SCAN: Secure CAN Framework by using Deep Learning

1st Sumansmita Rout
Advance Technology Development Centre
IIT Kharagpur, West Bengal, India
rout.sumansmita@gmail.com

2nd Ayantika Chetterjee
Advance Technology Development Centre
IIT Kharagpur, West Bengal, India
cayantika@gmail.com

***Abstract*—With intelligent vehicle networking flourishing, adding more interfaces to interact with the outside world results in security vulnerabilities arising, which might have disastrous implications due to communication gaps. The Controller Area Network (CAN) is a standard communication between vehicle electronic control units (ECUs). Given the widespread usage of the CAN system and its inherent protocol flaws, exploiting attacks such as DoS, fuzzy, impersonation attacks, etc., is an unavoidable concern. Despite the availability of various authentication and encryption algorithms, this paper adopted deep learning models for its detection mechanism for new attacks, and is feasible to deploy on a low-cost resource-constrained platform. To achieve the objective of real-time detection, the proposed approach SCAN presents a lightweight attack detection model based on a Gated Recurrent Unit (GRU). The proposed SCAN is extensively experimented on the server and the edge device as well, using a publicly available dataset. It obtained an attack detection performance of 90.20% accuracy and 90% accuracy on Look-Up Table (LUT) substitution on the sigmoid activation. Altering the sigmoid activation function with LUT resulted in a 50% reduction in delay on the server and a 61.9% reduction on the edge.**

***Index Terms*—CAN, GRU, sigmoid function, LUT.**

I. Introduction

The automotive industry is transforming significantly by adopting advanced automation technologies, including a network of sensors and computing systems. These sensors, controlled by ECUs, efficiently manage various vehicle operations such as engine control and anti-lock braking [1]. These ECUs use internal communication, such as CAN, which is a serial-bus communication protocol that serves as the main channel for exchanging information between sensors, actuators, and ECUs through a robust message-based system [3]. While CAN provides practical advantages, its growing inter and intra-vehicle connections expose it to cyberattacks. The protocol lacks global synchronization mechanisms and built-in cryptographic features, leaving it vulnerable to unauthorized access and data injection attacks, compromising system integrity [1], [4], [7]. Additionally, its priority-based messaging can lead to availability issues when high-priority messages flood the network [1]. Attackers exploit these vulnerabilities with denial-of-service (DoS) assaults, spoofing attacks, fuzzy assaults targeting CAN IDs and data payloads, [3], [17] and impersonation attacks [6]. Data manipulation techniques like frame faking, insertion, and replay attacks further exacerbate security risks [3], [5]. Therefore, various authentication and encryption methods have been implemented to provide CAN security. [13] introduced a lightweight authentication solution for automotive networks. Other methods include Honeypot, MAC authentication, and CBC-MAC. However, these methods alone cannot fully mitigate severe threats like replay attacks and only safeguard a limited range of vehicle components, necessitating extensive modifications to all ECUs, which is impractical [7]. Furthermore, preliminary studies indicate that ECUs lack sufficient resources, such as power and processing capabilities, to run cryptographic algorithms for secure vehicle communication [9]–[12]. So, protection modules must be compatible with the limited powered ECUs for real-time scenarios. The only reported work addressing this issue, as documented in [8], utilizes advanced attribute-based access-control encryption with concealed policy and credentials. However, expanding the number of nodes requires reconfiguring the entire framework and managing new secret keys for each node, resulting in significant data transmission overhead. Hence, these approaches are unsuitable for CAN networks with large nodes as separate protections for each node pair, creating significant computation and communication overhead for addressing attacks like DoS, fuzzy, impersonation, and spoofing separately.

Therefore, we opted for deep learning strategy, which can comprehend intricate patterns in the dataset and detect multiple attacks more effectively [17] than traditional machine learning algorithms and statistical methods. Our detailed contributions to this paper are as follows:

- We developed SCAN, which focuses on end-to-end prediction for detecting attack-based frames specifically tailored for vehicle-mounted ECUs with limited storage and processing capabilities.
- SCAN employs a lightweight binary classification deep neural network (DNN) based on GRU, capable of detecting whether the data frame is benign or malicious.
- Further, SCAN utilizes the training setup to get optimized LUT parameters in order to substitute the sigmoid activation function to reduce computational requirements at the edge.
- We experimented SCAN using the publicly available CAN dataset to assess the accuracy and the time complexity of the proposed DNN when the dataset is balanced.

