IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

IEEE VEHICULAR TECHNOLOGY SOCIETY SECTION

Received 30 July 2024, accepted 23 August 2024, date of publication 30 August 2024, date of current version 10 September 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3452634

RESEARCH ARTICLE

Intrusion Detection Using Transformer in Controller Area Network

HYUNJUN JO AND DEOK-HWAN KIM, (Member, IEEE)
Department of Electrical and Computer Engineering, Inha University, Incheon 22212, South Korea
Corresponding author: Deok-Hwan Kim (deokhwan@inha.ac.kr)

This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by Korean Government (MSIT) under Grant RS-2024-00336286, and in part by the Inha University Research Grant.

ABSTRACT The message broadcast network of the Controller Area Network (CAN) protocol is vulnerable to external attacks. The ongoing development of intrusion detection systems (IDS) aims to prevent malicious attacks on vehicles. Time series analysis of language models has emerged as a new approach in this area and has significantly contributed to the development of IDS performance. Nevertheless, because the language model requires significant resources to process, its application to actual vehicles requires balancing model performance with complexity. In this paper, we propose an efficient IDS model that uses transformer-based techniques while operating with limited resources. The proposed IDS leverages a transformer-based spatial and temporal data analysis mechanism, enabling quick response to attacks even with limited data, and demonstrates excellent performance. Since the IDS uses unsupervised learning, labeling the input sequence during preprocessing is not required. This approach helps protect the vehicle from both predictable and unpredictable attacks. Furthermore, the prediction range can be expanded to make the model's performance more robust against various attack scenarios.

INDEX TERMS Cybersecurity, controller area network, CAN, intrusion detection, transformer, unsupervised learning.

I. INTRODUCTION

With the recent emergence of connected cars, vehicles are no longer independent means of transportation, but have become mobile computers connected to wireless networks. The connected cars, once limited to accessing customer service centers through vehicle communication terminals, have now evolved to support cross-industry services such as real-time route guidance, remote ignition-on, self-diagnosis, and various financial and IT services. However, as the number of connected cars rapidly increases, so do concerns about cyberattacks targeting them. Because cyberattacks can threaten road safety, there is an urgent need to implement legal and institutional measures to ensure the cybersecurity of connected cars. UN Regulation No. 155 [1] is the international safety regulation developed by the United Nations Economic Commission for Europe (UNECE) for cybersecurity in vehicles. The goal of the regulation is to enhance the security of connected and automated vehicles and to provide a common cybersecurity framework for the entire automotive industry. The cybersecurity management system requires a systematic and risk-based approach to defining organizational processes, responsibilities, and governance for managing cyber threats and protecting vehicles from cyberattacks. One of the issues is detecting and recovering from a denial of service (DoS) attack [2], which can be triggered on internal networks by flooding a controller area network (CAN) bus or by provoking faults in an electronic control unit (ECU) via high rates of messaging. In most cases, these attacks can be launched either online or offline and may be injected through the CAN bus.

There are several algorithms for detecting injection attacks through CAN. Gaussian density estimation [3] is used as a statistical method. However, it is difficult to find an appropriate probability distribution, and it does not work well with multi-dimensional data. The support vector machine [4]

The associate editor coordinating the review of this manuscript and approving it for publication was Jie Gao.

© 2024 The Authors. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

is a machine learning technique used to identify the boundary that separates normal and abnormal data. However, it requires labeled data and can handle non-linear classification problems by using kernel functions. In contrast, deep learning techniques can handle large amounts of high-dimensional data. For time-series data, models specialized for sequence data, such as temporal convolutional neural networks (CNNs), recurrent neural networks (RNNs), long short-term memory networks (LSTMs), and transformers, are commonly used to identify correlations.

The transformer has been used in natural language processing, demonstrating high performance in various tasks such as translation, summarization, and question answering. We propose a new intrusion detection system (IDS) that leverages the ability of effectively capturing correlations between words with high contextual relevance. The main contributions of this paper are as follows.

1) We propose an IDS based on the transformer that predicts the next data point according to the flow of previously input data. To minimize preprocessing work, it only requires simple editing such as converting data formats, appending, setting null values for tokenizing.
2) CAN data consist of temporal data recorded over time (such as changes in vehicle speed) and spatial data recorded among devices (such as the status or command of each ECU). The two-dimensional data are used to train our model. Thus, the performance is higher than when using one-dimensional data, and any attacks on temporal and spatial data can be detected.
3) To defend against both predictable and unpredictable attacks simultaneously, the model is updated using an unsupervised learning method. This allows a large amount of data to be trained sequentially without the need for labeling.
4) To increase performance under the same conditions, we present a method to sort the values of the final output dimension in descending order and expand the prediction range starting from the highest value.

This paper is organized as follows. Section II presents the background of controller area network and language model. Section III introduces intrusion detection methods in controller area networks using various deep learning models. Section IV proposes the transformer-based spatial and temporal data analysis method. Section V describes the experimental results and compares the proposed method with other methods. Finally, Section VI and VII present the conclusion and discussion.

II. BACKGROUND

This section describes an overview of the CAN protocol and common attack scenarios, and introduces the basic mechanisms of language models.

A. CONTROLLER AREA NETWORK

In the past, automobile communications had a system structure where each module communicated via a universal asynchronous receiver transmitter (UART), but as more functions were added, the number of connection lines increased proportionally, making communication between modules more complex. To address this complexity, the CAN protocol was developed. The biggest advantage of CAN communication is that modules can easily be attached or removed by simply connecting them with two wires to the CAN bus.

ECUs in a vehicle communicate using the CAN protocol. CAN enables microcontroller devices within a vehicle to communicate directly without needing a host computer [5]. As shown in Fig. 1, the CAN bus consists of CAN-high and CAN-low twisted pair wires that connect each ECU node. When the node transmission is dominant, 3.5V is applied to the CAN-high wire and 1.5V is applied to the CAN-low wire. When the node transmission is recessive, 2.5 V is applied to both wires [6]. The CAN packets (messages) are transmitted using a broadcast method because they do not have specific departure or destination addresses. When nodes receive messages, they determine whether to accept them based on the CAN identifier (ID) [7]. The messages (CAN data frames) containing information such as speed, braking, and engine RPM are broadcast over the bus.

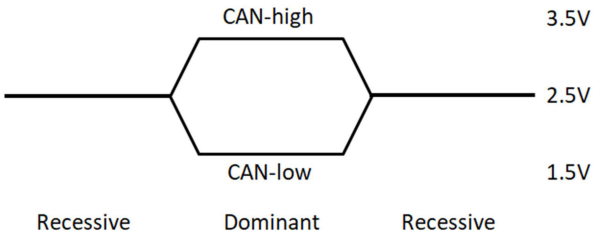

FIGURE 1. The controller area network bus.

SOF	Arbitration field	Control field	Data field	CRC field	ACK field	EOF
SOF	ID	DLC	DATA	CRC	ACK	EOF

FIGURE 2. Structure of the CAN data frame.

Fig. 2 shows the structure of the CAN data frame, with each constituent field serving the following functions [8]:

SOF: the start-of-frame bit initiates a new message and synchronizes all nodes on the bus.

Arbitration field: the CAN ID used in the arbitration process can be either a standard frame (11 bits) or an extended frame (29 bits).

Control field: the Data Length Code (DLC) indicates the number of bytes in the data field.

Data field: the payload contains the information communicated among ECUs and can be up to 64 bits.

CRC field: the Cyclic Redundancy Check (CRC) is a checksum used for detecting errors in the data.

ACK field:the transmitter sends a recessive bit to indicate that there are no errors in the received message, serving as an acknowledgment.

EOF: the end-of-frame marks the end of a CAN frame.

However, the CAN protocol is vulnerable because it lacks self-defense measures, such as authentication or encryption, to protect against various attacks [9]. For example, attackers may attempt to inject messages to execute commands in an ECU, either online or offline. Common attack scenarios on the CAN bus are as follows.

The DoS(flooding) attack disrupts normal service by sending a large number of valid requests to the target system, causing the service to terminate functions.

The fuzzy attack randomly generates CAN messages, including both the arbitration field and the data field. Randomly generated CAN IDs range from 0×000 to $0 \times 7FF$ and include both legitimate and non-legitimate IDs.

The replay attack extracts legitimate messages from the CAN bus over a certain period and then injects them to cause unexpected activity.

The spoofing (or malfunction) attack involves an attacker obtaining valid CAN IDs and altering their data fields to cause unintended functions.

B. LANGUAGE MODEL

A language model predicts the words likely to appear in a specific position within a sentence by probabilistically assessing how natural the sentence is. For example, the language model predicts the next word in a sentence based on the preceding words to ensure the sentence flows naturally. To leverage these characteristics, the words and sentences are replaced with the CAN data and the CAN sequences, respectively.

Corpus	I am going to attend the meeting
Pieces	_I, _am, _going, _to, _attend, _the, _meeting
Tokens	6, 172, 151, 8, 1118, 5, 635

FIGURE 3. Tokenization.

Tokenization is required before input data are entered into a language model. This process involves splitting a corpus into smaller units called tokens. Fig. 3 shows how tokenization works using SentencePiece [10].

