Improving In-vehicle Networks Intrusion Detection Using On-Device Transfer Learning

Sampath Rajapaksha*, Harsha Kalutarage*, M.Omar Al-Kadri†, Andrei Petrovski* and Garikayi Madzudzo‡
*Robert Gordon University, †Birmingham City University, ‡Horiba Mira Ltd
{s.rajapaksha, h.kalutarage, a.petrovski}@rgu.ac.uk, omar.alkadri@bcu.ac.uk, garikayi.madzudzo@horiba-mira.com

***Abstract*—Modern automobiles are equipped with a large number of electronic control units (ECUs) to provide safe, driver assistance and comfortable service. The controller area network (CAN) provides real-time data transmission between ECUs with adequate reliability for in-vehicle communication. However, the lack of security measures such as authentication and encryption makes the CAN bus vulnerable to cyberattacks, which affect the safety of passengers and the surrounding environment. Intrusion Detection Systems (IDS) based on one-class classification have been proposed to detect CAN bus intrusions. However, these IDSs require large amounts of benign data with different driving activities for training, which is challenging given the variety of such activities. This paper presents CAN-ODTL, a novel on-device transfer learning-based technique to retrain the IDS using streaming CAN data on a resource-constrained Raspberry Pi device to improve the IDS. Optimized data pre-processing and model quantization minimize the CPU and RAM usage of the Raspberry Pi by making CAN-ODTL suitable to deploy in the CAN bus as an additional ECU to detect in-vehicle cyber attacks. Float 16 quantization improves the Tensorflow model with 78% of memory and 83% of detection latency reduction. Evaluation on a real public dataset over a range of seven attacks, including more sophisticated masquerade attacks, shows that CAN-ODTL outperforms the pre-trained and baseline models with over 99% detection rate for realistic attacks. Experiments on Raspberry Pi demonstrate that CAN-ODTL can detect a wide variety of attacks with near real-time detection latency of 125ms.**

I. Introduction

Recent improvements in vehicle autonomy and connectivity provide a more comfortable, safe and convenient experience to drivers and passengers. This includes services such as adaptive cruise control, lane departure warning, automated parking assistance, and infotainment systems [1]. To provide these services, automobiles are equipped with numerous microprocessor-based electronic control units (ECUs), such as engine and powertrain control units. The exchange of various information between these ECUs requires a unified network which supports real-time data transmission with sufficient bandwidth and adequate reliability [2]. Controller area network (CAN), meets these requirements and therefore considered as the most widely used in-vehicle network protocol. Despite the CAN bus having the required communication capabilities, it lacks security features such as message encryption and authentication. These security flaws make the CAN bus vulnerable to cyberattacks. Recent experiments have demonstrated that it is possible to attack modern automobiles by compromising the CAN bus [3]. This can be achieved through obtaining access to OBD-II port [4], over the air (OTA) updates or communication channels such as Bluetooth, Wi-Fi and cellular networks. These attacks enable attackers to gain physical control of the vehicle and activate different vehicle functions such as engaging the brakes, turning on the windshield wipers, activating the locking mechanism, and changing the vehicle's speedometer values. Therefore, cyberattacks on in-vehicle networks can pose direct threats to passengers, owners, operators, and the surrounding environment of the vehicle.

Symposium on Vehicles Security and Privacy (VehicleSec) 2023
27 February 2023, San Diego, CA, USA
ISBN 1-891562-88-6
https://dx.doi.org/10.14722/vehiclesec.2023.23088
www.ndss-symposium.org

IDSs have been proposed to identify malicious activities in in-vehicle networks (IVNs). IDSs can be classified into two categories as signature-based and anomaly-based detection [5]. Anomaly-based detection overcomes the limitation of signature-based detection which is less capable of detecting novel attacks. The structure and semantics of CAN data are not available to open access and are stored in a database-like file known as the database CAN (DBC). DBC file is considered confidential proprietary for vehicle manufacturers [6]. A modern vehicle includes up to 150 ECUs which leads to transmitting up to 2000 CAN frames per second [7]. This demands an IDS which have real-time or near real-time detection latency in a computationally constrained environment. Attackers could change different fields of CAN data to compromise the in-vehicle network. This creates both point and contextual anomalies in CAN data. Therefore it requires considering the context of CAN data to detect some sophisticated attacks such as masquerade attacks. These factors make developing an IDS for widespread adoption challenging. One-class classification-based IDSs have been successfully used in in-vehicle networks due to their capability to detect a wide variety of attacks and only require benign data to train the algorithms [8], [9]. However, one major limitation of this approach is a requirement of a large good representative sample of benign data which represents different driving scenarios. Having a small training dataset prone to higher false positives or negatives. Collecting a good benign representative sample is challenging as vehicles have a wide variety of benign driving behaviours. Collecting such a large dataset and training a deep learning model is also computationally expensive.