Additionally, we analyzed the precision, recall, and F1 score for the implementation of LUT parameters.

The subsequent sections are structured as follows: Section II presents a concise analysis of the existing literature on intrusion detection systems in CAN and optimization strategies for activation functions. Section III introduces a proposed framework that showcases the dataset, model details, and approaches for enhancing the activation layers in the proposed model. Section IV shows our experiments. Section V presents our results and observations. Section VI concludes this study and outlines future work.

II. Related Works

Researchers in CAN bus attack detection have focused on improving classification performance. This section reviews the literature on attack detection models and optimization strategies for expensive functions in deep learning models suitable for low-powered ECUs. Taylor et al. introduced an anomaly detection technique using Long Short-Term Memory (LSTM) to predict the next state message and compare it with the actual value, identifying anomalies based on a threshold. Experimental results showed that LSTM effectively classified sequential CAN traffic [15]. Tariq et al. developed an approach to detect DoS, replay, and fuzzy attacks in a real vehicle using heuristics and Recurrent Neural Networks (RNNs). They combined these methods in an ensemble to achieve the final detection outcome [14]. [17] developed CANintelliIDS, an intrusion detection system using convolutional attention and GRU to identify CAN bus attacks. [18] introduces CAN-ADF, a framework for attack identification. It combines heuristic/rule-based methods with LSTM models and ensembles the best results to detect attacks effectively. Nevertheless, these strategies render the solution cumbersome and may not ensure real-time performance when deployed on vehicle-mounted computing equipment. [16] proposed GRU as the hidden layer of the neural network model, with the sigmoid activation function employed in the output layer of the model. This study offers a robust architecture that is feasible to implement on an embedded platform. In this direction, we explore developing an end-to-end prediction on the edge for CAN attack detection. However, costly activation functions are employed in deep learning models to map outputs within a specific range and facilitate proper network training, but their implementation carries significant expenses. Therefore, approximation methods are required to implement activation functions in hardware. They can be classified into the following categories: direct computing, piecewise linear approximation (PWL), look-up table (LUT), and hybrid approaches. Implementing LUT approximation is relatively straightforward compared to other methods, facilitating easy modification of LUT contents for various nonlinear computations [21], [22]. However, the implementation of the linearization technique in the referenced study and the training setup are not well-detailed.

It is important to note that most of the current intrusion detection models are not explored for making real-time predictions on edge. Studies, such as [16], have successfully implemented basic models for intrusion detection and demonstrated their feasibility on edge devices like Jetson Nano, which are equipped with GPUs (Graphical Processing Units). Nevertheless, these devices are expensive. Our goal is to create a very lightweight model with minimal parameters and also can achieve adequate accuracy for attack detection that can be easily deployable on low-powered as well as low-cost edge devices for prediction. Additionally, we are driven to optimize computationally expensive functions used in our proposed model.

III. Proposed Framework

A. System Model

This paper introduces, as depicted in Fig. 1, the training of a lightweight GRU-based binary class attack detection model developed to distinguish between attack and non-attack frames. The focus is on demonstrating the end-to-end prediction of the quantized versions of the detection model. Additionally, it highlights the latency difference observed in the LUT-based approach for sigmoid activation, utilized on the output layer in the detection model.

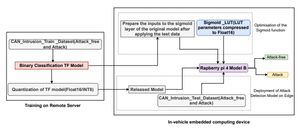
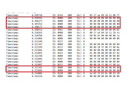

CAN_Intrusion_Train _Dataset(Attack_free and Attack)
Binary Classification TF Model
Quantization of TF model(Float16/INT8)
Training on Remote Server
Prepare the inputs to the sigmoid layer of the original model after applying the test data
Sigmoid _LUT(LUT parameters compressed to Float16)
Optimisation of the Sigmoid function
Rapberry pi 4 Model B
Attack-free
Attack
Released Model
CAN_Intrusion_Test_Dataset(Attack_free and Attack)
Deployment of Attack Detecion Model on Edge
In-vehicle embedded computing device

Fig. 1: End-to-End Prediction of Binary Classification Model and LUT-based method to classify attack state

B. Threat Model

This paper presents attack-free data and three distinct types of attacks: DoS, fuzzy, and impersonation attacks.


