAN traffic being logged through the OBD-II port during the attacks. Each attack scenario contains 300 instances of message injection, with each attack lasting between 3 to 5 seconds, and the total recorded data spans 30 to 40 minutes of CAN traffic. The CHD [15] includes attributes like Timestamp, CAN ID, DLC (Data Length Code), DATA (byte values), and Flag to distinguish between normal and injected messages. This dataset is designed to help researchers develop and evaluate IDS that can identify malicious activities in the CAN.

2) Survival Analysis Dataset (SAD): The SAD [16] from the HCRL is designed to support the development and evaluation of intrusion detection methods for IVN. It includes data from three different attack scenarios: flooding, fuzzy, and malfunction, each of which poses a significant threat to vehicle functionality. The data was collected from multiple vehicles, with normal driving data captured alongside attack. For the attack scenarios, messages were injected at specific intervals (e.g., every 20 seconds for 5 seconds) to simulate real-world intrusion attempts. The SAD [16] includes attributes such as Timestamp, CAN ID, DLC, and Flag (indicating normal or injected messages), which are used to analyze the impact of these attacks on the IVN. This dataset is essential for developing models that can detect and mitigate these attacks, offering valuable insights into enhancing vehicle security and driver safety.

III. IVAD Construction

As existing in-vehicle intrusion detection datasets have limited diversity and a small scale of attack types, coupled with the lack of standardized data formats and labeling methods among different datasets, there is an urgent need for a larger-scale dataset encompassing a wider array of attack types, unified labeling, and standardized data formats. The IVAD dataset presented in this paper precisely meets these requirements. It is primarily composed of data collected from real and simulated environments, supplemented by datasets from the HCRL, including the CHD [15] and SAD [16].

A. Data Collection and Processing

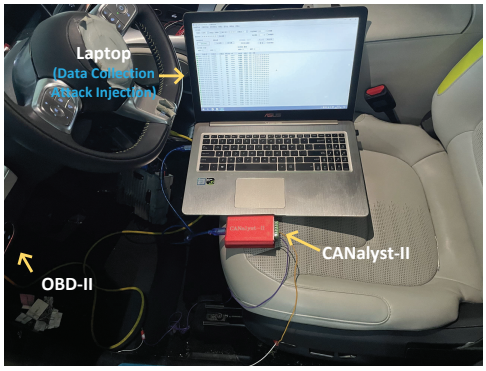


Fig. 3. Real Data Collection

The core of IVAD was collected from the real Smart #1 vehicle. As depicted in Fig. 3, a laptop was connected to the OBD-II port via CANalyst-II. The CANalyst-II device is a professional CAN bus analyzer that enables real-time monitoring and analysis of CAN bus traffic. In our experimental setup, we utilized the CANalyst-II to capture CAN bus messages during normal vehicle operation and during simulated attack scenarios. The device allows us to analyze the captured traffic to identify patterns indicative of various attacks, such as replay, fuzzy, and DoS attacks. Regarding the implementation of different attacks, we simulated various attack scenarios by injecting malicious CAN messages into

the CAN bus using custom-built scripts and tools. The specific descriptions of the attacks are as follows:

- **DoS attack:** Inject 587,521 blank messages with both CAN ID and DATA set to 0 every 0.1 milliseconds.
- **Randomized DoS attack:** Inject 100,000 blank messages with both CAN ID and DATA set to 0 randomly.
- **Combination attack:** Randomly mix 50,000 replay messages (with slightly modified CAN IDs to simulate legitimate traffic) and 30,000 fuzzy messages (malformed CAN messages with random data fields) before injecting them into the CAN bus.
- **Intermittent Masquerade attack:** Begin by sending 20,000 crafted messages with valid CAN IDs and slightly altered data fields to mimic typical network traffic. Wait for a random interval between 2 to 5 minutes, then inject another set of 10,000 messages, varying both the message frequency and content to avoid detection.
- **Fuzzy attack:** Inject messages of totally random CAN ID and DATA values every 0.1 milliseconds.
- **Replay attack:** Replay the data packets captured in the previous 10 seconds every 20 seconds.

To enhance the variety of attacks in the IVAD, spoof attacks such as gear/RPM from the CHD [15] and malfunction attacks from the SAD [16] were extracted and incorporated into IVAD. The malfunction attack data extracted from the SAD [16] mainly comes from HYUNDAI YF Sonata, KIA Soul, and CHEVROLET Spark vehicles, while the attack data extracted from the CHD [15] only comes from Hyundai's YF Sonata. Furthermore, to align with the cloud-vehicle collaborative detection environment, we collected data (include normal, DoS attack, fuzzy attack, masquerade attack, combination attack and replay attack) from a cloud-vehicle collaborative simulation environment. This simulation environment is composed of a cloud server and multiple virtual machines (details in subsequent Section V).

Since IVAD requires a uniform labeling and data format, the compiled IVAD undergoes standardized preprocessing. Initially, the annotation format of the 'label' column is standardized (1 representing malicious traffic and 0 representing normal traffic). Also, missing values are removed. Subsequently, hexadecimal data is converted to decimal form for subsequent computations using the following formula:

$$y = hex\_to\_dec\,(f) \tag{1}$$

f represents the hexadecimal data to be converted, $hex\_to\_dec()$ is the function to convert hexadecimal to decimal, and y denotes the corresponding decimal data after conversion. Finally, this study applies Z-score normalization techniques to standardize the CAN messages across different vehicle types. This regularization helps in minimizing the impact of protocol differences on the performance of models.

B. Traffic Classification

As shown in Table I, the total number of flows in IVAD is 5295067, mainly categorized into seven types. Among these, there are 2542535 instances of normal traffic, 587521 instances of DoS attack traffic, 100000 instances of randomized DoS attack traffic, 80000 instances of combination attack traffic, 30000 instances of intermittent masquerade attack traffic, 590531 instances of fuzzy attack traffic, 96357 instances of replay attack traffic, 597252 instances of gear spoof attack traffic, 654897 instances of RPM spoof attack traffic, and 15974 instances of malfunction attack traffic.

TABLE I
CLASSIFICATION OF IVAD

Classification	Amount
Normal	2542535
DoS Attack	587521
Randomized DoS Attack	100000
Combination Attack	80000
Intermittent Masquerade Attack	30000
Fuzzy Attack	590531
Replay Attack	96357
Gear Spoof Attack	597252
RPM Spoof Attack	654897
Malfunction Attack	15974
Total	**5295067**

C. Features

After data preprocessing, the retained features in IVAD are CANID, Data [0-7], and Label, which hold the following specific meanings:

1) CAN ID: The ID segment carries information about the source and priority of a message. It distinguishes different types of messages and facilitates message filtering.

2) Data [0-7]: 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.

3) Label: The label of the traffic is used to differentiate whether the flow is malicious or not. Here, '1' represents a malicious message while '0' represents a normal message.

IV. PROPOSED CC-IDPS FRAMEWORK

The CC-IDPS is an efficient protective mechanism designed to address the security challenges faced by smart vehicles. The design of this system framework aims to leverage the advantages of cloud computing by centrally processing data from multiple IVNs, thereby enhancing the efficiency, accuracy, and real-time capability of intrusion detection and prevention [20]. The overall framework and general steps of CC-IDPS are illustrated in Fig. 4, which mainly comprises the following two entities:

- **Cloud Server:** The cloud server is one of the core components of this framework, primarily responsible for tasks such as data collection and preprocessing, model training, model distribution, and storing malicious data.
- **Vehicles:** At the vehicle end, the primary responsibilities include data uploading, model loading and detecting, and corresponding defensive operations.

The detection process of CC-IDPS can be divided into four major blocks: cloud-based data collection and processing, feature engineering, BERT model detection, and feedback with defense.

This article has been accepted for publication in IEEE Transactions on Cognitive Communications and Networking. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCCN.2024.3516052