TABLE 1. The structure of the can data.

Time stamp	ID	DLC	Data
1479121434.850202	0350	8	05 28 84 66 6d 00 00 a2
1479121434.850423	02c0	8	14 00 00 00 00 00 00 00
1479121434.850977	0430	8	00 00 00 00 00 00 00 00
1479121434.851215	04b1	8	00 00 00 00 00 00 00 00
1479121434.851463	01f1	8	00 00 00 00 00 00 00 00
1479121434.851711	0153	8	00 00 00 ff 00 ff 00 00
1479121434.851963	0002	8	00 00 00 00 00 00 00 0a

In Table 1, the CAN ID and the payload in the data field have been tokenized into hexadecimal format.

A sequence is a group of objects listed in order. A sentence can be considered a sequence of words, with the order of the words taken into account. The sequence W, consisting of n words, can be expressed probabilistically as follows:

$$p(W) = p(w_1, w_2, w_3, \ldots, w_n) \tag{1}$$

The most common way to assign probabilities to a sequence is to have the language model predict the next word based on the previous words:

$$\prod_{i=1}^{n} p(w_i | w_1, w_2, w_3, \ldots, w_{i-1}) \tag{2}$$

Those formulas apply to the CAN message sequence, because the process of arranging information generated from ECUs is quite similar to the process of language communication, which can be modeled using probabilistic language sequencing.

RNNs [11] are models that process both input and output as sequences of words. A characteristic of RNNs is to pass the result of the activation function from a hidden layer node to the output layer and then feed it back to the hidden layer node as input for subsequent calculations. However, RNNs are limited to relatively short sequences because their output depends on the results of previous calculations. That is why Sutskever et al. [12] chose LSTM [13], which adds an input gate, a forget gate, and an output gate to the memory cells of the hidden layer to remember only necessary information. LSTM predicts the conditional probability p for the input sequence $[x_1, \ldots, x_T]$ and the output sequence $[y_1, \ldots, y_{T'}]$. Using T or T' indicates that the input sequence length can differ from the output sequence length. When the final hidden state for the input is a fixed-dimensional context vector v, the probability of $[y_1, \ldots, y_{T'}]$ is

$$p(y_1, \ldots, y_{T'} | x_1, \ldots, x_T) = \prod_{i=1}^{T'} p(y_t | v, y_1, \ldots, y_{t-1}) \tag{3}$$

Sutskever et al. [12] proposed a model consisting of two modules: an encoder and a decoder. The encoder receives all words of the input sentence sequentially, and finally compresses all word information into one vector, which is called the context vector. The decoder receives the context vector and sequentially outputs the transformed words one by one. But the encoder-decoder structure can create a bottleneck that occurs when the encoder generates the entire sentence as a single fixed-length vector. The performance of the encoder-decoder model deteriorates as the input sentence length increases because it becomes challenging to represent the entire sentence with a single fixed-length vector.

Bahdanau et al. [14] proposed a method to improve performance by adding a new structure to the encoder-decoder model. Each time the model generates a result from the decoder, it sequentially searches the input sentence and applies the area most related to the part currently being generated. Finally, the next word is generated based on the areas judged to be highly relevant from the context words generated by the encoder, as well as the results already generated by the decoder. One of the biggest advantages of the method is that the input sentence does not have to be expressed as a vector of fixed length. Since the context vector generated by

the encoder is continuously referenced during the decoder's operation, there is no need to contain the information of the entire sentence in one vector. This allows performance to be maintained even as the sentence length increases.

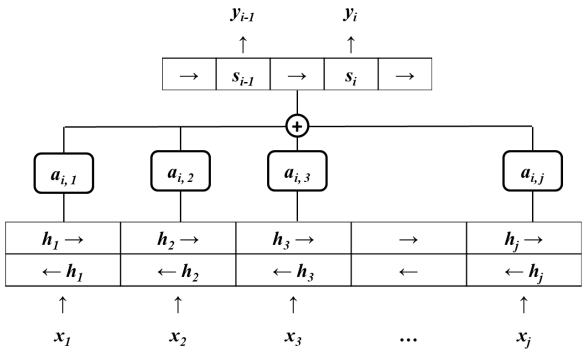

FIGURE 4. The structure for sequence to sequence using attention.

Fig. 4 shows that $x = [x_1, \ldots, x_j]$ is provided as the input to the encoder. A bidirectional RNN consists of two RNNs (a forward RNN and a backward RNN). The forward RNN reads the input sequentially from the beginning to generate a hidden state. The backward RNN reads the input backwards from the end of the input to generate a hidden state in the reverse direction. For the input x_j at the time j_{th}, the hidden state h_j is created by concatenating the forward and backward hidden states:

$$h_j = [\vec{h}_j^{\top}, \overleftarrow{h}_j^{\top}]^{\top} \tag{4}$$

The correlation between the encoder and decoder can be calculated as the score e_{ij}, which can be obtained from the hidden state s_{i-1} of the decoder at the $i\text{-}1_{th}$ time and the hidden state h_j of the encoder at the j_{th} time, through the feed forward neural network w:

$$e_{ij} = \mathrm{w}(s_{i-1}, h_j) \tag{5}$$

The corresponding attention weight a_{ij}, indicates how related the target word y_i is to the source word x_j. Finally, the context vector c_i is computed as the weighted sum after performing a dot product with the hidden state h_j:

$$a_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ij})} \tag{6}$$

$$c_i = \sum_{j=1}^{T_x} a_{ij} h_j \tag{7}$$

Neural network language models calculate the likelihood of a given word sequence and predict which word is most likely to follow in a given context, thereby generating more natural-sounding words, phrases, or sentences.

III. RELATED WORK

Various studies have been conducted on how to detect malicious injections on the CAN bus. Recent studies have introduced methods that use deep learning techniques.

In studies using LSTM, Hossain et al. [15] employed a vanilla LSTM to learn temporal features from multivariate time series CAN traffic. They investigated changes in hyperparameter values and suggested optimal parameters for an LSTM-based supervised classification model for CAN bus attack detection. The LSTM classifier was trained with one normal state (benign) and three attack states (DoS, fuzzy, and spoofing). In order to minimize a loss function, it was necessary to compare the target output values with the predicted output values. The architecture by Desta et al. [16] consisted of three dense layers and two LSTM layers. In a CAN ID sequence listed in chronological order, a set of 20 IDs was input into the layers, and a set of 20 predicted IDs was output. This architecture trained the layers by comparing the 20 predicted IDs with the 20 ground truth IDs. If abnormal messages were injected into the input, the predicted IDs would differ from the ground truth IDs. However, LSTM models are prone to overfitting when training data is insufficient. Training a deep LSTM model on a large dataset can be time-consuming, and finding suitable hyperparameters is difficult.

Models that combine LSTM and CNN were also introduced. Sun et al. [17] proposed a CNN combined with LSTM and an attention model, using the bit flip rate to preprocess input data. This approach analyzed continuous CAN message sequences and extracted continuous signal boundaries without background from the CAN communication protocol. To test the computing environment of ECUs, additional equipment was installed on a real vehicle to measure the detection time. The results showed that the detection time of the model on a real vehicle is not significantly different from the detection time on a PC. The bit flip rate algorithm is needed to calculate the total number of bit flips, which means that additional data preprocessing is required beyond converting the data to an appropriate type. Lo et al. [18] used a CNN-LSTM model to extract spatial and temporal features automatically from in-vehicle network traffic. The spatial features represented network traffic in the form of traffic images, while the temporal features represented the dependency of these features over time. The combination of spatial and temporal features has great potential to characterize anomalous vehicle network traffic. However, it only uses supervised learning for developing the CNN-LSTM model.

Inception-ResNet is used to perform classification tasks on the ImageNet [19] dataset and has demonstrated high performance in various computer vision tasks. Desta et al. [20] generated images using CAN IDs to train a model based on Inception-ResNet. During the image generation, unique patterns emerged in the images produced by different types of attacks. This indicated that certain patterns appeared in the CAN ID flow, and the CAN ID flow in a normal state differed from that in an attack state on a CAN bus. To implement real-time IDS, the online prediction was presented. This involved collecting CAN packets from real CAN networks using CAN transceivers and preprocessing them to fit the model's input. Song et al. [21] proposed a CNN-based model using the Inception-ResNet structure, showing good performance in detecting injection attacks. The original Inception-ResNet

model was too large to fit into a vehicle's IDS. A modified model with reduced size was used. Batch size was considered to minimize detection delays caused by data collection duration and processing time. As a result of testing GPU and CPU separately, when using GPU, 5,800 CAN messages were processed per second, and when using CPU, 4,300 CAN messages were processed per second. As the CAN bus of the test vehicle delivered approximately 2,000 messages per second, the model demonstrated sufficient capacity in real-time detection. However, the models have a fundamental limitation in detecting unlearned types of attack because it is based on supervised learning. The main difference between supervised and unsupervised learning is the requirement for labeled data. Supervised learning relies on labeled data, whereas unsupervised learning processes data without labels.