To alleviate this problem and overcome the aforementioned challenges, this paper proposes an on-device transfer learning technique (ODTL) to retrain the classification layer of context-aware shallow neural network-based IDS in a resource-constrained Raspberry Pi device. The main contribution of this paper can be summarized as follows:

1) This paper proposes an on-device transfer learning technique, named as CAN-ODTL, to retrain a one-class classification-based CAN IDS in a resource-constrained environment. The proposed model address the need for having a large benign dataset to train the algorithm by incremental retraining the classification layer on the IDS with streaming CAN data.
2) The proposed method uses optimized data pre-processing algorithm to pre-process the streaming CAN data. This helps to improve the detection latency while minimizing CPU and RAM usage of Raspberry Pi.
3) CAN-ODTL uses the quantization technique to reduce the model size and detection latency. Optimized float 16 quantization helps to achieve near-real-time detection on Raspberry Pi.

The rest of this paper is structured as follows: section II provides the background. Section III presents the related work. The proposed method is explained in section IV. In section V, the experiment results and performance evaluations are presented. Finally, section VI concludes the paper.

II. Background

A. Controller Area Network (CAN Bus)

CAN is a lightweight multi-master, message-based protocol developed by Bosch. This is used by ECUs as their main in-vehicle network protocol. CAN bus is divided into three sub-systems as the drive train, convenience and infotainment based on the supported data transmission rate. Time-critical ECUs such as the engine controller and break controller units use high-speed drive train. Infotainment systems, including radio navigation and phone interface box use low-speed infotainment CAN bus whereas, ECUs such as door control and climate control units use low-speed convenience CAN bus. A CAN dataframe includes seven fields. These are: start of frame (SOF), arbitration field (CAN ID), control field (DLC), payload (data), cyclic redundancy code (CRC), acknowledgement (ACK) and end of frame (EOF). These seven fields are depicted in Figure 3. CAN payload and ID fields are considered as the most important fields of a CAN frame. CAN payload field supports up to 64 bits of data and this includes different values such as sensor data, category data, constant data or cyclical counter data [10]. CAN ID is a unique identifier for an ECU. This is used to prioritize the messages based on the CAN ID value. Higher values get the lower priority, while lower values get the higher priority. This is referred to as the arbitration mechanism and uses to manage concurrent messages. ID-based arbitration ensures that the higher priority messages get access to the CAN bus when multiple IDs transmit messages concurrently.

B. Attacks on the CAN bus

CAN bus is designed considering the requirement of in-vehicle network communication without considering the security-related features. Therefore, by design, CAN bus is vulnerable to cyber attacks. Since the CAN frames are not encrypted and use broadcast property, an attacker can get access to all the messages that transmit through the CAN bus. This is referred to as sniffing attack. Due to the lack of authentication, any node could transmit a message and therefore, an attacker can inject malicious frames into the CAN bus by compromising an ECU or through a new node. Additionally, the ID-based priority mechanism also makes the CAN bus vulnerable as it allows injecting higher-priority IDs and creating Denial-of-Service (DoS) attacks.

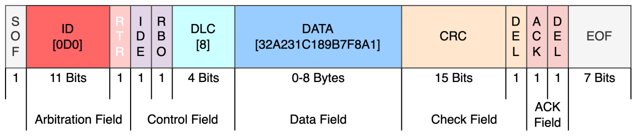


Fig. 1: CAN bus dataframe with example values for ID, DLC and Payload fields

Attacks on the CAN bus can be mainly classified as injection and masquerade (impersonation) attacks. Injection attacks insert new frames into the CAN bus. Some common injection attacks on the CAN bus are DoS, fuzzing, replay and spoofing. DoS attacks try to make communication services unavailable by sending a large number of high-priority IDs. In a fuzzing attack, the attacker sends frames with randomly generated ID, DLC and payload values. Replay attacks record the benign frames of the network and inject them at different times. Spoofing attacks, also referred as targeted ID attacks, are somewhat similar to fuzzing attacks but target specific IDs without randomly injecting them. The payload of these frames changes into malicious values. It is easy to implement injection attacks. However, it requires the attacker to continuously inject these frames at a higher rate to override the benign frames and change the behaviour of the vehicle [11], [12]. In a masquerade attack, a compromised node impersonates another node, which requires extremely low-level access to the CAN transceiver and therefore more difficult to perform than simple injection attacks [12]. Masquerade attacks require advanced detection methods as typically these are not changing the ID frequency-related features like injection attacks.

C. CAN ID Sequences

ECUs in a vehicle typically transmit frames at a fixed intervel [4], [8], [11]. Since CAN uses an ID-based priority mechanism, only one frame transmits at a given time. Therefore, CAN data can be considered as time series data. This behaviour creates a finite set of CAN ID sequences for a fixed window size. We used a publicly available real CAN dataset, the Real ORNL Automotive Dynamometer (ROAD) CAN Intrusion dataset [12] to analyze this behaviour. This dataset includes 106 CAN IDs and a benign data sample of 30 minutes drive was selected for this analysis. Figure 2 depicts the frequency for the top 20 sequences for five consecutive IDs (window size). According to this, some sequences appear in high frequency, whereas others appear less in frequency. During an injection attack, this could create new CAN ID sequences that cannot be observed during benign driving periods. This is mainly because new frames appear in an unusual context. Sophisticated attacks such as masquerade attacks do not inject IDs into the CAN bus. However, as a result of time synchronization issues or no messages from a particular ECU for a brief period of time, this might still