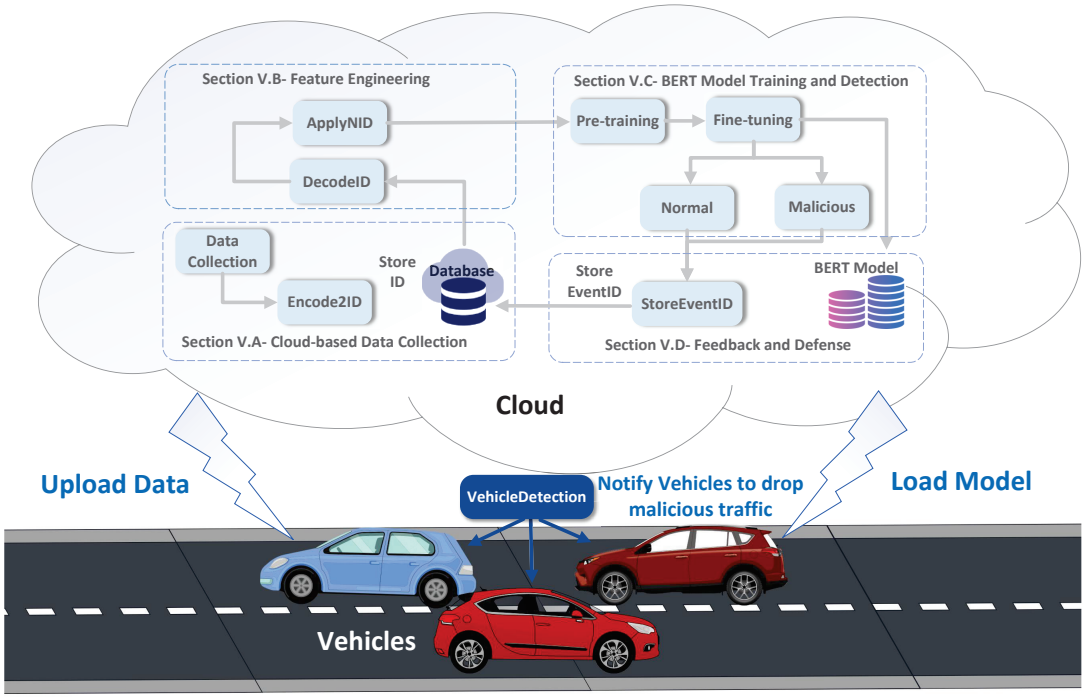


Fig. 4. The Framework of CC-IDPS

A. Cloud-based Data Collection

This system initially achieves centralized collection and processing of IVN data through cloud infrastructure. Each vehicle uploads IVN data to the cloud server via an internet connection. These data may include information from various vehicle communication buses such as the Body-CAN (B-CAN), Multimedia-CAN (M-CAN), and Cabin Communication bus [21]. As the initial cloud server does not contain data, the CC-IDPS employs the aforementioned IVAD to train the initial models. Additionally, due to the substantial volume of collected data, the CC-IDPS optimizes storage space by storing only malicious traffic. Moreover, the system utilizes the proposed $Encode2ID$ algorithm to transform malicious traffic into unique ID fields. This approach not only enables swift identification of duplicate IDs to save storage space, but also prevents malicious data theft and misuse. Furthermore, the $Encode2ID$ algorithm serves as a form of encryption for data processing.

Algorithm 1 Encode2ID

Input: $traffic$, $alphabet$
Output: id

1: $offset \leftarrow 0$
2: **for** i, v in $traffic$ (i is the element of $traffic$, v is index) **do**
3: $offset$ += return unicode($alphabet[v$ % length($alphabet$)]+i)
4: **end for**
5: $offset \leftarrow (offset$ + length($traffic$)) % length($alphabet$)
6: $new\_alphabet \leftarrow alphabet[offset:$] + $alphabet[$: $offset]$
7: $prefix \leftarrow alphabet[0]$
8: $new\_alphabet \leftarrow new\_alphabet[$: :-1]
9: $ret \leftarrow [prefix]$
10: **for** j in $traffic$ **do**
11: ret.append(Convert $new\_alphabet[1:]$ to id)
12: **if** $j \geq$ length($traffic$)-1 **then**
13: Continue
14: **end if**
15: ret.append($new\_alphabet[0]$)
16: $new\_alphabet \leftarrow$ shuffle($new\_alphabet$)
17: **end for**
18: $id \leftarrow$ " ".join(ret)
19: **if** length(id)< min_length **then**
20: id += $new\_alphabet[0]$
21: **while** length(id)< min_length **do**
22: $new\_alphabet \leftarrow$ shuffle($alphabet$)
23: id += $new\_alphabet[$: min(min_length-length(id), length($new\_alphabet$))]
24: **end while**
25: **end if**
26: **return** id

The specific process of the $Encode2ID$ algorithm is illustrated in Algorithm 1. Firstly, this algorithm computes an $offset$ value. It iterates through each number in the numbers list, takes the remainder to retrieve the ASCII value corresponding to the letter in the $alphabet$, and accumulates these values to form the $offset$ (line 1-4). Then, the calculated $offset$ undergoes further processing to ensure it falls within the length range of the $alphabet$ (line 5). Next, based on the computed $offset$ value, the algorithm reorders the $alphabet$ to create a $new\_alphabet$. It extracts the first letter from the

$new\_alphabet$ as a $prefix$ and performs an inverse operation on the entire $alphabet$ (line 6-8). An id is created using characters from the $alphabet$. It appends these characters to the ret list. During character addition, specific letters are added to ret according to certain rules (line 10-15), and a shuffle operation is applied to the $alphabet$ (line 16). Finally, the characters in the ret list are concatenated into a string named id (line 18). If the length of the generated id is less than the defined minimum length (min_length), padding is applied to ensure the id meets the minimum length requirement (line 19-25).

Algorithm 2 DecodeID

Input: id, $alphabet$
Output: $traffic$
1: $traffic \leftarrow$ Empty List
2: **if** id is empty **then**
3: **return** $traffic$
4: **end if**
5: $alphabet\_chars \leftarrow$ convert $alphabet$ to list
6: **if** any $character$ not in $alphabet\_chars$ exist in id **then**
7: **return** $traffic$
8: **end if**
9: $prefix \leftarrow id$[0]
10: $offset \leftarrow$ find index of $prefix$ in $alphabet$
11: $alphabet \leftarrow alphabet[offset:] + alphabet[:offset]$
12: Reverse $alphabet$
13: $id \leftarrow$ substring id from index 1 to end
14: **while** id is not empty **do**
15: $separator \leftarrow$ first character of $alphabet$
16: $chunks \leftarrow$ split id by $separator$
17: **if** $chunks$ is not empty **then**
18: **if** first $chunk$ in $chunks$ is empty **then**
19: **return** $traffic$
20: **end if**
21: $traffic$.append(Convert $chunks[0]$ to number)
22: **if** length($chunks$) >1 **then**
23: $alphabet \leftarrow$ shuffle($alphabet$)
24: **end if**
25: **end if**
26: $id \leftarrow$ join $chunks$ of id from index 1 to end with $separator$
27: **end while**
28: **return** $traffic$

To further analyze the data and train models, it is necessary to extract data from the malicious traffic database for decoding operations. The primary algorithm used for decoding, referred to as $DecodeID$, is illustrated in Algorithm 2. The Algorithm 2 first defines an empty list $traffic$ to store the decoded list of integers. If the input id is an empty string, it directly returns an empty list of integers, $traffic$ (line 2-4). It checks whether the character in the input id are included in the $alphabet$. If any characters in id is not in the $alphabet$, it returns an empty list of integers, $traffic$ (line 5-8). The algorithm retrieves the first character from id to serve as a $prefix$ (line 9). Then, it identifies the position of this $prefix$ within the $alphabet$ and rearranges the $alphabet$ based on this position, reversing its order (line 10-12). Next, the first character of id is removed (line 13). Following this is a loop that decodes id (line 14). Within the loop, it first identifies the current $separator$ used for division. Then, id is segmented into several parts, referred to as $chunks$, based on the $separator$ (line 15-16). It checks if the first part of $chunks$ is empty. If so, it returns an empty list of integers, $traffic$ (line 17-20). Subsequently, it converts the first part of $chunks$ into a number and adds the result to $traffic$ (line 21). If there are more than one part in $chunks$, the $alphabet$ is shuffled (line 22-24). Finally, it updates id with the remaining part and returns $traffic$ (line 26).

B. Feature Engineering

Algorithm 3 ApplyNID

Input: CAN traffic set df, window size w
Output: df_NID
1: $df\_NID \leftarrow df$
2: Add $w-1$ CANID rows in df_NID
3: **for** i in range(1, length(df_NID)) **do**
4: **for** j in range(1, $w+1$) **do**
5: **if** $i \geq j$ **then**
6: df_NID.iloc[i, j] = df_NID.iloc[$i-j$, 0]
7: **end if**
8: **end for**
9: **end for**
10: **return** df_NID

Due to the relatively unfruitful features related to the CAN bus, which adds difficulty to the subsequent model training and classification, the CC-IDPS employs sliding window mechanism to extract the CANID of the previous w data entries for each data entry as a new feature. Here, w represents the size of the sliding window. Algorithm 3 demonstrates the specific process by first assigning df to df_NID (line 1), then adding $w-1$ columns of CANID to df_NID (line 2), and finally writing the CANIDs within the window w to their corresponding positions (line 3-9), forming the new df_NID which is then returned. Assuming w is set to 3, then the $t-th$ data entry possesses three CANID features. CANID1 represents the CANID of the current data, CANID2 represents the CANID of the $(t-1)-th$ data, and CANID3 represents the CANID of the $(t-2)-th$ data.

C. BERT Model Training and Detection

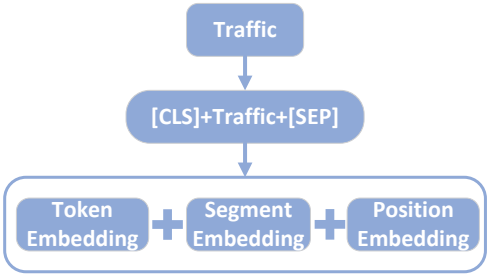


Fig. 5. The Embedding of CAN

BERT, as one of the commonly used models in NLP, has the ability to capture contextual relationships and perform subsequent tasks [22], [23]. 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.