Generative adversarial network (GAN) is a deep learning architecture that trains two neural networks to compete against each other: one generates new data, and the other discriminates which outputs have been artificially created. Seo et al. [22] proposed training for known attack and unknown attack. The first discriminator received both normal and abnormal CAN images to detect known attacks. The second discriminator received normal CAN images and fake images generated by the generator to detect unknown attacks. To reduce detection time for real-time IDS, CAN IDs were converted into images using one-hot encoding, which were then used as input data for the model. However, the fake images are initially generated from random noise. It is necessary to verify whether generating abnormal messages using random noise results in feasible attacks in an actual vehicle environment.

An autoencoder is a neural network that compresses the input data by reducing its dimensionality and then reconstructs the data to produce an output as close as possible to the original input. To reduce the amount of labeled data, Hoang and Kim [23] used a convolutional adversarial autoencoder (CAAE) for semi-supervised learning, combining a small amount of labeled data with a large amount of unlabeled data during training. The encoder generated two latent variables, which were then forwarded to two discriminators: one portion was directed to a categorical distribution, while the other portion was directed to a Gaussian distribution. This approach was used because the patterns of both normal and abnormal data were so complex that a single discriminator was insufficient to handle them. Although the model was trained with the parameters of an encoder, a decoder, and two discriminators, only the weights of the encoder were used in real-time IDS to reduce inference time. Wei et al. [24] used a denoising autoencoder to encode and decode the data field in the messages. First, the data field's format was converted from hexadecimal to binary. Next, the data was fed into an autoencoder model containing an attention mechanism layer. This model determined whether the message was normal or abnormal based on certain patterns. The binary data were encoded into a concise form and then decoded to ensure they closely resembled the original message. Any errors in the reconstruction could indicate a potentially anomalous message. The attention mechanism allowed the model to assign importance to different aspects of the input data and to automatically learn the relative significance of each feature. However, the autoencoder may lose information during the encoding and decoding process due to the lossy compression of the input data. This loss can result in poor reconstruction quality, especially for complex or high-dimensional data.

To address the problem of lossy compression in the encoding-decoding process, recent research has proposed IDS solutions using the transformer. Nguyen et al. [25] used a transformer-based attention network to identify intrusive messages in CAN segments, which were groups of CAN IDs organized based on a window size. Each CAN ID segment needed to be labeled as either a normal message or an attack message. They also demonstrated that transfer learning can reuse information trained on a large dataset to improve a target model trained on a smaller dataset. They presented that the CAN bus can transmit up to 1,908 CAN frames per second at a recommended signaling rate of 250 kb/s (with a maximum capacity of 1 Mb/s), according to the Society of Automotive Engineers (SAE) J1939 standards. Their model could process 10,000 frames per second by measuring inference time in milliseconds per batch. Nam et al. [26] combined two generative pretrained transformer (GPT) networks in a bidirectional manner to enhance attack detection, compared to using a single-direction GPT network. The input data were processed more flexibly by adding a hidden layer that transfers information in the reverse direction. Once the model was trained to minimize the negative log-likelihood (NLL) loss function using normal CAN IDs, an NLL value exceeding a threshold indicated an attack. Alkhatib et al. [27] proposed an IDS based on Bidirectional Encoder Representations from Transformers (BERT). Unlike autoregressive models, they randomly masked some CAN IDs in the input sequence, enabling a bidirectional understanding of the sequence. It aimed to predict the masked CAN ID token based on its left and right tokens by unsupervised learning. The model size and the inference time per sample were measured to check whether it was suitable for real-time IDS. Due to the large number of model parameters, a high-performance ECU or a cloud server wirelessly connected to the vehicle was required. However, models that use only temporal features, such as CAN IDs, may not be able to detect attacks on spatial features, such as payload. Cobilean et al. [28] employed a method using CAN message sequences. A CAN ID and its corresponding payload were combined into a set, and multiple sets were linked in series to create an input sequence for training the transformer-based model. Although the optimal sequence length was suggested to be approximately 11-15 messages (99-135 tokens), using a shorter context led to a higher false positive rate. Table 2 presents several comparisons of the related studies.

IV. PROPOSED METHOD USING TRANSFORMER

In this section, we present an IDS using the transformer to utilize both the temporal data (CAN IDs) and the spatial data (payload).

A. STRUCTURE OVERVIEW

Fig. 5 shows the overview of proposed architecture. The model uses two sequences separately: CAN ID and payload sequences. The CAN ID sequence consists of a series of CAN IDs, while the payload sequence includes 8 bytes of payload along with the CAN ID. The CAN ID in the last slot of the CAN ID sequence is the same as the CAN ID in the last slot of the payload sequence. Therefore, the last slots of the both input sequences are filled with the same CAN ID. The CAN ID sequences, with the last token replaced by a mask token, and the payload sequences, with the last token removed, pass through the model separately. A final vector is calculated by adding the two CAN ID vectors from the last slots of the

TABLE 2. Comparison of related studies.

Related work	Main method	Dataset	Pre-proccess	Number of attack types	Feature	Learning method	Performance measures	Contribution	Limitation
[15]	LSTM	Shared	Hex to dec	3	CAN ID & payload	Supervised	FPR, FNR, Recall, F1	Select the best hyper-parameter values	Vulnerable to unknown attacks
[16]	LSTM	Own	One hot encoding	3	CAN ID	Unsupervised	Precision, Recall, F1	Unsupervised learning	Vulnerable to unknown attacks
[17]	CNN-LSTM & Attention	Shared	64 bit flip	5	Payload	Unsupervised	Error rate, Precision, Recall, F1	Unsupervised learning	Relying on the continuous signal extracted by the bit flip rate method
[18]	CNN-LSTM	Shared	Hex to dec	4	CAN ID & payload	Supervised	FNR, Precision, Recall, F1	Extracting spatial and temporal features	Vulnerable to unknown attacks
[20]	Inception-ResNet	Shared & own	Hex to dec	4	CAN ID	Supervised	TPR, FPR	Data visualization	Unable to detect attacks on payload sequence
[21]	Inception-ResNet	Shared	Hex to bit	4	CAN ID	Supervised	Error rate, Precision, Recall, F1	Good performance in error rate	Low performance in others
[22]	GAN	Shared	One hot encoding	4	CAN ID	Supervised	Detection rate, Accuracy	Requiring small amount of labeled data	Unable to check individual CAN ID
[23]	Autoencoder	Shared	Hex to bit	4	CAN ID	Semi supervised	Error rate, Recall, Precision, F1	Reducing the effort required for labeling	Vulnerable to unknown attacks
[24]	Autoencoder & Attention	Shared	Hex to bit	1	Payload	Unsupervised	Precision, Recall	Low computing power	Low performance in the case of small changes in CAN messages
[25]	Transformer	Shared	Hex to dec	4	CAN ID	Supervised	Precision, Recall, F1, Error rate	Transfer learning	Vulnerable to unknown attacks
[26]	Bi-GPT	Own	Hex to dec	4	CAN ID	Unsupervised	TPR, FPR, F1	Improved performance compared to uni-GPT	Unable to detect attacks on payload sequence
[27]	BERT	Shared	Hex to dec	3	CAN ID	Unsupervised	Precision, Recall, F1	Detect known and unprecedented attacks	Unable to detect attacks on payload sequence
[28]	Transformer	Shared	Hex to dec	3	CAN ID & payload	Unsupervised	F1	Considering both CAN ID and payload values	Requiring long input sequence

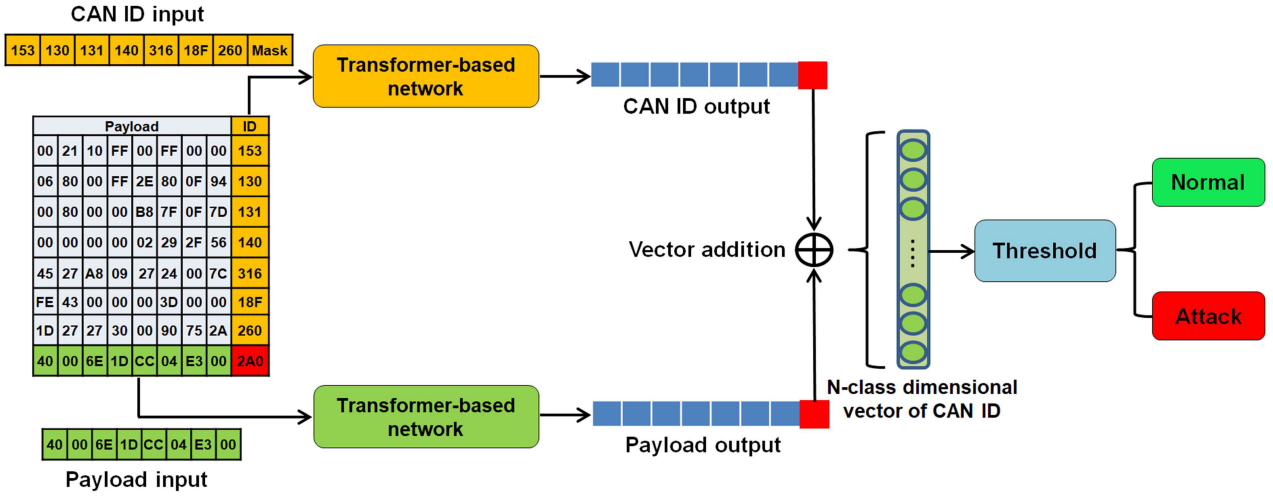

FIGURE 5. Overview of proposed architecture.