(a) DoS Attack

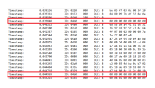
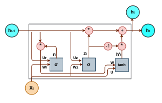

(b) Fuzzy Attack

Fig. 2: Instances of Attack in CAN

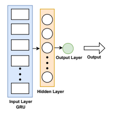
Attack-Free State signifies uninterrupted communication in the CAN bus. The dataset includes 2,369,398 data frames unaffected by attacks. In a DoS attack, the attacker floods the CAN ID with zeros and injects CAN messages with random data payloads, overwhelming the target node with excessive requests and rendering it inaccessible to legitimate nodes, as depicted in Fig. 2a. The dataset comprises 656,579 instances of DoS attack messages. In a fuzzy attack, the attacker scrutinizes in-vehicle data frames and targets specific IDs to flood with random anonymous data, causing delays in accessing other nodes, as shown in Fig. 2b. The dataset includes 591,990 instances of fuzzy attack messages. In an impersonation attack, the compromised node replaces the benign data by mimicking the identity of a legitimate data frame from an authentic node, leading to sudden and unexpected consequences for the CAN channel. The dataset contains 995,472 impersonation attack messages [17], [19].

C. Dataset-Preprocessing

To ensure the applicability of SCAN, we preprocess the dataset extracted from [19], categorized from CAN traffic obtained through the On-Board Diagnostics-II (OBD-II) port of a KIA Soul during message insertion attacks, containing 46,13,909 CAN dataframes. In this paper, the dataset consists of 4,613,439 unprocessed CAN data frames, including the attributes TimeStamp (recorded time), CAN ID (data frame identifier), DLC (length of the data frame), and Data Payload (data bytes quantity). It encompasses three attack types, as discussed in Section III-B, and is divided into two classes: label 0 for attack-free data (2,369,398 instances) and label 1 for attacks (2,244,041 instances). The dataset is split into 10% test data, 10% validation data, and the remaining for training. All CAN data frames are continually gathered without interruptions [19]. The time series feature of CAN frames is a crucial input for the attack detection model.



(a)



(b)

Fig. 3: GRU cell and Neural Network Architecture

D. CAN Classification Model Architecture

SCAN features a three-layer architecture for the attack detection model, as shown in Fig. 3b, which consists of 64 GRU cells in the input layer. GRU is chosen for its efficiency in handling time series data, capturing dependencies over time, and requiring fewer parameters than LSTM network. This results in faster training and prediction time, making it suitable for the limited processing power and storage capacity of ECUs [16], [17], [20]. The structure of the GRU cell, depicted in Fig. 3a, includes h_{t-1} representing the state at the previous time step relative to the current time step t. At the current time, the input and output of the GRU module are x_t and h_t, respectively. The GRU module consists of two key structures: the reset gate ($r_t = \sigma(W_r * [h_{t-1}, x_t])$) and the update gate ($z_t = \sigma(W_z * [h_{t-1}, x_t])$). The output candidate value after reset gate processing is represented by $h_t = (1-z_t)*h_{t-1} + z_t * \hat{h_t}$, where $\hat{h_t} = \tanh(W_h * [r_t * h_{t-1}, x_t])$. Here, W_r is the reset gate parameter, W_z is the updated gate parameter, and W_h is the parameter used to generate the output candidate value h_t [16], [20]. Furthermore, following the input layer, the detection model includes a dense layer with 8 hidden neurons using a ReLU activation function, enabling low-cost computations. The output layer is a single dense neuron with a sigmoid activation function for binary classification. It is important to note that the purpose of this work is to classify data frames as either attack or normal on the embedded platform. In our future research, we intend to distinguish between various forms of attack. The proposed architecture is further improved by incorporating LUT-based activation implementation.