Translating CAN data into the input form of a BERT model is crucial for several reasons. First, BERT is designed to process sequential data and relies on contextual relationships to understand meaning. Therefore, converting CAN messages into a format that resembles natural language allows the model to leverage its inherent strengths. By structuring CAN features, such as message IDs and payload contents, into sequences, we facilitate the ability of CC-IDPS to identify patterns and dependencies between adjacent messages. Additionally, this transformation is essential for capturing the temporal dynamics of in-vehicle communications, which can provide valuable insights into normal versus anomalous behavior. The pre-training of BERT on vast amounts of text data equips it with a rich understanding of language context, enabling it to apply this knowledge to the automotive domain. By adapting the input format, we can maximize the performance of CC-IDPS in detecting intrusions, as it can more effectively analyze the intricate relationships within the data. Moreover, this approach streamlines the feature extraction process, transforming raw CAN data into meaningful representations that enhance classification accuracy.

Therefore, when applying it to IVN data, the data needs to be converted into a format suitable for BERT input. Specifically, this involves serializing each feature of the IVN data (such as message IDs, data frame contents, etc.) into "sentences", and combining multiple feature sequences into a "document". Each "document" represents a segment of IVN traffic over a period of time. Then, the CC-IDPS converts multiple specific features, such as message IDs, payload contents, and timing characteristics from the IVAD, into vector representations, making them suitable as inputs for the BERT model. Finally, since the goal of this manuscript is to detect whether the in-vehicle traffic is malicious, the activation function in the final output layer is adjusted to a sigmoid function.

As illustrated in Fig. 5, for the CAN traffic set t, where $t(i)$ represents the i-th specific CAN traffic within the set, the traffic $t(i)$ is prefixed with [CLS] (Classification), while [SEP] is inserted as a delimiter between each 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 preprocessing for BERT.

1) Pre-training: As shown in Fig. 6, 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]) [24]. These tokens are then used as input into the encoder layer to predict the original tokens. As illustrated in the encoder layer of BERT, it consists of two components: multi-head attention mechanism [25] and feed-forward neural network [26]. 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:

$$\mathbf{H}^{(i,l)} = h(\mathbf{Y}^{(i,l)}) + nor\left(\mathbf{Y}^{(i,l)} + h(\mathbf{Y}^{(i,l)})\right) \tag{2}$$

$$\mathbf{F}^{(i,l)} = f(\mathbf{H}^{(i,l)}) + nor\left(\mathbf{H}^{(i,l)} + f(\mathbf{H}^{(i,l)})\right) \tag{3}$$

$$\mathbf{Y}^{(i,l+1)} = \mathbf{F}^{(i,l)}, \forall l < L \tag{4}$$

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

$$\mathbf{Q}^{(i,l)} = \mathbf{Y}^{(i,l)}\mathbf{W}^{(Q)} \tag{5}$$

$$\mathbf{K}^{(i,l)} = \mathbf{Y}^{(i,l)}\mathbf{W}^{(K)} \tag{6}$$

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

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

$$\text{Attn}\left(\mathbf{Q}^{(i,l)}, \mathbf{K}^{(i,l)}, \mathbf{V}^{(i,l)}\right) = \sigma\left(\frac{\mathbf{Q}^{(i,l)}\mathbf{K}^{(i,l)T}}{\sqrt{d}}\right)\mathbf{V}^{(i,l)} \tag{8}$$

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 $h\left(\mathbf{Y}^{(i,l)}\right)$:

$$head^{(i,l)} = Attn\left(\mathbf{Q}^{(i,l)}, \mathbf{K}^{(i,l)}, \mathbf{V}^{(i,l)}\right) \tag{9}$$

$$h(\mathbf{Y}^{(i,l)}) = linear(head^{(i,l)}) \tag{10}$$

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

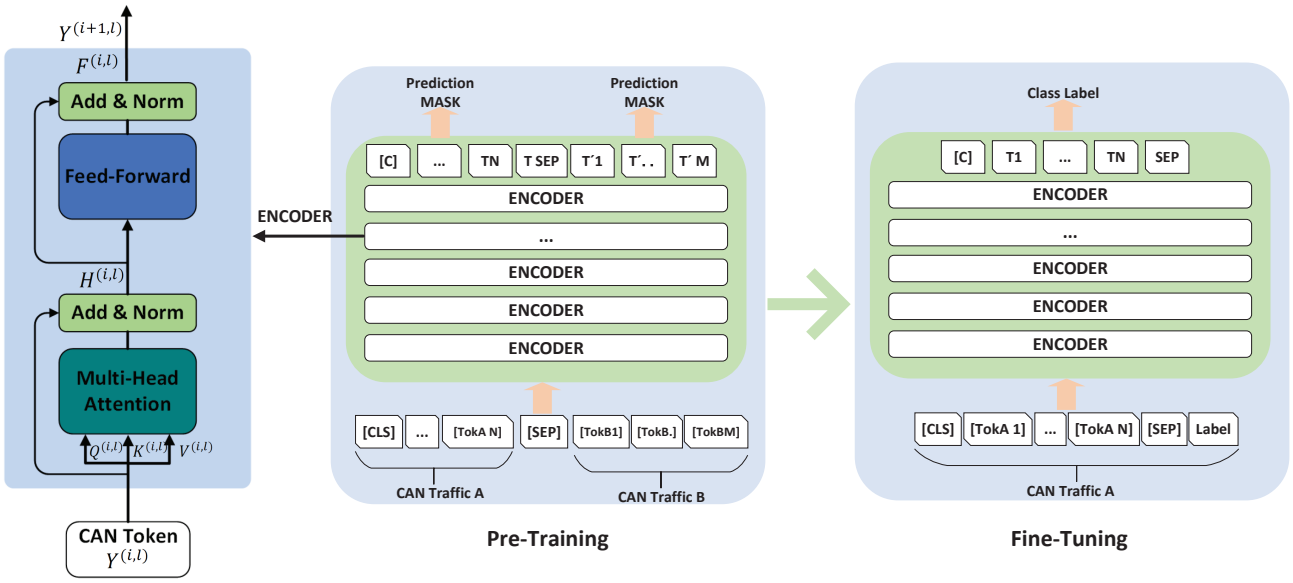


Fig. 6. The Main Process of BERT

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

$$FFN(X) = max(0, \mathbf{h}^{(i,l)} \cdot W_1 + b_1) \cdot W_2 + b_2 \qquad (11)$$

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]:

$$PredictionMask = \sigma[linear(\mathbf{F}^{(i,L)})] \qquad (12)$$

2) Fine-tuning: Fine-tuning involves using the pre-trained BERT model as initial parameters. Then, in order to adapt the CC-IDPS 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. In this paper, the classification task of CC-IDPS is binary (normal, malicious). Therefore, this layer is a fully connected layer used to extract task-related features from the output of BERT. Then, the sigmoid activation function is applied to process the results to output either 0 or 1.
- **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:** 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, 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.

The sigmoid function used in binary classification is shown below:

$$s(x) = \frac{1}{1 + e^{-x}} \qquad (13)$$

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 $x = 0, s(x) = 1/2$.

D. Feedback with Defense

Algorithm 4 StoreEventID

Input: $traffic$
Output: $EventID$

```
EventID ← empty list
for i in traffic do
    if i is malicious then
        i.append(1)
    else if i is normal then
        i.append(0)
    else
        send i to BERT model
    end if
    id ← Encode2ID(i)
```

```
    if id exist in EventID then
      drop i
    else
      EventID.append(id)
    end if
  end for
  return EventID
```

The traffic processed by the BERT model needs to be integrated with the classification results and fed back to the cloud server. The cloud server operates on the traffic according to Algorithm 4. Firstly, the cloud server appends a label character to the traffic based on the output of the BERT model. If the traffic i is malicious, it appends '1' (line 3-4). Conversely, if it is normal, it appends '0' (line 5-6). If there are no results, the traffic i is reclassified through the BERT model (line 7-8). Then, the traffic i is encoded using the aforementioned $Encode2ID$ algorithm (line 10). Next, the encoded id undergoes non-redundant storage (line 11-16), and finally, the $EventID$ is returned.

Algorithm 5 VehicleDetection

```
1: load model from cloud
2: transmit error counter (TEC) = 0
3: if model is the latest version then
4:   for i in traffic do
5:     upload i to cloud
6:     send i to model
7:     if the output of model is malicious (1) then
8:       transmit the error frame
9:       TEC (the node of sending i) = TEC + 8
10:      if TEC <255 then
11:        continue
12:      else
13:        if the node of sending i belongs to important
           ECU set then
14:          the vehicle system enters limp mode
15:          report security incidents to VSOC
16:        else
17:          this node enters bus-off state
18:        end if
19:      end if
20:    else
21:      continue
22:    end if
23:  end for
24: else
25:    update model from cloud
26:    repeat line 3-22
27: end if
```

The detection process of vehicles end is shown in Algorithm 5. First, the classification $model$ is loaded from the cloud and the latest version is ensured (line 1 and line 24). Subsequently, two operations are performed on each traffic to be detected: uploading to the cloud and sending to the classification $model$ (line 5-6). According to the output result of the classification model, if the traffic is malicious, the CC-IDPS will transmit the error frame into the CAN bus, causing the node of sending i to increase its Transmit Error Counter (TEC) by 8 (line 7-9). If the TEC less than 255, this algorithm will continue. Otherwise, if this node belongs to important ECU set (Engine control unit ECU, brake system control unit, etc.), the vehicle enters limp mode, where only the basic safety functions of the vehicle are maintained. Then, the vehicle system reports security incidents to Vehicle Security Operation Center (VSOC). If this node does not belong to important ECU set, it enters bus-off state [27] (line 10-18). If the output result of the classification model is normal, it will continue to perform the next step (line 19-20).