CAN ID sequence output and the payload sequence output. An attack is determined by comparing the ground truth CAN ID with the top n scores in the final vector.

During training, normal CAN data passes through the model to learn patterns in the normal data. During evaluation, both normal and abnormal CAN data pass through the model. When a normal CAN message passes through the model, the predicted CAN ID matches the ground truth CAN ID, following the learned pattern. However, when an abnormal (attacked) CAN message passes through the model, the predicted CAN ID does not match the ground truth CAN ID, as it deviates from the learned pattern.

B. DATASET

The dataset [29] was shared by the Hacking and Countermeasures Research Lab in Korea. It was extracted from Hyundai Sonata and included a normal dataset as well as attack datasets for flooding, fuzzy, and malfunction attacks. Table 3 shows the number of frames recorded in each dataset.

This dataset was suitable for evaluating the proposed model and other comparison models, as both CAN ID and payload sequences were manipulated in the attack dataset.

TABLE 3. In vehicle network intrusion detection dataset.

Vehicle	Dataset	Normal records	Abnormal records
Sonata	Normal	117,173	-
	Flooding	78,906	17,093
	Fuzzy	78,904	9,095
	Malfunction	78,797	8,202

The normal dataset contains only normal recording data. The other datasets contain both normal and abnormal recording data. A flooding dataset involves CAN messages with the most dominant CAN ID (0000) and payload filled with 0. A fuzzy dataset involves both CAN ID and payload being randomly generated, whereas a malfunction dataset involves valid CAN IDs with randomly manipulated payload. The normal dataset was used for training, while the flooding, fuzzy, and malfunction datasets were used for evaluating.

C. PREPROCESSING

CAN ID and payload in the CAN message are crucial for IDS, as typical attacks on a CAN bus, such as DoS, fuzzy, and replay attacks, aim to manipulate both the CAN ID and the payload. Thus, both sides are preprocessed to extract features.

A CAN ID consists of 11 bits in the ID field of the data frame, allowing for up to 2,048 CAN IDs ranging from 0×000 to $0 \times 7FF$ in hexadecimal. The data field contains a 64-bit payload, which is divided into 8 bytes. Each single digit is represented by 8 bits, ranging from 0×00 to $0\times FF$ in hexadecimal. Hexadecimal format is converted to decimal because the data are entered into sequence models as tensors,

Payload	CAN ID
00 40 60 ff 5a 6c 08 00	043f
00 20 00 00 00 00 00 00	0370
ff 00 00 00 ff 6c 08 00	0440
14 00 00 00 00 00 00 00	02c0
⋮	0153
05 24 84 09 24 23 00 7d	0316
fe 36 00 00 00 3c 00 00	018f
1b 24 24 30 ff 90 6e 06	0260
06 80 00 ff 13 80 01 9f	0130

FIGURE 6. Sliding window dividing CAN IDs and payload.

a process known as tokenizing. However the values of CAN IDs and the values of the payload in the data field become indistinguishable, because the range of CAN ID values is from 0×000 to $0 \times 7FF$ (0 to 2047 decimal), while the range of payload values is from 0×00 to $0\times FF$ (0 to 255 decimal). So the range of values from 0 to 255 is overlapped. To address this issue, 256 is added to all CAN IDs. This shifts the range of CAN ID values from $0 \sim 2047$ to $256 \sim 2303$ in decimal. Additionally, the data length code represents the byte length of the data field. For example, a DLC of 1 indicates that only 1 byte (one slot) is filled in the data field, while a DLC can be filled with up to 8 bytes (eight slot). Therefore, all blank slots in the data field are filled with a void token <2304> as shown in Table 4, when DLC is between 1 and 7.

Fig. 6 shows a sliding window that segments the sequence of CAN IDs by moving one slot at a time. The number of CAN IDs in each segment varies depending on the size of the sliding window.

Suppose T_n is the n_{th} set of CAN IDs $[t_{n-w+1}, \ldots, t_n]$, where w is the window size of T_n and S_n is the n_{th} segment of the payload $[s_n^0, \ldots, s_n^7]$ with t_n being the CAN ID corresponding to that payload.

$$S_n = [s_n^0, s_n^1, \ldots, s_n^7, t_n] \tag{8}$$

$$T_n = [t_{n-w+1}, t_{n-w+2}, \ldots, t_{n-1}, t_n] \tag{9}$$

The CAN ID t_n serves as a bridge connecting the CAN payload sequence and the CAN ID sequence.

TABLE 4. Index allocation.

Index	Token
$0 \sim 255$	Data
$256 \sim 2303$	CAN ID
2304	Void
2305	Mask

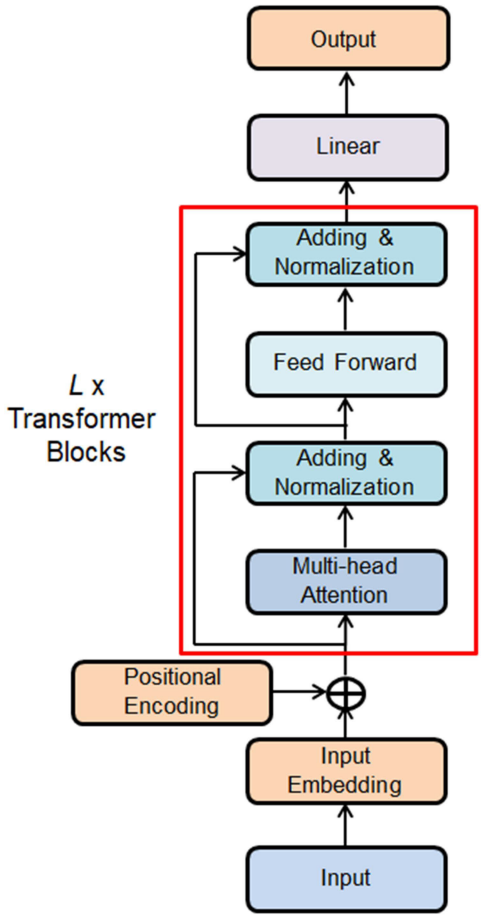

FIGURE 7. Structure of the transformer-based network.

D. TRANSFORMER

Fig. 7 shows the structure of the transformer-based network, which consists of an embedding layer, a positional embedding layer, L transformer blocks, and a linear layer. In particular, each transformer block consists of a multi-head self-attention layer, a layer normalization layer, a feed-forward layer, and a second layer normalization layer. The preprocessed CAN ID sequence T_n and payload sequence S_n are input to the network separately. The detailed process is outlined below.

First, through the embedding layer and positional embedding layer, each input token x is transformed into E-dimensional word embedding vectors X with initial random values. E-dimensional positional vectors Y are then added to incorporate the relative position information of tokens in the sequence. The learnable parameters [30] are used to provide the model with flexibility.

$$[X_0, \ldots, X_{E-1}] = \text{Embedding}[x_0, \ldots, x_{E-1}] \tag{10}$$

$$[Y_0, \ldots, Y_{E-1}] = \text{Embedding}[y_0, \ldots, y_{E-1}] \tag{11}$$

$$X_E = [X_0, \ldots, X_{E-1}] + [Y_0, \ldots, Y_{E-1}] \tag{12}$$

Second, X_E passes through the multi-head self-attention layer, where the queries Q, keys K, and values V for each sequence are obtained by applying linear layers to X_E.

$$\text{Q} = \text{linear}_\text{Q}(X_E),\ \text{K} = \text{linear}_\text{K}(X_E),\ \text{V} = \text{linear}_\text{V}(X_E) \tag{13}$$

The correlation between a query and a key can be expressed as the matrix multiplication of the query and the key. Dividing by $\sqrt{\text{d}_\text{k}}$ is a scaling factor, where d_k is the dimension of the key. Each specific part of the query is then evaluated as a ratio relative to all parts of the key using the softmax function. Finally, the output is computed by performing matrix multiplication with the value.

In the case of the payload sequence, a mask matrix of dimensions $i \times j$ is added to the attention score matrix of the same dimensions to prevent considering future tokens at the present time.

$$\text{M}_{i,j} = \begin{cases} 0 & if \quad j \le i \\ -\infty & if \quad j > i \end{cases} \tag{14}$$

$$\text{Attention}\,(\text{Q, K, V}) = \text{softmax}\left(\frac{\text{QK}^\text{T}}{\sqrt{\text{d}_\text{k}}} + \text{M}\right)\text{V} \tag{15}$$

The E-dimensional vectors of Q, K, and V are divided into n heads to form multi-head attention.

$$\begin{aligned} \text{MultiHead}\,(\text{Q, K, V}) &= \text{Concat}\,(\text{head}_1, \ldots, \text{head}_n)\,\text{W}^\text{O} \\ \text{where head}_\text{i} &= \text{Attention}(\text{QW}_\text{i}^\text{Q}, \text{KW}_\text{i}^\text{K}, \text{VW}_\text{i}^\text{V}) \end{aligned} \tag{16}$$

Equation (16) shows the multi-head attention, where the attention value is divided by the number of heads and then concatenated. During this process, Q, K, and V are projected onto parameter matrices $\text{W}_\text{i}^\text{Q}$, $\text{W}_\text{i}^\text{K}$, and $\text{W}_\text{i}^\text{V}$ respectively. The concatenated heads are then projected onto a parameter matrix W^O. This multi-head attention approach captures diverse types of dependencies, enhancing the model to understand the relationships between tokens.