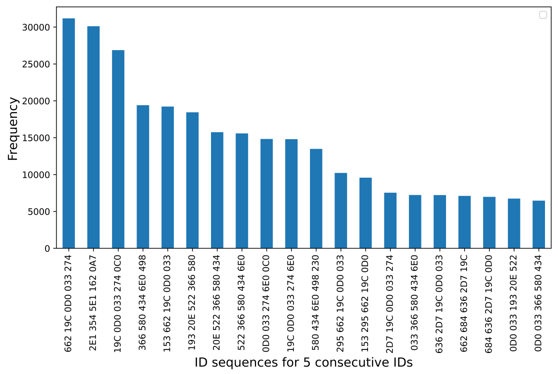


Fig. 2: The top 20 CAN ID sequences for five consecutive IDs

create new anomalous sequences [8], [13]. Therefore, it is possible to learn these benign sequences and identify malicious frames in the CAN bus. Even though ECUs transmit frequent frames, due to the randomness incurred from jitters, it might create a large number of benign CAN ID sequences for the fixed window size [13]. As a result, it requires a large benign dataset representing different benign driving behaviours to learn all these benign sequences. Therefore, an on-device incremental learning approach is more appropriate to handle this requirement.

III. Related work

Early experimental attacks [4], [11] and more recent experiments such as [14], [15] motivated researchers toward implementing countermeasures against cyber attacks on the CAN bus. Cyber attackers target different fields of CAN data such as ID field or payload values and therefore, IDSs designed for different CAN data fields. These can be classified into ID-based, payload-based and CAN Frame-based IDSs [6]. CAN payload data were used in the IDS proposed in [16]. This used a Gated Recurrent Unit (GRU)-based recurrent autoencoder to detect anomalies in the CAN bus. In [17], the authors used a Long Short-Term Memory (LSTM)-based IDS by introducing a self-attention layer to capture the most relevant information of input sequences. Both IDSs processed ID-wise data independently and trained separate models for each ID. Since modern vehicles could include up to 150 CAN IDs, these IDSs require training a large number of IDSs using benign data. Thus, these IDSs are not suitable for deployment in resource-constrained in-vehicle networks to detect attacks in near-real time. Generally, CAN payload-based IDSs are computationally expensive due to the complexity of the multi-dimensional payload data [6].

On the other hand, CAN ID-based IDSs used CAN ID sequences or time-related features. In [18], the authors trained an IDS using a LSTM model. This approach predicted the next CAN ID and then compared the predicted ID with the actual ID. This only achieved 60% accuracy. GAN and convolutional adversarial autoencoder-based model [19] was trained with unlabeled data to learn the normal patterns. Experimental results showed that the proposed model outperforms baseline models for both detection rate and latency. However, this work was only limited to message injection attack detection. In [9], the authors developed a context-aware anomaly detector for monitoring cyber attacks on the CAN bus using sequence modelling. N-gram distributions were used to build the sequence model. N-gram-based methods are capable of capturing the context of CAN sequences. However, this often leads to high computational overhead as N increases. An ensemble model based on GRU and time-based models was proposed in [8] to overcome the computational inefficiency of N-gram based models and to detect a wide variety of attacks. The GRU-based model predicted the next CAN ID, whereas the time-based model monitored ID inter-arrival times to detect anomalies. Despite the advantage of having only benign data to train these one-class classification based IDSs, the major limitation is the requirement of having a large dataset to train the model. To address this issue, this work presents an on-device transfer learning technique to incrementally retrain the CAN IDS.

IV. Methodology

This section explains the CAN intrusion detection model and the proposed CAN IDS on-device transfer learning technique to overcome the limitation of one-class classification-based IDS training for the CAN bus.

A. *CAN intrusion detection model*

By utilising the sequential behaviour of the CAN frames, CAN intrusion detection can be formulated as the next CAN ID prediction task. Given a context of n IDs from the left side (pre-context) of a CAN sequence, it can predict the next CAN ID with a p probability. This is similar to the next word prediction task in a language model. If the predicted next CAN ID's probability is below a certain threshold, then it can be identified as an anomalous frame in the given context. However, due to the randomness incurred from jitters and ID-based priority, there might be a large number of possible next CAN-IDs for the given context. This can be reduced by considering both the left side and right side (post-context) of a CAN sequence. This is equivalent to the continuous bag-of-words (CBOW) model architecture proposed in [20] where it learns the word vectors representing the middle word's meaning and the context words. Given the context from both sides of an ID reduces the number of possible distinct centre IDs. This helps to achieve a higher accuracy for the predicted CAN ID [8].

Accordingly, this paper uses the centre ID prediction model given the pre and post-context to detect the intrusion in the CAN bus. First, this uses an embedding layer which takes the input of vectorized benign CAN ID sequences. The objective here is to learn accurate word vectors that encode semantic relationships for all the IDs in the CAN bus. Then GRU layer is used to capture the temporal pattern of the sequential data. GRU is a variant of LSTM with only two gates: reset and update. Therefore, GRU is more computationally efficient with low memory overhead [21]. Self-attention layer is used with the GRU layer. The usage of the attention layer in neural networks allows focusing on the relevant parts of the input sequence and selectively output only the most relevant information [17]. This attention layer allows capturing the important information in CAN sequences for long CAN ID sequences. Finally, the classification layer is used with the softmax activation function to get the probability for each CAN ID.