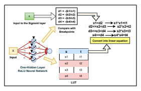


Fig. 4: LUT Approximation of Non-linear Function

E. Improvements and Challenges on Activation Implementation

We're examining the efficiency of non-linear functions like sigmoid, which demands significant processing time and computational resources due to complex arithmetic involving transcendental functions such as exponentiation. The traditional approach requires intricate datapaths and controls for floating-point calculations, leading to extended processing times and substantial hardware requirements. In contrast, approximation methods like Taylor's series or other hybrid approximation methods demand additional computational resources during inference, while saving a single LUT with limited parameters is a simpler and faster choice. However, in the work [21], the authors provide a general approach for LUT-based activation realization. Nevertheless, different activations demand different processes and different hyperparameter selections for LUT parameter generation. In this work, we focus on the problem of LUT-based realization of sigmoid activation only, represented as $\sigma(x) = \frac{1}{1+e^{-x}}$. Converting a neural network (NN) with a single hidden layer ReLU activation into a LUT is recognized as a universal approximator [21] and we use this approach to generate the LUT parameters by improving the hyperparameters in the training setup.

Consider, as shown in Fig 4, we are trying to approximate $\sigma(x)$ in the LUT form where $\sigma(x) = sx + t$, s and t are the LUT parameters, and x is the input value to the LUT. An offline [21] training process is used to train the neural network (NN) to generate LUT parameters using NN parameters n_i, b_i, m_i where n_i = first layer weights, b_i = first layer bias, and m_i = second layer weights. Subsequently, the trained NN has hidden neurons, $y = nx + b$, and forms an approximation of the desired function. The approximation is decomposed into break points $d_i = \{-\frac{b_i}{n_i}\}_{i=1:N-1}$, which are sorted in ascending order to span the input range(X) that obtained after training of the attack detection model Fig. 3b.

Using these sorted breakpoints, we now form the LUT as shown below:

$$z_i = \sum_{j=1}^{N-1} m_j y_j$$
$$= m_i[\sum_{j=1}^{i} n_j^{+} + \sum_{j=i+1}^{N-1} n_j^{-}]x + m_i[\sum_{j=1}^{i} b_j^{+} + \sum_{j=i+1}^{N-1} b_j^{-}],$$

$where\ if\ n_j > 0$, $n_j^{+} = n_j$, $b_j^{+} = b_j$ $\forall j = 1$ to i, and if $n_j < 0$, $n_j^{-} = n_j$, $b_j^{-} = b_j$ $\forall j = i+1$ to $N-1$, Else $n_j^{+} = n_j^{-} = b_j^{+} = b_j^{-} = 0$

$= s_i x + t_i$ if $(b_i < x_i < b_{i+1})$, $s_i, t_i =$ LUT parameters and x = Input value to the LUT (1)

Here, $s_i = m_i[\sum_{j=1}^{i} n_j^+ + \sum_{j=i+1}^{N-1} n_j^-]$, $t_i = m_i[\sum_{j=1}^{i} b_j^+ + \sum_{j=i+1}^{N-1} b_j^-]$ is generated by training the one-hidden-layer ReLU network.

This decomposition results in a series of linear functions, z_i, as depicted in Fig. 4, each specified by approximation parameters, s_i, t_i, stored in LUT for each interval of the breakpoints. It is important to understand that the LUT parameters and the breakpoints in the above equation remain fixed values once the dedicated NN has been trained for a specific non-linear operation. As depicted in Fig. 4, the possible input (X), given to compute the sigmoid outcome, needs to be compared with the sorted breakpoints first. If $X < b_i$ then it will fetch the i^{th} row of the LUT and produce the activation output value. However, to obtain accurate sigmoid-related LUT parameters, the hyperparameter plays a crucial role in the training setup of the NN. After extensive training, we obtained hyperparameters as detailed in Table I in Section IV.