The cloud server utilizes $EventID$ to train new BERT models and periodically distribute the updated BERT models to each vehicle for updates. At the same time, vehicles load the latest model using the $VehicleDetection$ algorithm, and responds with appropriate defenses based on the output of model. This feedback and defense mechanism not only notify the vehicles to defend against detected malicious traffic, but also ensure timely updates of detection models to cope with the evolving new attacks in the network.

V. Experimental Evaluation

A. Environment Setting

Due to limitations in the number of actual vehicles available, we conducted CC-IDPS testing using a server along with one vehicle (Smart #1) and three virtual machines, each hosting a simulated vehicle environment. On the cloud server side, we employed a configuration comprising an AMD Ryzen Threadripper PRO 5995WX processor, Nvidia RTX4090 GPUs (24GB each, four in total), and 64GB of RAM. All three virtual machines are equipped with 2GB of RAM. Each virtual machine was equipped with the Ubuntu operating system and installed with IC simulator [28] + SavvyCAN [29] as illustrated in the Fig. 7.

- **IC Simulator:** IC Simulator comprises a dashboard featuring a speedometer, door lock and turn signal indicators, along with a control panel enabling user interaction with the simulated IVN. This facilitates actions such as acceleration, braking, and controlling door locks and turn signals.
- **SavvyCAN:** SavvyCAN is a cross-platform QT-based C++ program. It is initially designed as a CAN bus reverse engineering and capture tool to work with EVTV hardware like EVTVDue/CANDue. Over time, it has expanded its compatibility to include any socketCAN compatible device, Macchina M2, and Teensy 3.x boards. This versatile tool allows capturing and sending data across multiple buses.

In addition, this study utilized a CANalyst-II device, a Raspberry Pi equipped with a CAN interface module, and a laptop (collect in-vehicle CAN bus data / inject attack traffic). For DoS, fuzzy, and replay attacks, the built-in software of CANalyst-II was used to inject the attacks. For masquerade attacks, scripts were written using the Python-CAN library to impersonate legitimate ECUs and send altered messages. For malfunction attacks, Python-CAN was used to send specially formatted CAN messages to induce faults. For spoof attacks,

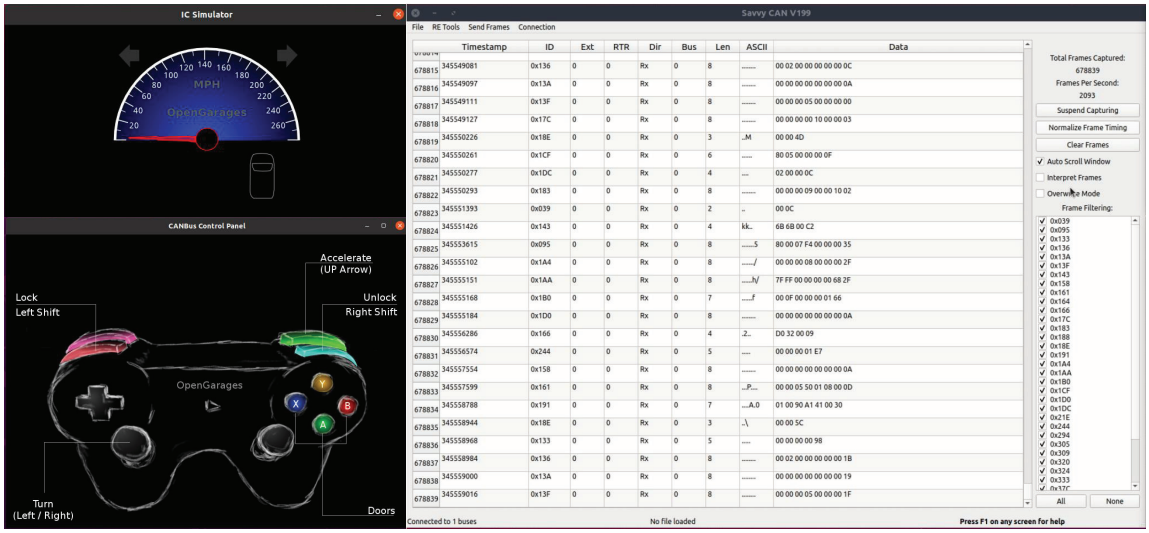


Fig. 7. The Simulated Environments of IVN (IC simulator+SavvyCAN)

the third-party Python library CANard was used to send customized spoofed CAN frames, mimicking legitimate ECU communication. For combination attack, a random combination of multiple attack types was employed to execute attacks.

Moreover, the CC-IDPS framework is designed specifically to provide a solution for a single automobile manufacturer, thus avoiding issues related to inter-manufacturer competition and proprietary protocol sharing. The framework ensures that all data processing and analysis remain within the boundaries of a single manufacturer's ecosystem. Looking forward, if cooperation among different manufacturers becomes necessary, the CC-IDPS has also demonstrated the feasibility of simultaneously processing CAN bus data from multiple manufacturers, provided that standardization and other related operations are performed before data is uploaded to the cloud.

This setup allowed us not only to simulate CAN bus data while the vehicle is in motion, but also to inject attacks into the CAN bus of vehicle. The simulation environment within the virtual machine loads the initially trained CC-IDPS model and classifies CAN traffic. Based on the classification results, the corresponding virtual machine drops identified malicious traffic, and uploads this traffic to the cloud server. Over time, the cloud server retrained the classification model based on the aggregated malicious traffic from the simulated environments. Specifically, retraining occurs at predefined intervals or triggers, such as when a significant amount of new data is accumulated, when there are changes in the attack patterns observed, or when performance degradation is detected in the existing model. Then, it distributed the updated model to the virtual machines for renewal.

B. Evaluation Metrics

To facilitate the comparison of performance between CC-IDPS and other algorithms, this paper primarily employs the following performance metrics: accuracy, precision, recall, F1-Score, loss, and Area Under Curve (AUC). These metrics effectively measure the prediction accuracy of IDPS. Their calculation formulas are as follows:

$$Accuracy = \frac{TRP + TRN}{TRP + TRN + FAP + FAN} \tag{14}$$

$$Precision = \frac{TRP}{TRP + FAP} \tag{15}$$

$$Recall = \frac{TRP}{TRP + FAN} \tag{16}$$

$$F1 - Score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{17}$$

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 calculation formula for binary classification using cross entropy is as follows:

$$BL = \frac{1}{n_b} \sum_{i}^{n_b} BL_i \tag{18}$$

$$BL_i = -\left[y_i \log\left(p_i\right) + \left(1 - y_i\right) \log\left(1 - p_i\right)\right] \tag{19}$$

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 i-th binary classification sample being predicted as the positive class.

AUC represents the area under the Receiver Operating Characteristic (ROC) curve. The ROC curve is a graphical plot that illustrates the trade-off between a model's true positive rate (sensitivity) and its false positive rate (1 - specificity) across

various threshold values. The AUC value ranges between 0 and 1, where a higher AUC score indicates better model performance at distinguishing between classes. An AUC of 1 represents a perfect classifier, while an AUC of 0.5 signifies a model with random performance.

C. Experimental Results

The main goal of in-vehicle intrusion detection is to swiftly pinpoint abnormal/malicious behavior on the CAN bus, indicating potential security risks. While multi-class classification tasks offer detailed insights into attack types, they pose challenges like complexity, computational burden, and dataset requirements. Therefore, the binary classification task is preferred for the CC-IDPS. The parameters selection of the BERT model used in CC-IDPS are shown in Table II.

TABLE II
THE PARAMETERS OF BERT

Parameters	Size
Encoder layers	12
Attention heads	12
Hidden size	768
Optimizer	Adam optimizer
Batch size	32
Learning rate	2e-5

For IVAD, this paper adopts various machine learning and deep learning models, including K-means, Logistic Regression (LR), Support Vector Machine (SVM), Decision Tree (DT), and CNN-LSTM, as comparative models. The reasons for selecting these models are as follows: K-means serves as a simple and intuitive clustering algorithm [30], LR offers fast training speed and is suitable for large datasets [31], SVM exhibits strong generalization capabilities [32], DT demonstrates robustness against outliers and missing values in data [33], and CNN-LSTM excels in handling complex multidimensional data while capturing spatial and temporal features within the data [34]. These models represent common and representative approaches in both machine learning and deep learning domains.