B. CAN IDS On-device transfer learning (CAN ODTL)

As aforementioned, it requires training one-class classification-based CAN IDSs with a large dataset representing different benign driving behaviours. However, collecting datasets covering all types of benign driving behaviours is not feasible. The more appropriate way to handle this issue is to do on-device training in an incremental way until the model trains up to a long duration of time with more data. This can be achieved by deploying the algorithm on the CAN bus and training the algorithm while it is driving. However, ECUs are not capable of training computationally expensive deep learning or shallow learning models due to the limited computation power. Since it is possible to access the CAN bus data through the OBD II port, the algorithm can be deployed in a small device like Raspberry Pi and connected to the CAN bus. Raspberry Pi is a general-purpose embedded computing device with limited memory and processing power. Even though it is used for machine learning and deep learning inferences, model retraining is challenging due to the limited computational resources [22]. As a solution, we propose a transfer learning technique to retrain the classification layer of the deployed model incrementally. Transfer learning is used to improve a learner from one domain by transferring information from a related domain. This concept can be used to improve a pre-trained model in the same domain by including a new classification layer or only retraining the last layers of the model with more data. This is also referred as model fine-tuning. Typically, transfer learning is used when the training data is expensive or difficult to collect. Therefore, this work use transfer learning to retrain the classification layer of the pre-trained model with more data to achieve a higher intrusion detection rate. This process is depicted in Figure 3.

The model is divided into two parts as encoder f_ϕ and decoder g_θ. The encoder includes the first three layers of the model and the output of the attention layer uses as the input to the decoder, which is the classification layer. First, we train the complete model with a sufficiently large benign dataset. This is trained to minimize the objective function of categorical cross-entropy. This is given by:

$$E = -\sum_{i=1}^{n} y_i log(\hat{y_i}) \tag{1}$$

where y_i is the true lable and $\hat{y_i}$ is the predicted softmax probability for the i^{th} class. Weights W_1, W_2, W_3 and W_4 are learned using backpropagation during the model training. After the pre-training, the encoder model is converted into the more lightweight TFLite version. During this conversion, we also apply quantization to reduce the encoder detection latency. Quantization is an optimization technique that reduces the precision of the numbers used for model parameters. Typically, Tensorflow uses 32-bit floating point numbers. Quantization leads to achieving a better throughput by moving 32-bit numbers into 16 or 8-bit numbers. However, this might reduce the model accuracy slightly due to the loss of precision. The on-device transfer learning procedure is shown in algorithm 1. During the experiments on Raspberry Pi, CAN data log was saved on the micro SD card and streamed from there. In a deployed vehicle, CAN streaming data can be directly used in the same way.

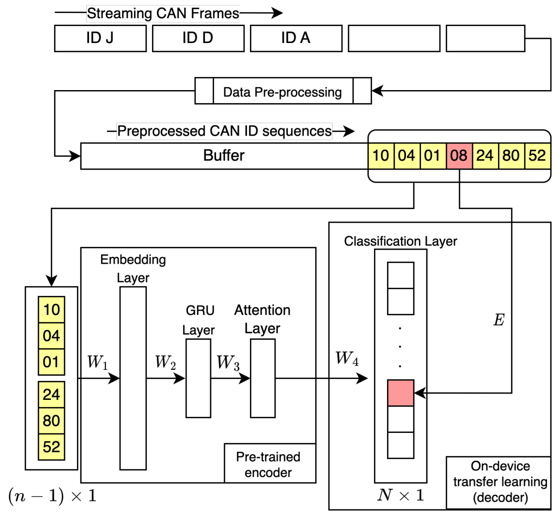


Fig. 3: Workflow diagram of On-device transfer learning

This algorithm uses the pre-trained encoder, CAN streaming data, buffer to save streaming data, sequence window size and batch size as the input. First, this pre-processes the data and extracts the numerical CAN IDs from the streaming CAN frames. Python double-ended queue (deque) is used during the data pre-processing as it supports efficient data append and pop operations from both sides with O(1) time complexity. Extracted IDs are saved in the buffer as streaming CAN data speed is faster than the retraining process. However, this can also be done on the SD card, as it does not require inferencing during the retraining period. A batch of context IDs from the buffer is input to the pre-trained encoder. The output of the encoder along with the batch of centre IDs are used to retrain the classification layer using transfer learning. This is initialized from the pre-trained weights W_4. Each batch is trained for one epoch. According to [23], a learning rate decay schedule is used to improve the incremental training process. During the retraining, updated W_4 weights are kept in the memory until the retraining cycle is ended. In our experiments, this is the end of a CAN log. This can be set into a time period or vehicle start-to-stop in real deployed settings. After triggering the retraining completion, learned weights are saved to the SD card for inferencing or future retraining. Once the model is trained to a satisfactory level such as a few days of training, the trained decoder is converted into the TFLite with the quantization for the inferences. We retrained the model for the available CAN logs. Using the learned weights, this model can retrain again as and when necessary or based on a pre-defined schedule. While retraining, we periodically assessed the progress of the model using a small sample of the benign dataset that had been saved to the SD card.