IV. Experiment

A. *Impementation of the Attack Detection Model*

Before proceeding for edge deployment, we must validate the proposed attack detection model's efficacy by assessing its performance on the test set. Initially, we focus on the training of GRU-based DNN, which is the core component of SCAN, on the dataset, which is detailed in Section III-C. The training process is conducted using the TensorFlow (TF) framework and trained in Google Colab-TPUv2 for 50 epochs, with a batch size of 512. It employs the Adam optimizer with a learning rate of 0.01, and binary cross-entropy is chosen as the loss function due to integer-format labels. After training completion, an evaluation of the test data is conducted to measure accuracy, latency, and memory usage, as discussed in the results section. For predicting data types (benign or assault) based on probabilities (p) generated by the trained DNN, a simple thresholding approach is employed: if $p < 0.5$, it is classified as benign; otherwise, it is classified as an assault. Subsequently, the original TF model is implemented without any compression technique and undergoes quantization to produce TFLite models, including TFLite-default (TFL-D), TFLite-floating-point 16 (FP16), and TFLite-Integer 8 (INT8). These models are then deployed on the edge for latency evaluation. It is important to note that the experimental platform for the edge comprises a Raspberry Pi Model B, equipped with a Quad-core 64-bit ARM-Cortex A72 processor, 2 GB RAM, Debian GNU/Linux 12 (bookworm) operating system, TF-2.12, and Sklearn-1.3.

The next objective is to reduce the computation of the target function, namely the sigmoid activation, in the implemented DNN. It is to note that the experimental platform, we utilize for the training of NN with one hidden layer using ReLU activation and input-output layers for LUT creation, has an Intel Z2 i7-11700 processor, 64 GB of RAM, 500 GB of SSD, Jupyter Notebook as a machine learning platform having TF-2.12, and Sklearn-1.3. However, the training setup for the LUT creation is crucial, and that is where our focus lies.

B. *Preparation of the LUT on the Server*

TABLE I: **Training Setup for LUT Approximation for Sigmoid Function**

Target Function	Sigmoid
# hidden neurons	16
Input Range, sample size	(-2,2), 10000
Epochs, Batch size	150, 64
Weight and Bias Initializer	Random

As we mentioned in Section III-E, and Fig. 4, the complete process of converting a non-linear function into a LUT involves extensive training. To evaluate the performance of this universal approximation network, inference testing is conducted using random data in the server. This enables us to substitute LUT parameters for different target functions, thereby mitigating computational delays for embedded devices.

After conducting extensive experiments to determine optimal parameters that yield accurate approximations, we consider some architecture hyperparameters and optimization hyperparameters, which will be discussed in this section. Hyperparameters have a crucial role in the neural network performance; given this context, for the training dataset for constructing an LUT, we evenly sample data within a given input range of interest. Determining an appropriate range of input data for the successful training of approximation networks is essential, as non-linear operations have distinct areas of interest. Moreover, it is crucial to appropriately initialize the neural network parameters in order to determine the LUT parameters effectively. Table I offers a condensed summary of the methodology employed in establishing the input data range and other hyperparameter initialization for approximating sigmoid non-linear function via LUT. Insights gained from these tests enabled the imposition of constraints on input range, sample size, epochs, batch size, and parameter value initialization.

- *Input Range and Sample Size:* The input dataset for the target non-linear operations undergoes thorough sampling within a specified range of interest to ensure comprehensive coverage. Successful training of approximation networks hinges on selecting an appropriate input data range, given the distinct regions of interest inherent in non-linear processes. After extensive trials, an input range of (-2, 2) is found optimal, with 10,000 evenly distributed samples, provided for training the NN to approximate the sigmoid function.
- *Epochs and Batch Size:* The performance of a model hinges significantly on the optimal parameters for batch size and epochs. However, underfitting, which occurs due to insufficient epochs, results in the model's failure to capture underlying data patterns. Conversely, overfitting, caused by excessive epochs, leads to the model relying

too heavily on the training set and losing its generalization ability. The choice of batch size is crucial, affecting both the model's generalization and training duration. For training sigmoid, 150 epochs are deemed sufficient. Given its complexity, multiple training rounds on all batches of the dataset are required to learn all values, thus 64 batches with 10,000 samples each are adequate.