Fig. 8 depicts the detection results (AUC) of the same model on three datasets: CHD [15], SAD [16], and IVAD. It is evident that the performance of the identical model on IVAD appears relatively inferior. Several primary reasons account for this discrepancy: Firstly, the IVAD encompasses a broader array of attack types compared to CHD [15] and SAD [16]. Secondly, the IVAD possesses a substantially larger data volume than CHD [15] and SAD [16]. Thirdly, the IVAD contains data collected from both real and cloud-based simulation environments. In contrast, the CHD [15] solely includes DoS, fuzzy, and spoof attacks on the in-vehicle CAN bus, while the SAD [16] comprises flooding, fuzzy, and malfunction attack data. Collectively, these factors present challenges for the detection and classification models within IVAD, resulting in varying degrees of reduced performance across different models.

While larger window sizes may offer better detection accuracy by capturing more contextual information, it is essential to ensure that the computational overhead does not compromise

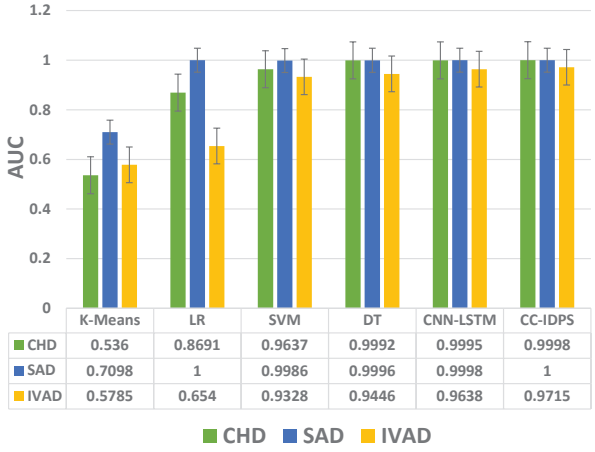


Fig. 8. The comparison of different models' AUC on CHD, SAD and IVAD

the real-time detection capabilities of the IDS. Therefore, it is crucial to strike a balance between the window size and detection efficiency. To verify the effectiveness of the sliding window mechanism in extracting CANID, the CC-IDPS conducted experiments by setting the sliding window sizes (w) to 1, 3, 5, and 7. As depicted in Fig. 9, it is evident that when w is set to 5 and 7, the detection accuracy, recall, precision, and F1-score of various models outperform the case when w is set to 1 and 3. As shown in Fig. 10, with the increase in the number of features, only K-Means exhibited classification difficulty and a decrease in AUC value. LR achieved the highest AUC value at w=3, while DT, CNN-LSTM, and CC-IDPS reached their peak AUC values at w=5, with AUC values of 0.9508, 0.9642, and 0.9886, respectively. The CC-IDPS attained its highest AUC value at w=5, but decreased to 0.9871 at w=7.

TABLE III
THE PARAMETERS OF CNN-LSTM ON IVAD (w=5)

Layer	Output Size	BN	Dropout	Activation
Input	13*1	✓	✓	LeakyReLu
Convolution	13*64	✓	✓	LeakyReLu
Convolution	13*32	✓	✓	LeakyReLu
LSTM	13*16			
Dense	13*8		✓	
Flatten	104		✓	
Dense	2			Sigmoid
Hyperparameters				
Epoch	10			
Batch Size	1000			
CNN Filter Size	64			
LSTM Units	16			
Dropout	0.3			
Learning Rate	0.0001			

Particularly, among the machine learning models, DT exhibits the best performance, achieving an AUC of 0.9508. This result is primarily attributed to the advantages of DT in interpretability, robustness, and suitability for various data types. Additionally, the CNN-LSTM model also achieves

This article has been accepted for publication in IEEE Transactions on Cognitive Communications and Networking. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCCN.2024.3516052

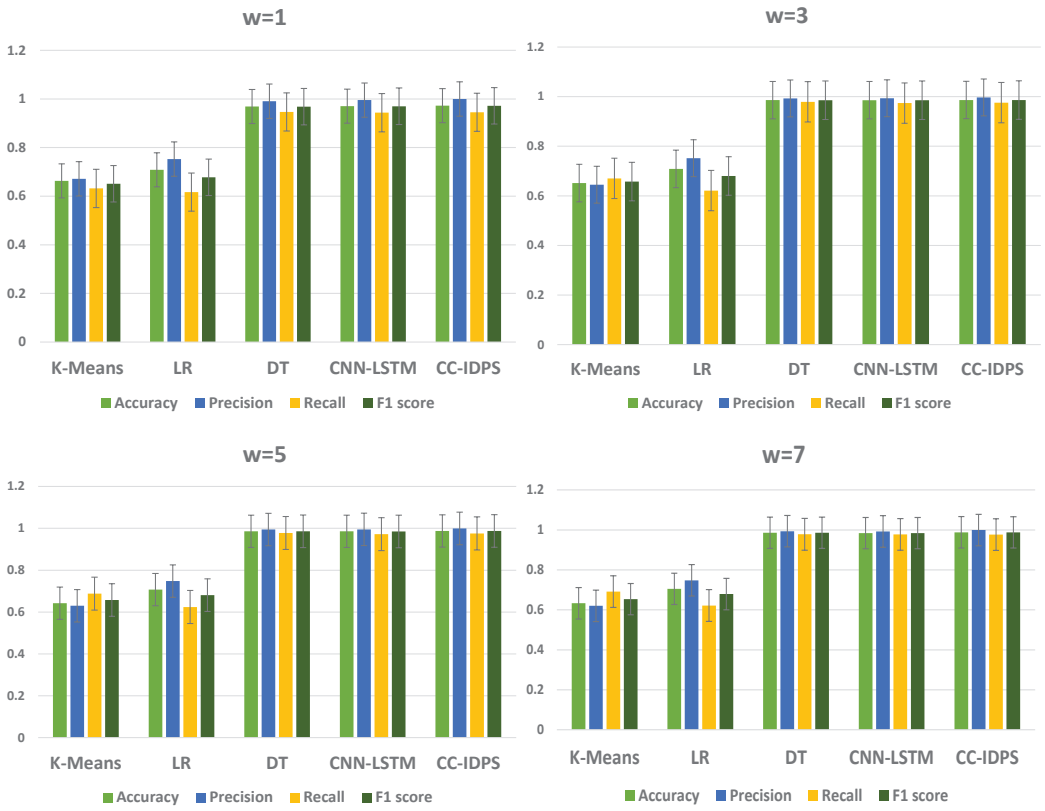


Fig. 9. The comparison of different window size on IVAD

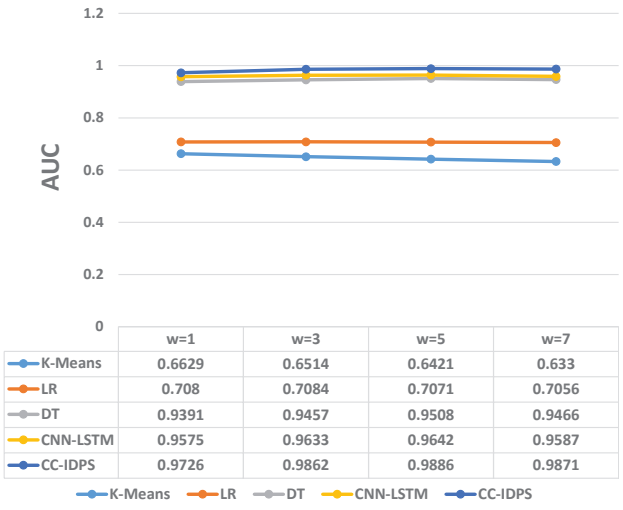


Fig. 10. The comparison of different window size on IVAD (change line chart of AUC)

an AUC of 0.9642, surpassing the performance of several machine learning models. Mainly because it can effectively handling complex multidimensional data and capturing spatial-temporal features within the data. Table III presents the CNN-LSTM model parameters for IVAD when $w = 5$. However, the F1-score of CNN-LSTM remains slightly lower than CC-IDPS model, which achieves an AUC of 0.9886. This significant improvement is attributed to BERT's use of a Transformer structure, incorporating pre-training techniques such as MLM. This allows BERT to comprehend comprehensive upper and lower CANID information within the window, not just local features.

TABLE IV
THE COMPARISON OF CC-IDPS AND OTHER PRE-TRAINED MODELS

Methods	Accuracy	Precision	Recall	F1-Score
GPT-2	0.9385	0.9436	0.9105	0.9273
XLNet	0.9617	0.9698	0.9122	0.9401
ERNIE	0.9544	0.9628	0.9379	0.9512
BERT	0.9927	0.9905	0.9861	0.9886

To validate the advantages of the CC-IDPS model compared to other NLP pre-trained models, this study selected GPT-2, XLNet, and Enhanced Representation through Knowledge Integration (ERNIE) for comparison. As shown in Fig. 11 and Table IV, the CC-IDPS model significantly outperformed the other three models in terms of accuracy, precision, recall, and F1-score. Among them, the GPT-2 model performed the worst, with an accuracy and F1-score of only 0.9385 and 0.9273, respectively. The XLNet and ERNIE models demonstrated similar performance, both achieving accuracy and F1-score around 0.95.