CAN ODTL anomaly detection procedure is shown in algorithm 2. Counting weak anomalies over a window help to minimize the false positives [8]. Therefore, this approach considers a small observation window of time T to identify anomalous or benign status. Similar to the model retraining procedure, this also preprocesses and extracts the CAN IDs.

Algorithm 1 CAN IDS ODTL procedure

```
Input: Pre trained encoder $f_\phi$, Streaming CAN data $F$, Buffer size $Z$, Window size $W$, Batch size $B$
Output: Retrained decoder $g_\theta$
 1: Init: $ID\_list$ = [ ] $\in Z$
 2: while $F$ is not empty do
 3:     Read $l$ in $F$                              ▷ Read line by line in $F$
 4:     Pre-process $l$,
 5:     Extract $id$
 6:     Append $id$ to $ID\_list$                     ▷ Parallel append
 7:     while $len(ID\_list) \geq W$ do
 8:         Init: Empty arrays X,Y
 9:         while $len(X) \leq B$ do
10:             Append earliest context $ids$ in $W$ to X
11:             Append earliest center $id$ in $W$ to Y
12:             Remove earliest $id$ from $ID\_list$
13:         end while
14:         $z = f_\phi(X)$                           ▷ Output of attention layer
15:         $\hat{Y} = g_\theta(z)$                   ▷ Retraining
16:     end while
17:     Save $W_4$
18:     Convert $g_\theta$ to TFLite format
19: end while
```

Additionally, the time stamp is extracted to identify the time-based observation windows. If the predicted softmax probability for the centre ID is less than a pre-defined threshold ω, the frame is declared as a weak anomaly. These anomalous and benign frames in the observation window are used to detect the window as anomalous or benign.

Algorithm 2 CAN ODTL anomaly detection

```
Input: Streaming CAN data $F$, Anomaly threshold $\omega$, Window threshold $\psi$, Time window $T$
Output: Anomaly status for each window
 1: while $F$ is not empty do
 2:     read $x, y$, time_stamp $t$, $t\_min$
 3:     while $t - t\_min \leq T$ do
 4:         Init: Benign count $C_b = 0$ , Anomaly count $C_a = 0$
 5:         $\hat{y} = g_\theta(f_\phi(x))$           ▷ Quantized TFLite model
 6:         if $\hat{y} < \omega$ then                ▷ for $id$ $y$
 7:             Declare $x$ as a weak anomaly
 8:             $C_a = C_a + 1$
 9:         else
10:             Declare $x$ as a benign
11:             $C_b = C_b + 1$
12:         end if
13:     end while
14:     if $C_a/(C_a + C_b) > \psi$ then
15:         Return Anomaly
16:     else
17:         Return Benign
18:     end if
19:     $t\_min \leftarrow t$
20: end while
```

TABLE I: Description of ROAD dataset attacks

Attack	Description
Fuzzing	Inject random IDs and payloads to achieve unexpected physical behaviours
Max speedometer	Change a payload byte to display false speedometer value
Reverse light on and off	Change a payload bit to change reverse light status

C. Threshold estimation

CAN ODTL used two thresholds. Due to the fact that the training procedure totally depends on benign data, all thresholds must be chosen based on the benign datasets. Therefore, a separate benign dataset is used to estimate both thresholds. For anomaly threshold ω, softmax probabilities were calculated for all IDs in the benign sample. While the minimum values of each ID are ideal as the threshold values to minimize the false positives, there might be a few benign sequences in the threshold estimation benign dataset, which were not observed during the training phase. As a result of this, these benign frames tend to get lower probabilities. Therefore, we consider the Nth lowest quantile values as the anomaly thresholds. Window threshold ψ is defined in a way that it produces a minimum false positive rate for a fixed window size of time t(s). For example, if benign windows produce an average of two false positives, then at least three anomalous frames should be there to consider the window status as anomalous. This helps to reduce the false positives of the proposed model. For both threshold estimations, we used the trained TFLite model.

V. Experiments

This section discusses the selected datasets and attacks, the parameters of the algorithm and performance evaluation.

A. Threat model and datasets

In this work, we use a publicly available dataset ROAD [12] to evaluate the proposed model. ROAD dataset is the most realistic CAN attack dataset with verified attacks [8]. ROAD dataset was collected via OBD-II port under fully compromised ECU mode. This includes 12 benign datasets which cover a wide variety of driving activities such as drive, accelerate, decelerate, break, and reverse. ROAD dataset consists of fabrication and masquerade attacks. A fabrication attack uses a strongly compromised ECU to inject malicious frames which alter the ID and payload fields. This includes fuzzing and targeted ID attacks. In contrast, masquerade attack is the most sophisticated type of attack which uses a strongly compromised ECU to inject malicious messages without changing the CAN ID sequences. In ROAD dataset, this was created during the post-processing by removing the legitimate target ID frames relevant to each injected frame. Masquerade attacks were created for each type of targeted ID attack. The attacks shown in Table I along with their masquerade versions are selected for the performance evaluation.