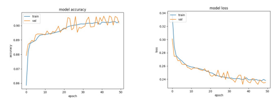
- *Number of Hidden Neurons and Parameter Initialization:* Proper weight initialization is vital for the accuracy of neural network models during training, as it can mitigate issues like vanishing gradient problems. After extensive experimentation, it's determined that random weight and bias initialization provide a satisfactory approximation for the sigmoid function. This initializer effectively balances positive and negative parameter values, enhancing model performance. Additionally, the choice of the number of hidden neurons is crucial, depending on the function's complexity. For instance, 16 hidden neurons suffice to achieve satisfactory accuracy in approximating the parameters of the sigmoid function.

In order to assess the accuracy and delay of the LUT function, the input values to the sigmoid layer on the trained DNN are extracted by applying the test dataset, illustrated in Fig. 1 and afterward, assess LUT by quantizing its parameters to FP16 on these input values. This experiment continues on the low-powered edge device to evaluate the latency of both the LUT and the sigmoid function.

V. Result and Analysis

The main objective of SCAN is to assess the ability of a server-trained attack detection model to conduct real-time inference on an embedded device. To examine the prediction delay, the test content consists of a single CAN message derived from the test data. The test timing starts while fed into the neural network for prediction, and the timing concludes once the probability is generated. The training accuracy achieved is 90.28%, as depicted in Fig. 5a, and the test accuracy of TF model and TFL-D is 90.20%, resulting in our model working well on unseen data. The TFLite-FP16 model maintains the same accuracy as the original TF model due to FP16 offers a large dynamic range, similar to FP32 (Floating Point - 32), for its exponent component. This helps in maintaining the representational capacity for large and small values. The TFLite-INT8 model achieved an accuracy of 80.95%, which shows the quantization error in terms of a small amount of accuracy drop here. However, we assess the ability of server-trained models on embedded devices to examine real-time inference. Table II compares the server-based and edge-based prediction of the attack detection model in terms of accuracy, delay, and memory usage. Nevertheless, Fig. 5b illustrates a declining pattern in training loss and an early decline followed by a period of stability in validation loss, which results in well-trained parameters. Further, in terms of memory usage on the server, the TF model uses 206,248 bytes, whereas the TFLite-FP16 and TFLite-INT8 models use 39,460 bytes and 27,824 bytes, respectively. This makes the quantized FP-16 and INT-8 TFLite models suitable for deployment on low-capacity edge devices, compared to the original TF model. Additionally, the TF model requires 92.80 ms to make a prediction for a single data point on the server. In contrast, TFLite-FP16 takes 0.96 ms, and TFLite-INT8 takes only 0.84 ms significantly reducing the delay compared to the TF model while maintaining an adequate accuracy level. Furthermore, the edge prediction on a single data point, using TFLite-FP16 and TFLite-INT8, demonstrates excellent latency performance.

Nevertheless, to optimize the sigmoid function, we have undertaken performance testing using a distinct random dataset tailored for the training of the target function to evaluate the LUT approximation capability comprehensively. This approach thoroughly scrutinizes the LUT's effectiveness for a particular function, providing insights into its performance and suitability for different scenarios. After many trials, we fixed the particular hyperparameters since they play a crucial role in training algorithm behavior and model performance. Through experimentation, our aim is to achieve error rates of less than 40%. Table III, indicates a 47% error rate for the sigmoid function, illustrates the findings, highlighting the importance of hyperparameters and their impact on sigmoid function approximation using LUT.



(a) (b)

Fig. 5: Accuracy and Loss graph for Binary Classification of Attack Detection Model on 50 epochs

TABLE II: **Attack Detection Model Performance on Server and Edge** [B=bytes, S=server, E=edge, NR = Not Recorded]

Matrices	TF	TFLite-D	TFLiteFP16	TFLiteINT8
Accuracy	90.20%	90.20%	90.20%	80.95%
Memory	206248 B	67400 B	39460 B	27824 B
Delay(S)	92.80 ms	1.09 ms	0.96 ms	0.84 ms
Delay(E)	NR	NR	12.39 ms	1.21 ms

TABLE III: **LUT Based Activation Implementation results**

Target Function	Input Range	sample size	error rate(under 40%)
Sigmoid	(0,1)	277	47%

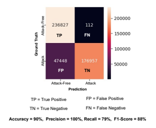


Fig. 6: Confusion Matrix of Attack Detection Model using LUT-Based Approximated Activation