To validate the effectiveness of the proposed CC-IDPS framework, this study replicated the methods from [34], [35], [36] and [37] for comparison. Table V presents the perfor-

TABLE V
THE COMPARISON OF CC-IDPS AND OTHER LITERATURES UNDER INITIAL/UPDATED DATASETS

Methods	Accuracy (initial)	F1-Score (initial)	Accuracy (updated)	F1-Score (updated)
CNN-LSTM (with attention) [34]	97.98%	0.9706	96.05%	0.9572
Multi-scale Fusion [35]	99.06%	0.9873	97.35%	0.9709
ECF-IDS [36]	99.26%	0.9888	97.41%	0.9722
CE ResNet [37]	98.66%	0.9837	95.40%	0.9517
SupCon ResNet [37]	98.82%	0.9855	96.32%	0.9598
CC-IDPS	99.27%	0.9886	99.02%	0.9863

TABLE VI
DETECTION RESULTS OF CC-IDPS AND SOTA METHODS AGAINST UNKNOWN ATTACKS

Type	Normal - Accuracy	Normal - F1-Score	Unknown Attack - Accuracy	Unknown Attack - F1-Score
CNN-LSTM (with attention) [34]	98.43%	0.9822	95.14%	0.9489
Multi-scale Fusion [35]	99.26%	0.9895	96.37%	0.9608
ECF-IDS [36]	99.86%	0.9985	97.28%	0.9711
CE ResNet [37]	98.75%	0.9854	96.41%	0.9637
SupCon ResNet [37]	99.14%	0.9907	97.26%	0.9712
CC-IDPS	99.88%	0.9986	98.26%	0.9804

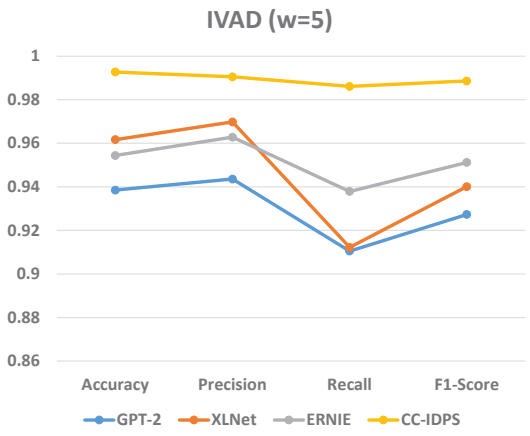


Fig. 11. The comparison of CC-IDPS and other pre-trained models

mance of six methods on both the initial dataset (IVAD) and the updated dataset. The updated dataset primarily involves a combination and replacement of other attacks within the combination attack of IVAD. As shown in Table V, in the results comparison on the initial dataset, the detection performance of Multi-scale Fusion [35], ECF-IDS [36], CE ResNet [37], SupCon ResNet [37] and CC-IDPS is nearly identical, with both outperforming the CNN-LSTM (with attention) [34] method. However, under the updated dataset, the accuracy and F1-score of all six methods declined. Nevertheless, due to the feedback update mechanism of CC-IDPS, its performance remained more stable and significantly higher than the other five methods.

To further evaluate the robustness of the CC-IDPS model, we conducted additional experiments to test its performance against unknown attacks. These experiments included replayed DoS attacks, a combination of fuzzy and malfunction attack data, and attacks involving the permutation of CAN IDs for gear spoofing and RPM spoofing. As shown in Table VI, the CC-IDPS model achieved a detection accuracy of 99.88% for normal data, which is consistent with the high performance seen in previous tests. More importantly, even when exposed to unknown attack scenarios, the CC-IDPS maintained an impressive detection accuracy of 98.26%. When compared with other SOTA methods, the CC-IDPS outperformed the Multi-scale Fusion [35], ECF-IDS [36], CE ResNet [37], and SupCon ResNet [37] models in both detection accuracy and F1-score. Specifically, the F1-score for CC-IDPS was significantly higher, indicating not only a greater ability to correctly identify attack traffic but also fewer false positives and false negatives.

D. Cost of CC-IDPS

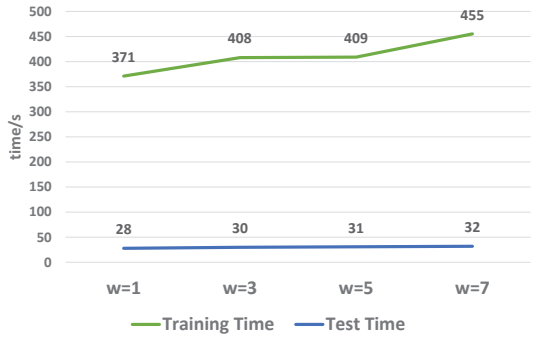


Fig. 12. The training/test time of CC-IDPS on different window size

Taking into account the comprehensive analysis of the cloud server training and virtual machine simulation environment testing times depicted in Fig. 12, adopting $w = 5$ achieves the best overall performance. This selection ensures optimal detection accuracy metrics while maintaining efficient detection times. Fig. 13 displays the AUC curve comparisons of

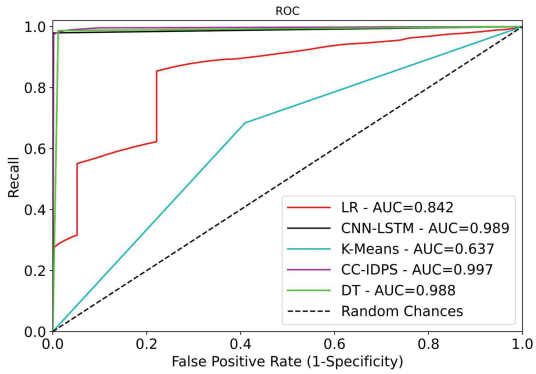


Fig. 13. The AUC of models on IVAD (w=5)

various models when w = 5, where the CC-IDPS model exhibits the highest AUC, reaching up to 0.997. The AUC of CNN-LSTM also reached 0.989, second only to CC-IDPS, and higher than all other models. Furthermore, DT, within the scope of machine learning, achieved the highest AUC of 0.988. Despite a marginal 0.008 AUC difference between CC-IDPS and the second-ranking CNN-LSTM, within the cloud-vehicle collaborative system handling massive IVN traffic, this slight variation significantly mitigates false positives. Next, Fig. 14 illustrates the loss curve during training iterations. It demonstrates that the training loss quickly decreases from an initial value of 0.7 to approximately 0.05 within the first 2000 iterations and then gradually stabilizes.

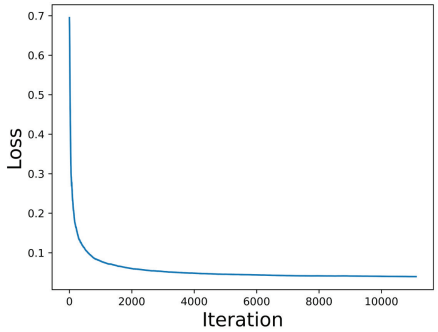


Fig. 14. The training loss of CC-IDPS on IVAD (w=5)

TABLE VII
COMPARISON OF ENCODE/ENCRYPTION ALGORITHMS

Algorithms	Key	Output length(bytes)	Reverse Decryption
MD5	No	16	No
SHA-256	No	32	No
DES	Yes	168	Yes
RSA(2048)	Yes	256	Yes
$Encode2ID$	No	85	Yes

Regarding the feedback and defense mechanism, after calculating, assuming the storage of an original data sequence '79, 320, 305, 304, 1349, 69, 41, 36, 255, 41, 36, 0, 255, 1', which occupies 168 bytes, after applying the $Encode2ID$ algorithm described in this paper, it transforms into 'BfgjOZoHPJK7lfraNXPl58eSWrS3syjmOcxR', occupying 85 bytes. Consequently, the $Encode2ID$ algorithm in this study saves 49.4% of storage space. At the same time, the $Encode2ID$ algorithm also serves a certain privacy protection, preventing the traffic from being stored in plain text within the cloud-based database. For the aforementioned array occupying 168 bytes, the $Encode2ID$ algorithm is compared with traditional cryptographic algorithms such as MD5, SHA-256, DES, and RSA. As shown in Table VII, although MD5 and SHA-256 message digest algorithms are more space-efficient, they lack the capability of providing decryption reversibility, meaning original plaintext cannot be retrieved. In comparison to DES and RSA algorithms, the $Encode2ID$ algorithm not only operates without requiring a key, but also generates strings occupying less space.

TABLE VIII
TRAINING/TEST TIME OF CC-IDPS

Cloud training	Vehicle test
1.133 (ms/each traffic)	0.203 (ms/each traffic)
403s (355539 traffic flow)	31s (152375 traffic flow)
Model size	108.5MB

As shown in Table VIII, the total training time of CC-IDPS on the cloud server is 403s, involving the training of 355,539 samples over 11,110 iterations. Simultaneously, the total testing time on the vehicle end is 31s, encompassing the testing of 152,375 samples. These results highlight that leveraging the abundant computational resources of the cloud server, the training time of CC-IDPS model on the cloud server is 1.133ms per traffic flow. Furthermore, the detection time per traffic flow on the vehicle end is 0.203ms. The total time from feeding attack traffic into the model to issuing a warning is 0.286ms. Since the computational overhead on the cloud primarily occurs during model training on the cloud server, it does not impose a significant computational burden on the vehicle end. The vehicle end is mainly responsible for traffic detection based on the loaded model, which greatly enhances detection efficiency and saves computational resources on the vehicle end.