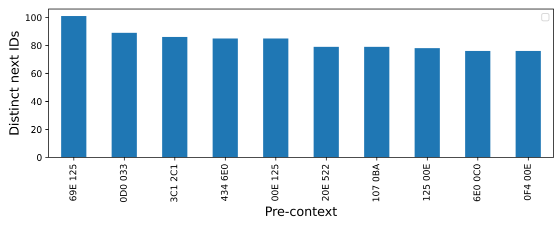
B. Experimental setup

For the initial model training, different subsets of benign data are used by merging into one dataset without changing the temporal behaviour of the CAN data. A separate benign dataset is used as the testing set to evaluate the model loss and accuracy. Similarly, a good representative dataset sample is used for the threshold estimation. Grid search is used to find the optimum hyperparameters. The anomaly detection capability of this model is dependent on the centre ID prediction accuracy. It is expected to obtain a high softmax probability for benign centre IDs, whereas a low softmax probability is expected for anomalous centre IDs. Therefore, accuracy is calculated considering the true and predicted ID, which obtained the highest softmax probability. This is used to monitor the model progress. Both model accuracy and the number of training parameters are considered during the hyperparameter selection to obtain a lightweight model. Accordingly, 11 is selected as the window size by giving 5 IDs as the pre and post-context from each side of the centre ID. The embedding size is set to 50 and only 16 nodes are used in the GRU Layer. The classification layer includes 107 classes, in which one class is allocated for new IDs, which can appear due to some injection attacks. This model architecture resulted in having 5350, 3264, 26 and 1819 trainable parameters in the embedding, GRU, attention and classification layers, respectively. The initial model is trained to 100 epochs with 128 batch size. Early stopping is used to avoid overfitting. Stochastic Gradient Descent (SGD) is used with a 0.01 learning rate due to its high generalization capability for model retraining [23]. During the classification layer retraining, the learning rate is reduced by 1.1 factor for every 5000 epochs to avoid overfitting. The retraining batch size is set to 256. All these parameters are selected based on repeated experiments to achieve optimal retraining for accuracy and time.

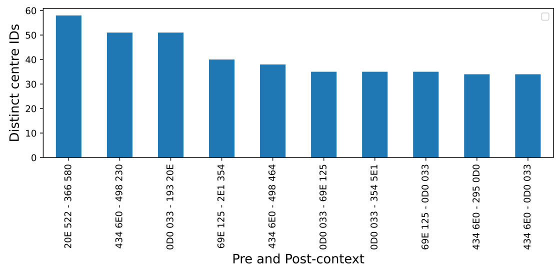
Considering the frame transmission rate of around 2500 frames per second in ROAD dataset, attack datasets split into 100 milliseconds windows to identify attack windows. This can be considered as a smaller window for near-real-time prediction. The window threshold is set to 0.04 based on the lowest false positive rate (average) for the benign dataset. The Anomaly threshold is calculated for each CAN ID. Allowing a small margin to unseen benign data, 0.001^{th} quantile values are considered for the threshold. Tensorflow and Keras libraries are used with python 3.8 for the implementation, whereas a custom training loop is used during the retraining. KerasTuner is used for the grid search. All pre-training experiments run on a MacBook M1 Pro with 16 GB RAM. Raspberry Pi 4 Model B 8GB version is used with a 16GB micro SD card for on-device transfer learning. Tensorflow 2.10 and python 2.8 is installed in Raspberry Pi for the model retraining, whereas TFLite runtime is used for the inference.

C. *Results and discussion*

1) Importance of pre and post-context: We compare the effect of using pre and post-context to predict a CAN ID. Figure 4 shows the results for the top 10 CAN ID sequences. Figure 4a shows the number of possible distinct CAN IDs given two pre-context IDs as the context. For example, given the context as ’69E 125’, there are 100 CAN IDs that can appear as the next CAN ID. In contrast, as shown in the Figure 4b, given the pre-context of ’69E 125’ and the post-context of ’2E1 354’ there are only 40 CAN IDs eligible as the centre ID. Increasing the context size will further reduce this number. This makes the centre ID prediction much more reliable for anomaly detection task in CAN ID sequences than the next ID prediction.



(a) The top 10 distinct next ID count for the given pre-context



(b) The top 10 distinct centre ID count for the given pre and post-context

Fig. 4: Top 10 distinct next and centre ID counts for the given context

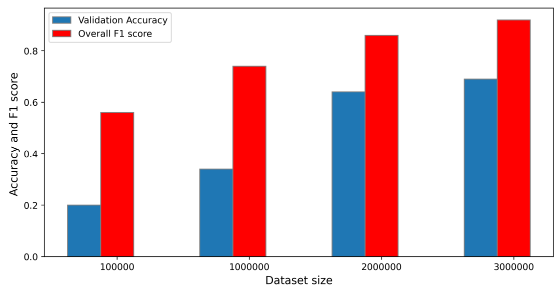


Fig. 5: Accuracy and overall F1-Score improvement with dataset size

2) Effect of dataset size: We experimented with different dataset sizes as the initial training dataset to identify the effect of the dataset size and centre word prediction accuracy to monitor the model performance. Figure 5 shows the accuracy for the evaluation dataset and overall F1 score for the selected attack datasets. Accuracy for the evaluation dataset is estimated considering ID, which obtained the highest softmax probability. Since there are seven attack datasets, including the masquerade attacks, the overall F1 score is considered to make a fair comparison. Accordingly, this shows that centre ID prediction can be used to monitor the model progress and attack detection capability is improved with the dataset size.