Third, the residual input is added to the output of the attention mechanism, followed by layer normalization [31].

$$X_{\text{Norm}}^{(1)} = \text{Layernorm}(X_{\text{Attention}} + X_E) \tag{17}$$

This residual connection helps stabilize training and allows the model to learn identity mappings [32], while normalization ensures consistent distribution of the outputs across layers.

Fourth, the output from the previous step passes through a feed-forward network (FFN). This FFN comprises two linear layers W with a Rectified Linear Unit (ReLU) activation function [33] in between. The FFN helps to enhance the compatibility and expressiveness of the features generated by the attention mechanism from each head.

$$X_{\text{FFN}} = \text{W}_2\text{ReLU}\left(X_{\text{Norm}}^{(1)}\text{W}_1\right) \tag{18}$$

Fifth, the output of the FFN is combined with the residual input through addition, followed by layer normalization. This process generates the final output of the transformer block. The architecture of the transformer model may consist of

multiple blocks stacked together to capture more complex relationships in the data.

$$X^{(2)}_{\text{Norm}} = \text{Layernorm}(X_{\text{FFN}} + X^{(1)}_{\text{Norm}}) \tag{19}$$

Finally, the output of the transformer passes through a linear layer that converts the E-dimensional vector into an N-dimensional vector, where N represents the number of classes for CAN IDs. This final layer maps the transformer's output to the target space for classification.

E. DETECTION METHOD

During the training of the CAN ID sequence, the CAN ID located at the last slot in the input sequence is replaced with the mask token <2305> shown in Table 4 to prevent disclosing the target CAN ID.

$$T_{n(\text{input})} = [t_{n-w+1}, t_{n-w+2}, \ldots, t_{n-1}, < \text{Mask} >] \tag{20}$$

The output of the CAN ID sequence is compared to the target of the CAN ID sequence.

$$T_{n(\text{target})} = [t_{n-w+1}, t_{n-w+2}, \ldots, t_{n-1}, t_n] \tag{21}$$

The previous tokens $[t_{n-w+1}, t_{n-w+2}, \ldots, t_{n-1}]$ in the input sequence are utilized to transform the mask token <Mask> in the input sequence into the target token t_n in the target sequence.

During the training of the payload sequence, the CAN ID in the input sequence is removed to prevent disclosing the target CAN ID.

$$S_{n(\text{input})} = [s^0_n, s^1_n, \ldots, s^7_n] \tag{22}$$

The output of the payload sequence is compared with the target of payload sequence. The first CAN ID is removed in the target to make the input sequence and the target sequence the same length.

$$S_{n(\text{target})} = [s^1_n, \ldots, s^7_n, t_n] \tag{23}$$

The previous tokens $[s^0_n, s^1_n, \ldots, s^7_n]$ in the input sequence are used to help generate the target token t_n in the target sequence, that has been removed from the input sequence.

Each predicted CAN ID vector is obtained from the last slot of the CAN ID output sequence and the payload output sequence. The two predicted CAN ID vectors, each with N-dimensional space, are combined using vector addition.

$$t_n = t_{n(\text{CAN ID})} + t_{n(\text{payload})} \tag{24}$$

Finally, the index with the highest value among the N-dimensional vector t_n is selected as the predicted CAN ID. If the predicted CAN ID matches the actual CAN ID, it indicates a normal state. Conversely, if the predicted CAN ID differs from the actual CAN ID, it signifies an abnormal state. Additionally, the predicted CAN ID range can be extended from the highest value to include up to the n_{th} threshold.

V. EXPERIMENT

A. EXPERIMENTAL SETUP

The proposed model was implemented in Python using the PyTorch. The hardware setup was an Intel(R) i7-8700 CPU, 16 GB RAM, and an NVIDIA GTX 1660 GPU.

B. EVALUATION

A true positive (TP) is a case where the predicted CAN ID does not match the actual CAN ID for an abnormal CAN frame. A true negative (TN) is a case where the predicted CAN ID is the same as the actual CAN ID for a normal CAN frame. A false positive (FP) is a case where the predicted CAN ID does not match the actual CAN ID for a normal CAN frame. A false negative (FN) is a case where the predicted CAN ID is the same as the actual CAN ID for an abnormal CAN frame.

Precision, recall, and F-measure were used to evaluate the models:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{25}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{26}$$

$$\text{F} - \text{measure(F1)} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{27}$$

Furthermore, the receiver operating characteristic (ROC) curve was used to visualize the evaluation results. The ROC presents model's performance across various thresholds, consisting of true positive rate (TPR) and false positive rate (FPR).

C. HYPERPARAMETERS

Hyperparameters are tuned to balance the trade-off between model performance and computational resources. The goal is to determine the optimal parameter values through extensive testing based on the actual data used. All hyperparameters for this paper are listed in Table 5.

TABLE 5. Hyperparameter settings.

Hyperparameter	Setting
Loss function	Cross entropy
Number of heads	4
Transformer blocks	4
Embedding dimension	64
Feed forward dimension	64
Class dimension	2306
Batch size	1024

D. PERFORMANCE OF THE PROPOSED METHOD

We analyzed the performance changes of the proposed model depending on the window sizes. For the payload sequence, it consisted of a total of 9 tokens: 8 tokens for the payload and 1 token for the CAN ID. For the CAN ID sequences, the experiment was conducted by changing the window size of the sequence in three steps: 16, 32, and 64.

TABLE 6. Performance on the proposed method with window sizes.

	Window size	Precision	Recall	F1
Flooding	16	0.99650	1.0	0.99824
	32	0.99366	1.0	0.99682
	64	0.98791	1.0	0.99392
Fuzzy	16	0.56541	0.99384	0.72076
	32	0.56461	0.99406	0.72018
	64	0.73671	0.99395	0.84621
Malfunction	16	0.69603	0.56339	0.62273
	32	0.74818	0.89002	0.81296
	64	0.80524	0.96281	0.87700

Table 6 shows that during the flooding attack, the F1 score was high even though the window size was 16. During the flooding and fuzzy attacks, recall was high, but precision was low. During the malfunction attack, the increase in precision is smaller than the increase in recall as the window size increased. Fig. 8 shows that ROC performance improved depending on the window size during the malfunction attack. As a result, changing the window size affected model performance. However, the processing times also increased.

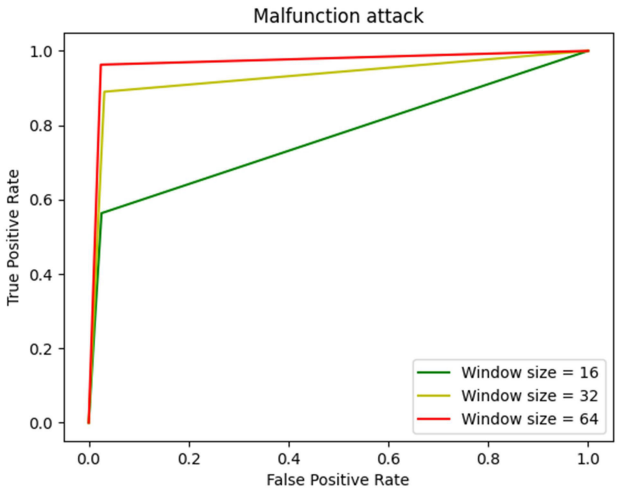

FIGURE 8. Performance on the proposed method against the malfunction attack.

E. COMPARISONS

The proposed model was compared with other four state-of-the-art models using the transformer mechanism. To make a fair comparison, we compared the predicted CAN ID from the core transformer network of each model with the actual CAN ID. The final predicted CAN ID was set as the index with the highest score in the output dimension. If the predicted CAN ID was the same as the actual CAN ID, it indicated a normal state. If it was different, it indicated an abnormal state.

CAN-BERT [27] is a transformer encoder-based model. In this model, CAN IDs in the CAN ID sequence were randomly replaced with a mask token, and the model was trained to predict the masked CAN IDs.

Uni-directional GPT [34] is a transformer decoder-based model. In this model, the last CAN ID in the CAN ID sequence was removed, and the model was trained to predict the removed CAN ID.

Bi-directional GPT [26] is a transformer decoder-based model that operates in both directions. The outputs from the forward GPT network and the backward GPT network were concatenated. It was trained in the same manner as the Uni-directional GPT method.

The three models [26], [27], [34] used the CAN ID sequence as the input and tried to analyze temporal data.

CAN-Former IDS [28] is a transformer decoder-based model. A CAN message consisted of 8 tokens for the payload and 1 token for the CAN ID. An input sequence was formed by arranging several CAN messages in order. The last CAN ID in the sequence was removed, and the model was trained to predict the removed CAN ID.