VI. DISCUSSION

In cloud-vehicle collaborative scenarios, the communication latency is influenced by the specific network environment. The results of this study indicate that, within a small local area network, the communication latency from vehicles to cloud servers is typically maintained around 3 ms. This low latency is advantageous for real-time intrusion detection, allowing for swift response times to potential threats. However, in practical deployments, the communication path between cloud services and vehicles may involve intermediaries such as Roadside Units (RSUs). The inclusion of these devices could introduce additional latency, potentially affecting the performance of the CC-IDPS. Future research should investigate the implications of varied network topologies on communication latency to optimize system performance under different conditions.

Furthermore, the dynamic nature of attack strategies in the field of in-vehicle intrusion detection necessitates continuous updates to detection models. Regular database updates are critical to maintaining the relevance and effectiveness of these models against evolving threats. Ideally, updates should occur at intervals that reflect the changing threat landscape, with common practices including daily or weekly refreshes to incorporate newly collected data. In situations characterized by rapid shifts in attack patterns or an influx of new attack vectors, more frequent updates may be required. This adaptability is vital for ensuring that the CC-IDPS can identify and respond to novel threats effectively. Future work should focus on developing adaptive algorithms that can facilitate real-time updates, thereby enhancing the resilience of the IDS against emerging vulnerabilities in the automotive domain.

VII. Related Work

The relevant work on in-vehicle IDSs has garnered significant attention in both academia and the industry. Some studies have focused on designing intelligent algorithms and machine learning models to identify abnormal behavior and network attacks. These approaches encompass statistical analysis, machine learning, deep learning, among other techniques, aiming to achieve real-time monitoring and threat detection within IVNs. Furthermore, researchers are also concentrating on aspects such as feature extraction from IVN data, model training, and data labeling to construct more accurate and reliable IDSs.

A. Intrusion Detection Methods Based on Time Series Prediction and Spatial-Temporal Features

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) [10] was introduced. This model integrates Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) structures to leverage spatial-temporal representations of traffic in an IVN. The CANnolo [38] was proposed to identify anomalies in CAN. It employs LSTM-autoencoders to analyze the time and data sequences of CAN. In addition, Qin et al. [39] 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.

B. Intrusion Detection Methods Based on BERT Model

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 [40] is designed to learn arbitration ID sequences for intrusion detection on the CAN bus. It was evaluated using the "Car Hacking: Attack & Defense Challenge 2020" dataset [41], achieving F1-Score ranging from 81% to 99%. Another BERT-based IDS model is CANBERT [42], which was tested using the OTIDS dataset [43] and achieved an accuracy rate of over 99%.

Compared to the CAN-BERT [40] and CANBERT [42] models, the main innovation of our CC-IDPS model lies in its utilization of cloud resources for training and updating the BERT model, while CAN-BERT and CANBERT are limited to offline training, lacking the capability for real-time model updates. Meanwhile, CAN-BERT and CANBERT lack subsequent deep analysis and processing operations for post-processed malicious traffic. Furthermore, Our IVAD dataset has a broader source base and richer feature set compared to the datasets utilized by CAN-BERT and CANBERT.

C. Intrusion Detection Methods Based on CNN Model

Utilizing a CNN as the base model in combination with other methods is also a promising choice for intrusion detection. The CANintelligentIDS model [44] 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. Besides, the fusion of recurrence plots with CNN for in-vehicle intrusion detection represents a favorable choice. A Rec-CNN model [11] 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.

D. Intrusion Detection Methods Based on Collaborative Model

There is scarce literature on collaborative intrusion detection within IVNs utilizing cloud computing resources. In 2023, Qin et al. [45] primarily investigated a cloud-vehicle collaborative IDS based on multi-dimensional features for the Internet-of-Vehicle (IoV), known as CVMIDS. To address the heterogeneity of vehicle data, the CVMIDS abstracts disparate vehicle data into a common feature space, thereby enabling model training based on multi-dimensional features and detection of multiple attack categories. By establishing a feature space that combines dimensions such as time, traffic, and voltage, the CVMIDS expanded its capability to detect various types of attacks, demonstrating its robustness and efficiency through experimental validation.

E. Intrusion Prevention System (IPS)

The research on IPS for in-vehicle CAN buses has also attracted considerable attention and participation. In 2021, De Araujo-Filho et al. [46] proposed an unsupervised IPS for automotive CANs to detect and hinder attacks without altering the ECUs' architecture or requiring restricted information. It evaluates two machine learning algorithms for detecting fuzzy and spoof attacks, achieving high accuracy and F1-score with minimal data bytes. In 2019, Olufowobi et al. [27] presented a novel algorithm for detecting and recovering from message spoof attacks targeting the CAN bus, a critical component of IVNs. Leveraging the predictable runtime behavior of CAN

TABLE IX
COMPARISON OF RELATED WORK

Related Work	Type of IDS	Datasets	Method	Prevention	Cloud collaborative
[10]	Hybrid Deep Learning-IDS	CHD	CNN, LSTM	No	No
[38]	LSTM-based IDS	ReCAN	LSTM (autoencoders)	No	No
[39]	Anomaly-based IDS	Private Dataset	LSTM	No	No
[40]	BERT-based IDS	Car Hacking: Attack & Defense Challenge 2020	BERT	No	No
[42]	BERT-based IDS	OTIDS	BERT	No	No
[44]	Deep Learning IDS	OTIDS	CNN, GRU	No	No
[11]	Recurrence Plots IDS	CHD, Private Dataset	Recurrence Plots, CNN	No	No
[45]	Cloud-Vehicle Collaborative-IDS	Private Dataset	XGBoost	No	Yes
[46]	Unsupervised IPS	OTIDS	Isolation Forest	Yes	No
[27]	IPS	Private	Proposed Recovery Approach	Yes	No
Proposed	Cloud-Vehicle Collaborative-IDPS	IVAD	BERT	Yes	Yes

message frames, the algorithm utilizes the CAN bus's error handling capability for reboot-based recovery of compromised network nodes.

F. Motivation

Table IX summarizes the related works in existing studies pertaining to in-vehicle IDSs. It can be inferred that despite the proliferation of research on in-vehicle IDSs in recent years, most of these systems have primarily accomplished the task of detection and classification, lacking subsequent defensive actions against malicious traffic. Meanwhile, while these works predominantly utilize datasets from HCRL and private sources, the range of attack types within these datasets is limited, posing insufficient challenges for their models.

Moreover, there is a considerable scarcity of research focusing on utilizing abundant cloud computational resources for model training in IVN IDSs. The advantages of IVN intrusion detection with the help of cloud resources are as follows: 1) Cloud-based solutions offer scalable computational resources that can handle the intensive processing requirements of advanced intrusion detection algorithms, such as those based on deep learning. 2) Leveraging cloud infrastructure allows for the aggregation and analysis of threat intelligence from multiple vehicles, enabling faster detection of emerging threats and the dissemination of updates to all connected vehicles. 3) Cloud platforms facilitate the centralized training of sophisticated models like BERT, ensuring they are continuously updated with the latest data. These updated models can then be distributed to vehicles via Over-The-Air (OTA) updates, ensuring all vehicles benefit from the most current intrusion detection capabilities.

VIII. CONCLUSION

This paper constructs the IVAD by integrating real-world scenarios, simulated cloud vehicle environments, and multiple data samples from CHD [15] and SAD [16]. Additionally, the proposed CC-IDPS fully leverages the advantages of cloud computing, multi-dimensional features, and cloud-vehicle collaboration to enhance the comprehensiveness and precision of detecting IVN attacks. Initially, the CC-IDPS extracts CANID information utilizing a sliding window mechanism and employs cloud servers to train the BERT model, distributing it to vehicles for the detection and classification of CAN bus traffic. Subsequently, through the $VehicleDetection$ algorithm, vehicles are informed to take appropriate defensive actions against malicious attack traffic. Meanwhile, the $Encode2ID$ and $DecodeID$ algorithms proposed in this paper are employed to store related malicious traffic. This approach not only saves space but also achieves certain privacy protection purposes. The stored malicious traffic can be utilized for subsequent training and updating of the BERT model. This research aims to provide comprehensive and innovative solutions for the future security of IVNs, contributing to the development of safer and more reliable smart vehicles.

REFERENCES

[1] R. Jin, X. He, and H. Dai, "Collaborative ids configuration: A two-layer game-theoretic approach," *IEEE Transactions on Cognitive Communications and Networking*, vol. 4, no. 4, pp. 803–815, 2018.

[2] M. S. E. Sayed, N.-A. Le-Khac, M. A. Azer, and A. D. Jurcut, "A flow-based anomaly detection approach with feature selection method against ddos attacks in sdns," *IEEE Transactions on Cognitive Communications and Networking*, vol. 8, no. 4, pp. 1862–1880, 2022.