3) All layers retraining and last layer retraining: Model retraining capability depends on the pre-trained model. Therefore, to evaluate the effect of the pre-trained model on the retraining process, we used four pre-trained models trained on

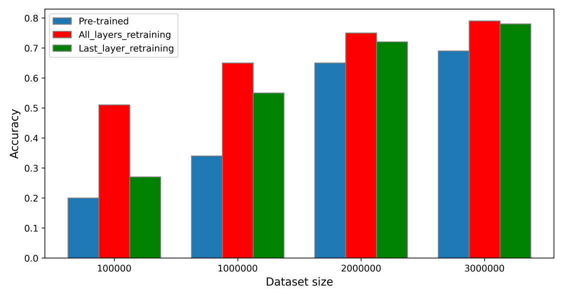


Fig. 6: Effect of pre-trained model on retraining process

four datasets. Then we retrained the models using an additional benign dataset (2000000 frames) as full model retraining and last layer retraining. Results are depicted in Figure 6. According to these results, a pre-trained model trained on a sufficiently large dataset is required to achieve good accuracy through retraining. With this initial dataset size, both full model training and last layer model retraining converge into approximately the same level of accuracy. Therefore, transfer learning can be successfully used to retrain the last layer with lower computational overhead to achieve the same accuracy level given a good pre-trained model. A sample of 3000000 CAN frames in ROAD dataset is approximately equivalent to 30 minutes drive dataset. This sample included a three benign datasets which covered different benign driving activities such as drive, break, accelerate and decelerate. Therefore, according to the obtained results, it is possible to improve the pre-trained model through transfer learning by using 30 minutes to one-hour drive dataset for the initial model training. However, this might depend on the number of CAN IDs on the vehicle and how frequently CAN frames are transmitted on the CAN bus. Additionally, it is important to include various benign driving activities in this dataset and ultimately, on-device transfer learning could improve this with a large set of streaming data.

Based on these results, we train our initial model with 3000000 CAN frames. A streaming dataset of 2000000 frames is used to retrain the last layer according to the algorithm 1 in Raspberry Pi. We compare our model (CAN-ODTL) detection with the baseline method of CAN-CID [8]. This uses a similar model architecture without having an attention layer. GRU layer uses 32 nodes and therefor, CAN-CID includes additional 6486 model parameters compared to CAN-ODTL. Additionally, we compare the performance with our pre-trained model (CAN-PreODTL). Table II shows the detection performance of CAN-ODTL compared to CAN-PreODTL and CAN-CID. To evaluate the detection performance, this paper use macro averaged F1-score (F1), false-positive rate (FP) and false-negative rate (FN). For the CAN-ODTL evaluation, experiments are run on MacBook M1 Pro by using the Raspberry Pi retrained model.

According to this, both CAN-CID and CAN-ODTL models achieved a higher detection rate for all the attacks. CAN-ODTL marginally outperforms CAN-CID for reverse light on and reverse light on masquerade attacks. Even though masquerade attacks do not inject malicious frames, they have changed CAN sequences due to the time synchronization mismatch. Therefore, CAN-ODTL is capable of detecting such sophisticated attacks. On the other hand, as expected CAN-PreODTL suffer from high false positives. This is due to the unseen benign sequences during the model training. On-device transfer learning helps to reduce false positives as it trains for large dataset. These results indicate the effectiveness of on-device transfer learning to retrain the CAN IDS.

TABLE II: Comparison of CAN-ODTL, CAN-PreODTL and baseline model detection performance of ROAD dataset

Attack	Model	F1	FP	FN
Fuzzing	CAN-CID	**100%**	0%	0%
	CAN-PreODTL	79.4%	18.5%	0%
	CAN-ODTL	**100%**	0%	0%
Max speedometer	CAN-CID	**100%**	0%	0%
	CAN-PreODTL	83.1%	21.3%	0%
	CAN-ODTL	**100%**	0%	0%
Max speedometer masquerade	CAN-CID	**100%**	0%	0%
	CAN-PreODTL	83.1%	21.3%	0%
	CAN-ODTL	**100%**	0%	0%
Reverse light on	CAN-CID	99.4%	0%	0.3%
	CAN-PreODTL	95.6%	6.9%	0%
	CAN-ODTL	**99.8%**	0.1%	0%
Reverse light on masquerade	CAN-CID	99.1%	0%	1.0%
	CAN-PreODTL	96.2%	6.6%	0%
	CAN-ODTL	**99.9%**	0%	0%
Reverse light off	CAN-CID	**100%**	0%	0%
	CAN-PreODTL	85.4%	17.0%	0%
	CAN-ODTL	**100%**	0%	0%
Reverse light off masquerade	CAN-CID	**100%**	0%	0%
	CAN-PreODTL	85.4%	17.0%	0%
	CAN-ODTL	**100%**	0%	0%