This model [28] used the CAN message sequence as the input and tried to analyze spatial data.

TABLE 7. Performance comparison with the window size of 16.

Attack	Model	Precision	Recall	F1
Flooding	Uni-GPT	0.37418	1.0	0.54458
	Bi-GPT	0.36775	1.0	0.53775
	CAN-BERT	0.41842	1.0	0.58998
	CAN-Former	0.17833	1.0	0.30268
	Proposed	0.99650	1.0	0.99824
Fuzzy	Uni-GPT	0.2678	0.99648	0.42215
	Bi-GPT	0.2435	0.95513	0.38806
	CAN-BERT	0.27252	0.99604	0.42795
	CAN-Former	0.1042	1.0	0.18873
	Proposed	0.56541	0.99384	0.72076
Malfunction	Uni-GPT	0.26634	0.96733	0.41768
	Bi-GPT	0.2435	0.95513	0.38806
	CAN-BERT	0.27416	0.99159	0.42956
	CAN-Former	0.09438	0.99817	0.17246
	Proposed	0.69603	0.56339	0.62273

TABLE 8. Performance comparison with the window size of 32.

Attack	Model	Precision	Recall	F1
Flooding	Uni-GPT	0.3876	1.0	0.55866
	Bi-GPT	0.3614	1.0	0.53092
	CAN-BERT	0.41133	1.0	0.5829
	CAN-Former	0.17844	1.0	0.30284
	Proposed	0.99366	1.0	0.99682
Fuzzy	Uni-GPT	0.29221	0.99648	0.45191
	Bi-GPT	0.26191	0.99659	0.4148
	CAN-BERT	0.29324	0.99703	0.45319
	CAN-Former	0.1037	1.0	0.18791
	Proposed	0.56461	0.99406	0.72018
Malfunction	Uni-GPT	0.27501	0.94684	0.42622
	Bi-GPT	0.25173	0.97342	0.40001
	CAN-BERT	0.28883	1.0	0.44821
	CAN-Former	0.09474	0.99976	0.17307
	Proposed	0.74818	0.89002	0.81296

Since the proposed model used a method of merging the CAN ID sequence and the payload sequence, CAN ID sequence of 16, 32, or 64 tokens depending on window size setting and the payload sequence of 9 tokens are input to the

TABLE 9. Performance comparison with the window size of 64.

Attack	Model	Precision	Recall	F1
Flooding	Uni-GPT	0.40878	1.0	0.58033
	Bi-GPT	0.36532	1.0	0.53515
	CAN-BERT	0.40115	1.0	0.5726
	CAN-Former	0.17806	1.0	0.30229
	Proposed	0.98791	1.0	0.99392
Fuzzy	Uni-GPT	0.32029	0.99626	0.48474
	Bi-GPT	0.252	0.99626	0.40225
	CAN-BERT	0.29959	0.99681	0.46071
	CAN-Former	0.10334	1.0	0.18732
	Proposed	0.73671	0.99395	0.84622
Malfunction	Uni-GPT	0.27711	0.81822	0.414
	Bi-GPT	0.238	0.985	0.38336
	CAN-BERT	0.28328	1.0	0.44149
	CAN-Former	0.09429	1.0	0.17233
	Proposed	0.80524	0.96281	0.87701

model separately. To ensure a fair comparison, the window size of the comparison models was adjusted to match that of the proposed model. For models [26], [27], [34] that used only the CAN ID sequence, the window sizes were set to 24, 40, or 72. For model [28] that used the CAN message sequence, the message sizes were set to 3 (27 tokens), 5 (45 tokens), and 8 (72 tokens).

Tables 7 - 9 show precision, recall, and F1 score records in the flooding, fuzzy, and malfunction attacks. All models showed good performance in the recall. However, there were significant variations in performance regarding precision. The proposed model demonstrated superior performance during flooding attack. It also outperformed other models in the other attacks. For other models, the precision during the flooding attack was slightly better than the precision during the other two attacks. In the fuzzy and malfunction attacks, unlike the flooding attack, payload was manipulated. For example, if the payload value was changed within a valid CAN ID, temporal analysis may show normal patterns with no abnormalities, but manipulation of spatial data may not be detected.

Fig. 9 and 10 show that there was no significant performance change as the window size increases in other models. In contrast, the performance of the proposed model gradually improved as the window size increases. For models that used only CAN ID sequences, a sufficiently large window size was required to detect patterns in the complex arrangement of CAN IDs. For CAN-Former that used CAN messages sequences, it was vulnerable to temporal analysis of CAN ID sequence, although spatial analysis of the payload sequence was possible. The model also required a sufficient window size to identify patterns in the complex arrangement of CAN message sequences. However, as the window size increased, more computing power was required and the time for training and inference became longer. Since the proposed model used CAN ID sequence and payload sequence simultaneously, it responded adaptively with shorter window sizes. Since similar payload sequences are recorded under an identical CAN ID, the payload sequence has more fixed patterns than the CAN ID sequence. Thus, the trained payload sequence contributes to predicting a CAN ID that matches the actual CAN ID with high probability. However, it is not possible to analyze temporal patterns, so the CAN ID sequence is used to supplement the temporal analysis. For example, detecting a flooding attack that injects multiple replicated CAN messages using only spatial analysis (payload sequence) is challenging. Therefore, temporal analysis (CAN ID sequence) is employed to identify disruptions in the time series data. The proposed model responded quickly to attacks manipulating both temporal and spatial data, and demonstrated the improved performance as the window size increased.

Fig. 11 shows that inference time increased as the window size increased. Inference time represents the average time required to process an entire frame for each attack dataset. For example, in the malfunction dataset shown in Table 3, when the recorded normal frames (78,797) and abnormal frames (8,202) are combined, the total number of frames is 86,999. When the window size was 64, the proposed model processed 86,999 frames in 6 seconds, resulting in an inference time of approximately 0.0689 milliseconds per frame. According to SAE J2284-3, the node-to-node bit rate is 500 kbps. Therefore, the average transmission time for one frame is approximately 0.26 milliseconds [35]. The proposed model is suitable for real-time detection in our experimental environment. Inference time can vary depending on factors such as hardware and software environments. Therefore, comparing inference times is most appropriate when models are evaluated under the same conditions. The proposed model required less time than the other models across all window sizes. The bi-directional model took more processing time than the uni-directional GPT, because it used the decoder in both directions.

Figs. 12 - 14 compare the ROC performance of the intrusion detection methods for flooding, fuzzy, and malfunction attacks with a window size of 64. The proposed model was more robust than other models against the flooding, fuzzy, and malfunction attacks. However, the performance in the fuzzy and malfunction attacks was not as good as in the flooding attack. To form the final vector, the last CAN ID vector of the CAN ID sequence and the CAN ID vector of the payload sequence were evenly combined and harmonized. Therefore, performance can be improved by increasing the window size, as the CAN ID sequence has lower regularity than the payload sequence.

Most models showed high recall values, as the false negative value was low. A high false-negative rate corresponds to a significant number of missed intrusions. However, the low precision values were the main factor affecting the F1 scores of the other models and this also applied to the proposed model. False positive values affect the precision and lead to false alarms.

F. EXPANSION OF THE PREDICTED CAN ID RANGE

One reason for the high false positive rate may be that, after an injection attack, even if the CAN ID and payload return

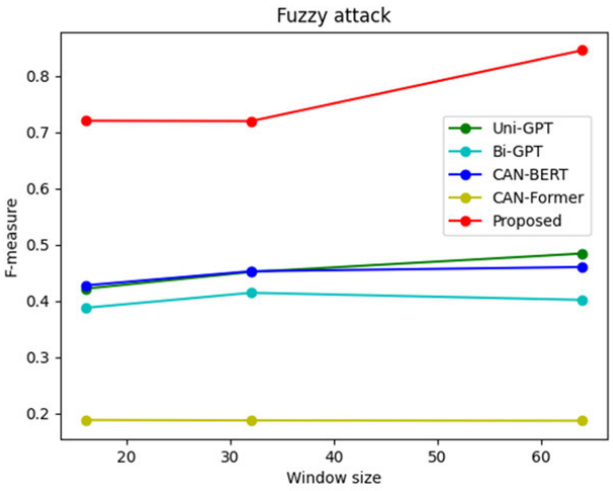

FIGURE 9. Performance comparison on the fuzzy attack.

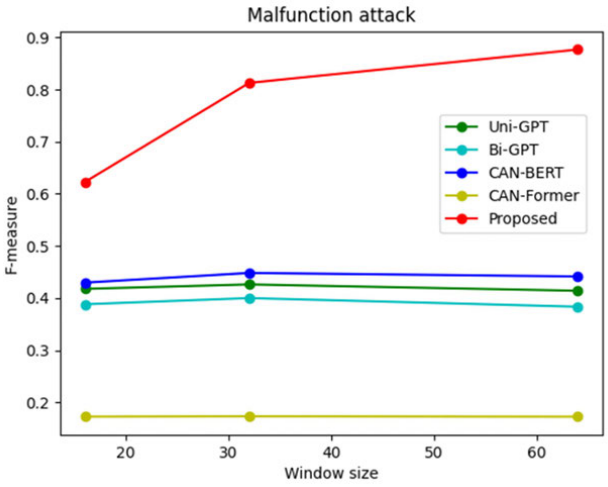

FIGURE 10. Performance comparison on the malfunction attack.