Furthermore, the F1-score, Recall, and Precision metrics are tested for LUT implementation. Upon applying the LUT parameters to the entire test dataset, it yielded 90% accuracy, 100% precision, 79% recall, and an F1-score of 88%. Fig. 6 displays the confusion matrix of the LUT implementation on the entire test dataset. Moreover, we performed a comparative analysis between the implementation of LUT-based approach and the sigmoid function, as shown in Table IV, and the results

TABLE IV: **LUT Approximated vs Actual Activation Implementation (Target Function: Sigmoid)**[S=server, E=Edge]

Matrices	Sigmoid	LUT
Accuracy	99%	90%
Delay(one sample(S))	0.10 ms	0.05 ms
Delay(one sample(E))	0.42 ms	0.16 ms
Delay(461344 test data(S))	1.7763388 seconds	1.6880729 seconds

are highly satisfactory. Prior to conducting experiments with the LUT parameters, we convert them to FP-16 format in order to facilitate deployment on the edge. Additionally, the results shown in the referenced table are achieved by compressing the input values to FP-16 format to ensure optimal compatibility with the LUT parameters. The LUT has a latency of 0.16 ms to predict a single data point on the edge; in contrast, the sigmoid function necessitates 0.42 ms for the same. In addition, we measure a comparable evaluation of the LUT and sigmoid function for the entire test dataset on the server. The results presented in Table IV indicate a 4.96% reduction in predicting 461344 data samples on the server.

TABLE V: **Comparision with Existing Work After Complete Optimization of Sigmoid Activation**[E=Edge, NR = Not Recorded]

	Accuracy	Precision	Recall	F1-Score	Delay(E)
Javed et al. [17]	-	93.69%	93.91%	93.79%	NR
Ma et al. [16]	-	-	-	-	0.178ms (Jetson Nano)
SCAN	90%	90%	79%	88%	0.16ms (Raspberry PI)

However, Table V depicted that our proposed model could achieve adequate precision compared to the state-of-the-art models. Nevertheless, the latency decrease in our implementation is approximately 10% for one datapoint compared to existing work, which is a significant improvement.

VI. Conclusion

The purpose of SCAN is to enable end-to-end prediction of the attack detection model on low-cost embedded device like Raspberry PI board without any powerful processing head like GPU and also optimize the costly function in the detection model. While, existing work [16], relied on costly boards like the Jetson Nano with high-end GPU support. Moreover, recent developments for AVs implemented on powerful boards like Qualcomm Snapdragon Ride, Google Coral board, etc., which are very costly. Our results show that using deep learning models in CAN bus systems is practical and can improve upon well-established techniques like GRU. Nevertheless, our future work includes enhancing SCAN to detect attacks in real-time, aiming for a reliable CAN network for AVs by leveraging real-time datasets in low-powered boards, which will also explore the 6G layer for enhanced security.

References

[1] Bozdal M, Samie M, Aslam S, Jennions I. "Evaluation of CAN Bus Security Challenges." *Sensors (Basel)*, 20(8):2364, 2020 Apr 21.

[2] M. Hataba, A. Sherif, M. Mahmoud, M. Abdallah and W. Alasmary. "Security and Privacy Issues in Autonomous Vehicles: A Layer-Based Survey." *IEEE Open Journal of the Communications Society*, vol. 3, pp. 811-829, 2022.

[3] S. Purohit and M. Govindarasu. "ML-based Anomaly Detection for Intra-Vehicular CAN-bus Networks." in *2022 IEEE International Conference on Cyber Security and Resilience (CSR)*, Rhodes, Greece, pp. 233-238, 2022. doi:10.1109/CSR54599.2022.9850292.

[4] Vinayak Tanksale, "Controller Area Network Security Requirements," *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*. Electrical and Computer Engineering, Purdue University, West Lafayette, USA, 2020.

[5] András Gazdag, Csongor Ferenczi, Levente Buttyán. "Development of a Man-in-the-Middle Attack Device for the CAN Bus." in *2021 Conference on Information Technology and Data Science*, Debrecen, Hungary, November 6–8, 2020.