[3] 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*, vol. 238, p. 122198, 2024.

[4] 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.

[5] X. Wang, M. He, J. Wang, and X. Wang, "Towards efficient neural networks through predictor-assisted nsga-iii for anomaly traffic detection of iot," *IEEE Transactions on Cognitive Communications and Networking*, pp. 1–1, 2024.

[6] (2022) Teen hacker says he's found way to remotely control 25 tesla evs around the world. [Online]. Available: https://fortune.com/2022/01/12/teen-hacker-david-colombo-took-control-25-tesla-ev/

[7] D. Keuper and T. Alkemade, "'the connected car ways to get unauthorized access and potential implications," *Computest, Zoetermeer, The Netherlands, Tech. Rep*, 2018.

[8] M. Tiwari, I. Maity, and S. Misra, "Rate: Reliability-aware task service in fog-enabled iov environments," *IEEE Transactions on Cognitive Communications and Networking*, pp. 1–1, 2024.

[9] 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.

[10] 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.

[11] 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.

[12] Z. Li, P. Wang, Z. Wang, and D.-c. Zhan, "Flowgananomaly: Flow-based anomaly network intrusion detection with adversarial learning," *Chinese Journal of Electronics*, vol. 33, no. 1, pp. 58–71, 2024.

[13] Z. Zhang, F. Liu, Y. Ge, S. Li, Y. Zhang, and K. Xiong, "An intrusion detection method based on depthwise separable convolution and attention mechanism," *Chinese Journal on Internet of Things*, vol. 7, no. 1, pp. 49–59, 2023.

[14] 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.

[15] 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.

[16] M. L. Han, B. I. Kwak, and H. K. Kim, "Anomaly intrusion detection method for vehicular networks based on survival analysis," *Vehicular communications*, vol. 14, pp. 52–63, 2018.

[17] S.-F. Lokman, A. T. Othman, and M.-H. Abu-Bakar, "Intrusion detection system for automotive controller area network (can) bus system: a review," *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, pp. 1–17, 2019.

[18] S. Woo, H. J. Jo, I. S. Kim, and D. H. Lee, "A practical security architecture for in-vehicle can-fd," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 8, pp. 2248–2261, 2016.

[19] R. Malekian, N. R. Moloisane, L. Nair, B. T. Maharaj, and U. A. Chude-Okonkwo, "Design and implementation of a wireless obd ii fleet management system," *IEEE Sensors Journal*, vol. 17, no. 4, pp. 1154–1164, 2016.

[20] K. L. Lim, J. Whitehead, D. Jia, and Z. Zheng, "State of data platforms for connected vehicles and infrastructures," *Communications in transportation research*, vol. 1, p. 100013, 2021.

[21] Y. Koh, S. Kim, Y. Kim, I. Oh, and K. Yim, "Efficient can dataset collection method for accurate security threat analysis on vehicle internal network," in *International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. Springer, 2022, pp. 97–107.

[22] 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.

[23] 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.

[24] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

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

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

[27] H. Olufowobi, S. Hounsinou, and G. Bloom, "Controller area network intrusion prevention system leveraging fault recovery," in *Proceedings of the ACM Workshop on Cyber-Physical Systems Security & Privacy*, 2019, pp. 63–73.

[28] S. Craig. (2020) Icsim: Instrument cluster simulator for socketcan. [Online]. Available: https://github.com/zombieCraig/ICSim

[29] C. Kidder. (2020) Savvycan: Qt based cross platform canbus tool. [Online]. Available: https://github.com/collin80/SavvyCAN

[30] H. Narasimhan, R. Vinayakumar, and N. Mohammad, "Unsupervised deep learning approach for in-vehicle intrusion detection system," *IEEE Consumer Electronics Magazine*, 2021.

[31] D. Tanaka, M. Yamada, H. Kashima, T. Kishikawa, T. Haga, and T. Sasaki, "In-vehicle network intrusion detection and explanation using density ratio estimation," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 2238–2243.

[32] A. Derhab, M. Belaoued, I. Mohiuddin, F. Kurniawan, and M. K. Khan, "Histogram-based intrusion detection and filtering framework for secure and safe in-vehicle networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 3, pp. 2366–2379, 2021.

[33] Y. Jeong, H. Kim, S. Lee, W. Choi, D. H. Lee, and H. J. Jo, "In-vehicle network intrusion detection system using can frame-aware features," *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[34] H. Sun, M. Chen, J. Weng, Z. Liu, and G. Geng, "Anomaly detection for in-vehicle network using cnn-lstm with attention mechanism," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 10, pp. 10 880–10 893, 2021.

[35] J. Zhou, S. Li, Y. Cao, H. J. Hadi, and H. Lin, "Robust intrusion detection system in can bus through multi-scale feature fusion," in *ICC 2024-IEEE International Conference on Communications*. IEEE, 2024, pp. 1316–1321.

[36] S. Li, Y. Cao, H. J. Hadi, F. Hao, F. B. Hussain, and L. Chen, "Ecf-ids: An enhanced cuckoo filter-based intrusion detection system for in-vehicle network," *IEEE Transactions on Network and Service Management*, 2024.

[37] T.-N. Hoang and D. Kim, "Supervised contrastive resnet and transfer learning for the in-vehicle intrusion detection system," *Expert Systems with Applications*, vol. 238, p. 122181, 2024.

[38] 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.

[39] 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.

[40] 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.

[41] H. Kang, B.-I. Kwak, Y. Lee, H. Lee, H. Lee, and H. K. Kim, "Car hacking and defense competition on in-vehicle network," 01 2021.

[42] 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.

[43] H. Lee, S. H. Jeong, and H. K. Kim, "Otids: A novel intrusion detection system for in-vehicle network by using remote frame," in *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, 2017, pp. 57–5709.

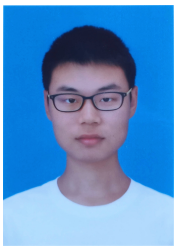
[44] A. R. Javed, S. Ur Rehman, M. U. Khan, M. Alazab, and T. Reddy, "Canintelliids: Detecting in-vehicle intrusion attacks on a controller area network using cnn and attention-based gru," *IEEE transactions on network science and engineering*, vol. 8, no. 2, pp. 1456–1466, 2021.

[45] J. Qin, Y. Xun, and J. Liu, "Cvmids: Cloud-vehicle collaborative intrusion detection system for internet-of-vehicles," *IEEE Internet of Things Journal*, 2023.

[46] P. F. De Araujo-Filho, A. J. Pinheiro, G. Kaddoum, D. R. Campelo, and F. L. Soares, "An efficient intrusion prevention system for can: Hindering cyber-attacks with a low-cost platform," *IEEE Access*, vol. 9, pp. 166 855–166 869, 2021.

Sifan Li is currently pursuing the M. S. degree at the School of Cyber Science and Engineering, Wuhan University, China. He received his B.S. degree from Nanchang University in 2021. His research interests include IoT Intrusion Detection, In-vehicle security, and Aerospace-Sky-Earth Security.

Yue Cao (M'16, SM'21) received the Ph.D. degree from the Institute for Communication Systems (ICS) formerly known as Centre for Communication Systems Research, University of Surrey, Guildford, U.K., in 2013. Further to his PhD study, he had conducted a Research Fellow with the University of Surrey, and academic faculty with Northumbria University, U.K., Lancaster University, U.K., and Beihang University, Beijing, China. He is currently a Professor with the School of Cyber Science and Engineering, Wuhan University, Wuhan, China. His research interests include intelligent transport systems, including E-Mobility, V2X, and edge computing.



Yu'ang Zhang received a B.Eng. degree in 2022 and is currently pursuing a M.Sc. degree at School of Cyber Science and Engineering, Wuhan University, Wuhan, China. His research interests focus on the misbehavior detection in V2X applications.

Tongxin Liao is currently pursuing the E.D. degree at the School of Electronic Information and Communication, Huazhong University of Science and Technology, China. She received her M. S. degree from Hunan University in 2023. Her research interests include cloud computing and blockchain.

Fei Yan received the Ph.D. degree from Wuhan University, Wuhan, China, in 2007. He is currently an Associate Professor with the the School of Cyber Science and Engineering, Wuhan University. He is also a Co-Founder of ChinaSigTC (China special interest group on trusted cloud) and was the Associate Chair of Program Committee of CTCIS (Chinse Trusted Computing and Information Security Conference) from 2017 to 2019. His research interests include system security, trusted computing, and blockchain.

Hai Lin (Member, IEEE) received the B.S. degree from the Department of Thermal Engineering, Huazhong Sciences and Technologies University, Wuhan, China, in 1999, the M.S. degree in computer science from the University of Pierre and Marie Curie, Paris, France, in 2005, and the Ph.D. degree from the Institute Telecom–Telecom ParisTech, Paris, in 2008. He held a Post-Doctoral Research with France Telecom. He was a Researcher with ZTE Europe. Since 2012, he has been with Wuhan University, Wuhan, where he is currently an Associate Professor with the School of Cyber Science and Engineering. His research interests include the Internet of Things, edge computing, sensor networks, and future networks.