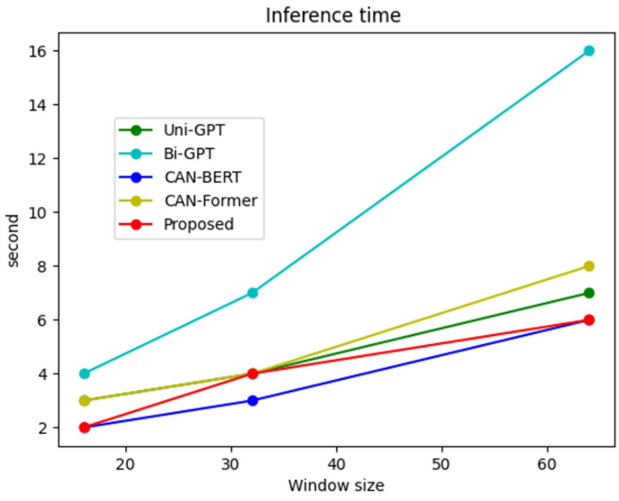

FIGURE 11. Comparison of average inference time per attack dataset.

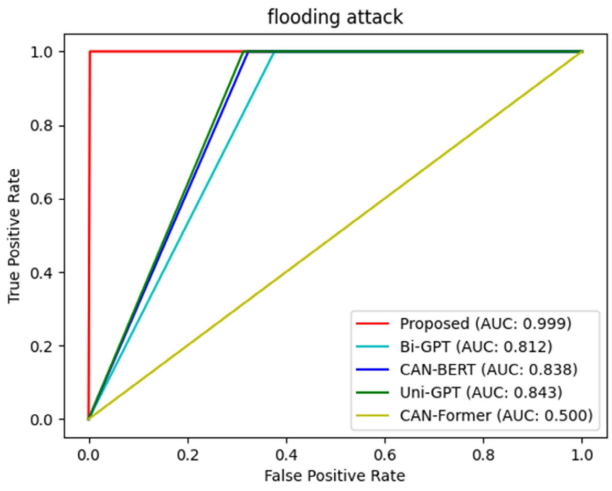

FIGURE 12. Performance comparison on the flooding attack with the window size of 64.

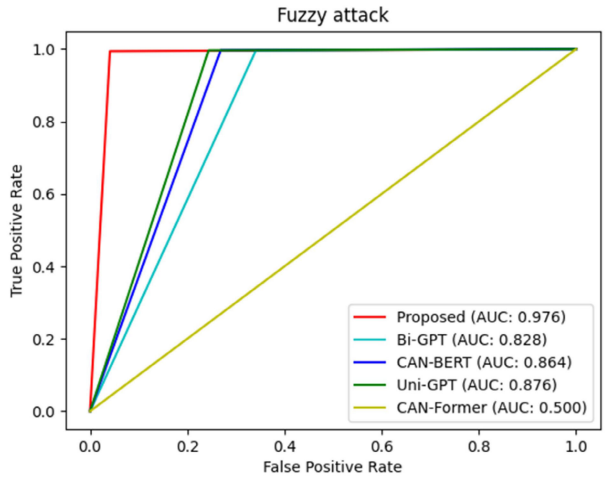

FIGURE 13. Performance comparison on the fuzzy attack with the window size of 64.

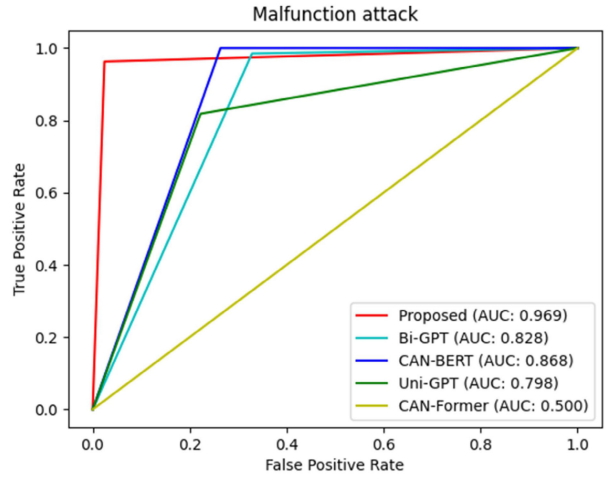

FIGURE 14. Performance comparison on the malfunction attack with the window size of 64.

to their normal state, the model might still fail to predict the correct CAN ID due to the influence of previously injected CAN IDs in the CAN ID sequence. To reduce the problem, the range of predicted CAN IDs was expanded.

TABLE 10. Performance comparison based on range adjustments with the window size of 128.

Attack	Range	Precision	Recall	F1
Flooding	0	0.96762	1.0	0.98354
	1	0.99801	1.0	0.99901
	2	0.99965	1.0	0.99982
	3	1.0	1.0	1.0
	4	1.0	1.0	1.0
	5	1.0	1.0	1.0
Fuzzy	0	0.58844	0.99406	0.73927
	1	0.81929	0.99318	0.89789
	2	0.88104	0.99285	0.93361
	3	0.91799	0.99219	0.95365
	4	0.93372	0.99153	0.96176
	5	0.94704	0.9912	0.96862
Malfunction	0	0.71241	1.0	0.83206
	1	0.82018	0.99988	0.90116
	2	0.85708	0.99951	0.92283
	3	0.88829	0.99854	0.94019
	4	0.90611	0.99659	0.9492
	5	0.92025	0.99183	0.9547

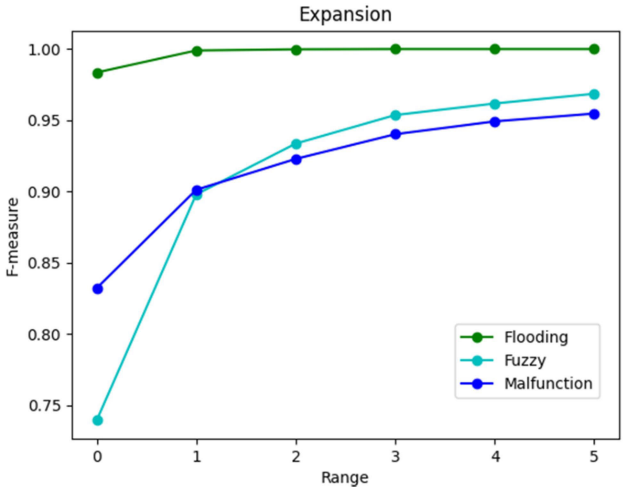

FIGURE 15. Expansion of the predicted CAN ID range.

The index of the highest value in the N-dimensional vector is selected as the predicted CAN ID and compared with the target index. If they match, it is regarded as a normal state. Otherwise, it is regarded as an abnormal state.

Table 10 shows the performance comparison based on range adjustments when the window size was set to 128. The initial state resulted in worse performance compared to using the window size of 64. However, when the output scores in the N-dimensional vector were sorted in descending order and the prediction range was expanded from the highest score to the n_{th} range, the confusion matrix changed. When the range was expanded, the recall value dropped only slightly, while precision increased dramatically.

Fig. 15 shows that the F1 score improved for the flooding, fuzzy, and malfunction attacks with the expanded range when the window size was set to 128. Expanding the range from the highest score to the top 5 scores significantly improved performance, demonstrating that the analysis of the CAN ID sequence and payload sequence was effective.

Expanding the range to an appropriate level can increase the probability of accurate prediction. However, if the range is expanded excessively, the concept of prediction may be compromised due to an increase in false negatives.

VI. CONCLUSION

The purpose of this paper is to identify the most suitable model for limited resources by applying transformer-based techniques to an intrusion detection system. To the best of our knowledge, existing IDS methods that use transformer networks require substantial computing power. In particular, processing long input sequences requires a lot of training time and computing power.

In the experiment, models using only temporal analysis of CAN ID sequences or only spatial analysis of CAN message sequences exhibited poor detection performance with short window sizes. The proposed model, which employs a transformer-based mechanism for both spatial and temporal data analysis, responds quickly with short window sizes and outperforms other state-of-the-art techniques using transformer networks. Using unsupervised learning, normal CAN data from general vehicles is able to be utilized directly without additional labeling and it is possible to prepare for both predictable and unpredictable attacks. Moreover, it is more robust against attacks by maximizing performance through range expansion.

However, the proposed approach uses the characteristics of probabilistic sequencing to predict whether the next CAN ID will be natural, based on the previously input CAN IDs within a given window size. If an abnormal CAN ID is injected among the previously input CAN IDs, the probability of predicting the correct CAN ID may decrease, even if the current CAN ID is normal. Therefore, future work will focus on developing methods to exclude abnormal CAN IDs and improve overall performance.