[6] Hyo Jin Jo and Wonsuk Choi. "A Survey of Attacks on Controller Area Networks and Corresponding Countermeasures." *IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS*, VOL. 23, NO. 7, JULY 2022.

[7] Lokman et al. "Intrusion detection system for automotive Controller Area Network (CAN) bus system: a review." *EURASIP Journal on Wireless Communications and Networking* (2019)184.

[8] Donghyun Yu, Ruei-Hau Hsu, Jemin Lee, Sungjin Lee. "EC-SVC: Secure CAN Bus In-Vehicle Communications With Fine-Grained Access Control Based on Edge Computing." *IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY*, VOL. 17, 2022

[9] H. Oguma, A. Yoshioka, M. Nishikawa, R. Shigetomi, A. Otsuka, and H. Imai, "New attestation based security architecture for in-vehicle communication," in Proc. IEEE Global Telecommun. Conf., Nov. 2008, pp. 1–6

[10] P. Kleberger, T. Olovsson, and E. Jonsson, "Security aspects of the in-vehicle network in the connected car," in Proc. IEEE Intell. Veh. Symp., Jun. 2011, pp. 528–533.

[11] H. Schweppe, Y. Roudier, B. Weyl, L. Apvrille, and D. Scheuermann, "Car2X communication: Securing the last meter—A cost-effective approach for ensuring trust in Car2X applications using in-vehicle symmetric cryptography," in Proc. IEEE Veh. Technol. Conf. (VTC Fall), Sep. 2011, pp. 1–5

[12] A. V. Herrewege, D. Singelee, and I. Verbauwhede, "CANAuth—A simple, backward compatible broadcast authentication protocol for CAN bus," in Proc. ECRYPT Workshop Lightweight Cryptogr., Nov. 2011, p. 20.

[13] Mundhenk, Philipp and Paverd, Andrew and Mrowca, Artur and Steinhorst, Sebastian and Lukasiewycz, Martin and Fahmy, Suhaib A. and Chakraborty, Samarjit. "Security in Automotive Networks: Lightweight Authentication and Authorization." *ACM Trans. Des. Autom. Electron. Syst.* 22, 1–27, 2017.

[14] S. Tariq, S. Lee, H. K. Kim, and S. S. Woo, "Detecting in-vehicle can message attacks using heuristics and rnns," in Proc. Int. Workshop Inf. Oper. Technol. Secur. Syst., 2018, pp. 39–45

[15] A. Taylor, S. Leblanc, and N. Japkowicz, "Anomaly detection in automobile control network data with long short-term memory networks," in Proceedings of the 2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA), pp. 130–139, IEEE, Montreal, QC, Canada, 2016 October.

[16] Haoyu Ma , Jianqiu Cao , Bo Mi , Darong Huang , Yang Liu , and Shaoqian Li. "A GRU-Based Lightweight System for CAN Intrusion Detection in Real Time," Hindawi Security and Communication Networks, Volume 2022, Article ID 5827056, 11 pages.

[17] Abdul Rehman Javed, Saif ur Rehman, Mohib Ullah Khan, Mamoun Alazab, and Thippa Reddy G, "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, APRIL-JUNE 2021

[18] Shahroz Tariqa, Sangyup Leea, Huy Kang Kimb, Simon S. Woo, "CAN-ADF: The controller area network attack detection framework," Computers Security, Volume 94, July 2020, 101857

[19] H. Lee, S. H. Jeong, and H. K. Kim, "Otids: A novel intrusion detection system for in-vehicle network by using remote frame," in Proc. 15th Annu. Conf. Privacy, Secur. Trust, 2017, pp. 57–5709

[20] Chung, Junyoung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. "Empirical evaluation of gated recurrent neural networks on sequence modeling." arXiv preprint arXiv:1412.3555 (2014).

[21] J. Yu et al., "NN-LUT: Neural approximation of non-linear operations for efficient transformer inference," in Proc. Design Autom. Conf. (DAC), Jul. 2022, pp. 577–582

[22] Chih-Hsiang Chang, Hsu-Yu Kao, and Shih-Hsu Huang. 2019. Hardware implementation for multiple activation functions. In IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW). 1–2.