VII. DISCUSSION

A. PATTERN EXTRACTION

Since automobile manufacturers encrypt the information necessary for vehicle control and transmit it through a CAN bus, it is natural for the CAN data extracted from the bus to appear abstract. Therefore, the role of the language model is to identify patterns in the CAN data extracted from the bus without decryption [36], [37], [38], [39]. For example, if a vehicle is driven under the same conditions and without accidents, a normal dataset would be available for use. If an attack is injected into the bus, it can be distinguished by the patterns learned from normal operating conditions. This is why the pattern extraction in unsupervised learning can be effective against unexpected attacks.

The purpose of IDS research is to detect malicious attempts to manipulate CAN messages in unintended ways. Intended normal conditions can be used for training. However, it is impossible to anticipate and prepare for every attack scenario.

In conclusion, it is necessary to research how to train language models to distinguish between intended and unintended situations.

B. FEASIBILITY

Considering the unique characteristics of vehicles, a trade-off is needed between the computing power available in real-world vehicle conditions and the performance of the IDS model.

It is unrealistic to complete all pattern learning at the factory level because unlimited conditions, such as various driving environments, driving patterns, and operating conditions, need to be taken into consideration. Recursive training is necessary under numerous conditions because a vehicle's operating environment changes continuously throughout its life cycle.

In the case of software updates during periodic technical inspections, there are various constraints. For example, it requires a large storage capacity for CAN data, a controller for uploading vehicle information, and sufficient processing time for the acquired data. Therefore, on-board training may be the most realistic option.

REFERENCES

[1] *Cybersecurity and Cybersecurity Management System*, document 155, 2021.

[2] C. L. Schuba, I. V. Krsul, M. G. Kuhn, E. H. Spafford, A. Sundaram, and D. Zamboni, "Analysis of a denial of service attack on TCP," in *Proc. IEEE Symp. Secur. Privacy*, Jun. 1997, pp. 208–223.

[3] S. J. Sheather, "Density estimation," *Stat. Sci.*, vol. 1, pp. 588–597, Oct. 2004.

[4] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Schol-kopf, "Support vector machines," *IEEE Intelligent Systems applications*, vol. 13, no. 4, pp. 18–28, May 1998.

[5] M. Farsi, M. Barbosa, and K. Ratcliff, "An overview of controller area network," *Comput. Control Eng. J.*, vol. 10, no. 3, pp. 113–120, Jun. 1999.

[6] K. H. Johansson, M. Torngren, and L. Nielsen, "Vehicle applications of controller area network," in *Handbook Net-Worked Embedded Control Systems*. Cham, Switzerland: Springer, 2005, pp. 741–765.

[7] W. Choi, H. J. Jo, S. Woo, J. Y. Chun, J. Park, and D. H. Lee, "Identifying ECUs using inimitable characteristics of signals in controller area networks," *IEEE Trans. Veh. Technol.*, vol. 67, no. 6, pp. 4757–4770, Jun. 2018.

[8] C. Shin, "A framework for fragmenting/reconstituting data frame in controller area network (CAN)," in *Proc. 16th Int. Conf. Adv. Commun. Technol.*, Feb. 2014, pp. 1261–1264.

[9] R. Buttigieg, M. Farrugia, and C. Meli, "Security issues in controller area networks in automobiles," in *Proc. 18th Int. Conf. Sci. Techn. Autom. Control Comput. Eng. (STA)*, Dec. 2017, pp. 93–98.

[10] T. Kudo and J. Richardson, "SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing," 2018, *arXiv:1808.06226*.

[11] R. M. Schmidt, "Recurrent neural networks (RNNs): A gentle introduction and overview," 2019, *arXiv:1912.05911*.

[12] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 27, 2014, pp. 1–11.

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[14] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2014, *arXiv:1409.0473*.

[15] M. D. Hossain, H. Inoue, H. Ochiai, D. Fall, and Y. Kadobayashi, "LSTM-based intrusion detection system for in-vehicle can bus communications," *IEEE Access*, vol. 8, pp. 185489–185502, 2020.

[16] A. K. Desta, S. Ohira, I. Arai, and K. Fujikawa, "ID sequence analysis for intrusion detection in the CAN bus using long short term memory networks," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops*, Mar. 2020, pp. 1–6.

[17] H. Sun, M. Chen, J. Weng, Z. Liu, and G. Geng, "Anomaly detection for in-vehicle network using CNN-LSTM with attention mechanism," *IEEE Trans. Veh. Technol.*, vol. 70, no. 10, pp. 10880–10893, Oct. 2021.

[18] 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," *Veh. Commun.*, vol. 35, Jun. 2022, Art. no. 100471.

[19] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 248–255.

[20] 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," *Veh. Commun.*, vol. 35, Jun. 2022, Art. no. 100470.

[21] H. M. Song, J. Woo, and H. K. Kim, "In-vehicle network intrusion detection using deep convolutional neural network," *Veh. Commun.*, vol. 21, Jan. 2020, Art. no. 100198.

[22] E. Seo, H. M. Song, and H. K. Kim, "GIDS: GAN based intrusion detection system for in-vehicle network," in *Proc. 16th Annu. Conf. Privacy, Secur. Trust (PST)*, Aug. 2018, pp. 1–6.

[23] T.-N. Hoang and D. Kim, "Detecting in-vehicle intrusion via semi-supervised learning-based convolutional adversarial autoencoders," *Veh. Commun.*, vol. 38, Dec. 2022, Art. no. 100520.

[24] P. Wei, B. Wang, X. Dai, L. Li, and F. He, "A novel intrusion detection model for the CAN bus packet of in-vehicle network based on attention mechanism and autoencoder," *Digit. Commun. Netw.*, vol. 9, no. 1, pp. 14–21, Feb. 2023.

[25] T. P. Nguyen, H. Nam, and D. Kim, "Transformer-based attention network for in-vehicle intrusion detection," *IEEE Access*, vol. 11, pp. 55389–55403, 2023.

[26] M. Nam, S. Park, and D. S. Kim, "Intrusion detection method using bi-directional GPT for in-vehicle controller area net-works," *IEEE Access*, vol. 9, pp. 124931–124944, 2021.

[27] 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 *Proc. IEEE/ACS 19th Int. Conf. Comput. Syst. Appl. (AICCSA)*, Dec. 2022, pp. 1–8.

[28] V. Cobilean, H. S. Mavikumbure, C. S. Wickramasinghe, B. J. Varghese, T. Pennington, and M. Manic, "Anomaly detection for in-vehicle communication using transformers," in *Proc. 49th Annu. Conf. IEEE Ind. Electron. Soc.*, Oct. 2023, pp. 1–6.

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

[30] G. Zerveas, S. Jayaraman, D. Patel, A. Bhamidipaty, and C. Eickhoff, "A transformer-based framework for multivariate time series representation learning," in *Proc. 27th ACM SIGKDD Conf. Knowl. Discovery Data Mining*, Aug. 2021, pp. 2114–2124.

[31] J. Lei Ba, J. Ryan Kiros, and G. E. Hinton, "Layer normalization," 2016, *arXiv:1607.06450*.

[32] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 630–645.

[33] A. F. Agarap, "Deep learning using rectified linear units (ReLU)," 2018, *arXiv:1803.08375*.

[34] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," OpenAI, San Francisco, CA, USA, 2018.

[35] L. Popa, B. Groza, C. Jichici, and P.-S. Murvay, "ECUPrint—Physical fingerprinting electronic control units on CAN buses inside cars and SAE J1939 compliant vehicles," *IEEE Trans. Inf. Forensics Security*, vol. 17, pp. 1185–1200, 2022.

[36] S. Tariq, S. Lee, H. K. Kim, and S. S. Woo, "CAN-ADF: The controller area network attack detection framework," *Comput. Secur.*, vol. 94, Jul. 2020, Art. no. 101857.
[37] M. Bozdal, M. Samie, and I. K. Jennions, "WINDS: A wavelet-based intrusion detection system for controller area network (CAN)," *IEEE Access*, vol. 9, pp. 58621–58633, 2021.
[38] A. Buscemi, I. Turcanu, G. Castignani, A. Panchenko, T. Engel, and K. G. Shin, "A survey on controller area network reverse engineering," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 3, pp. 1445–1481, 2nd Quart., 2023.
[39] H. Olufowobi, U. Ezeobi, E. Muhati, G. Robinson, C. Young, J. Zambreno, and G. Bloom, "Anomaly detection approach using adaptive cumulative sum algorithm for controller area network," in *Proc. ACM Workshop Automot. Cybersecurity*, Mar. 2019, pp. 25–30.

HYUNJUN JO received the B.S. degree in electronic engineering from Incheon National University, Republic of Korea, in 2015, and the M.S. degree in electronic engineering from Inha University, Republic of Korea, in 2022, where he is currently pursuing the Ph.D. degree. Since 2016, he has been a Researcher with Korea Automobile Testing and Research Institute. His research interests include vehicle safety, cybersecurity, data analysis, and artificial intelligence.

DEOK-HWAN KIM (Member, IEEE) received the Ph.D. degree in computer science from Korea Advanced Institute of Science and Technology (KAIST), in 2003. He has been a Professor with Inha University, South Korea, since 2006. He has authored and/or presented more than 100 publications in major journals and international conferences. His current research interests include storage systems, intelligent cloud storage, embedded and real-time systems, and biomedical systems.