4) Overhead analysis: Detection latency and memory are also critical aspects of a CAN IDS. We compared the mean inference time of CAN-ODTL and the baseline model CAN-CID on a MacBook M1 Pro. CAN-CID takes 0.1ms for the inference, whereas CAN-ODTL takes 0.08ms. This inference time improvement is due to fewer model parameters than the CAN-CID. Therefore, CAN-ODTL achieves the same or improved detection level with a lower detection latency than the baseline model. During the transfer learning on Raspberry Pi, we monitor resource consumption. On average, it takes 0.16s to retrain the classification layer for one batch, which consists of 256 sequence windows. Thus, it takes approximately 0.625ms to retrain one input window. Retraining can process around 1600 CAN frames per second, whereas CAN transmits around 2500 frames per second. This demands having a buffer with at least 900 CAN frames. Generally, it takes around 75KB of memory to keep 900 CAN frames. Given the 8GB of memory, this can be easily handled. During the model retraining, on average, it utilized 45% CPU and 157MB of RAM. For the inference, the Tensorflow model is converted into the TFLite version with three options: non-optimized default quantization, dynamic range quantization and float 16 quantization.

Model sizes and average inferences times are shown in Table III. Tensorflow to TFLite conversion significantly reduces the model size and inference time on Raspberry Pi. Default TFLite quantization model and float 16 quantization model output softmax probabilities are on par with the Tensorflow model outputs up to the fifth decimal point. Therefore, these two models achieve the same detection level as the Tensorflow model shown in Table II. However. The dynamic range quantization model shows a little drop in accuracy despite the smallest model size. By considering the model size and inference time on Raspberry Pi, the float 16 quantization model outperforms other variations along with higher intrusion

TABLE III: CAN ODTL model inference overhead on Raspberry Pi

Model	Model size (KB)	Inference time (ms)
Tensorflow	233	3
Default TFLite quantization	82	0.7
Dynamic range quantization	29	0.5
Float 16 quantization	49	0.5

detection capability. With compared to the Tensorflow model this is 78% model size reduction and 83% of inference time improvement. During the inference time, it utilizes an average of 38% CPU and 5MB of RAM on Raspberry Pi. It takes only 0.5ms for the one-frame prediction, including the data pre-processing. This is approximately similar to the CAN data transmission rate of the ROAD dataset. For a 100ms observation window, on average, this takes 125ms to give the prediction. Generally, average driver response time ranges from 0.7s to 1.5s [24]. Therefore, the 125ms detection time is minimal compared to the driver's response time. These experimental results on Raspberry Pi show the effectiveness of the on-device transfer learning process and the inference of the proposed method to detect intrusions in the CAN bus in near real-time. Raspberry Pi can be connected to the CAN bus through the OBD-II port as an additional ECU and monitor CAN traffic to detect a wide variety of attacks.

VI. Conclusion

CAN bus is vulnerable to cyberattacks that directly affect vehicle passengers' safety. One-class classification-based CAN IDSs, while effective, need a large dataset of benign frames to avoid unseen benign frames.

This paper proposes an on-device transfer learning technique to address this issue. On-device transfer learning on Raspberry Pi and quantized TFLite conversion shows the effectiveness of the proposed method to improve the detection capability incrementally and to achieve near real-time detection. It requires a pre-trained model trained on a dataset with different benign driving behaviours. Classification layer retraining on such a pre-trained model achieves approximately the same level of accuracy as full model retraining. Experiments on Raspberry Pi show that the proposed model is suitable for deploying in a real vehicle and detecting various attacks in near-real time. For future work, we plan to deploy this in a real vehicle and test it under real attacks on the proving ground of our industry partner.

References

[1] S. N. Narayanan, S. Mittal, and A. Joshi, "Obd_securealert: An anomaly detection system for vehicles," in *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*, 2016, pp. 1–6.

[2] Z. Bi, G. Xu, G. Xu, C. Wang, and S. Zhang, "Bit-level automotive controller area network message reverse framework based on linear regression," *Sensors*, vol. 22, no. 3, p. 981, 2022.

[3] Y. Lee, S. Woo *et al.*, "Can signal extinction-based dos attack on in-vehicle network," *Security and Communication Networks*, vol. 2022, 2022.

[4] C. Miller and C. Valasek, "Adventures in automotive networks and control units," *Def Con*, vol. 21, no. 260-264, pp. 15–31, 2013.

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

[6] S. Rajapaksha, H. Kalutarage, M. Al-Kadri, A. Petrovski, G. Madzudzo, and M. Cheah, "Ai-based intrusion detection systems for in-vehicle networks: A survey," vol. 55, no. 11, feb 2023. [Online]. Available: https://doi.org/10.1145/3570954

[7] M. Bresch and N. Salman, "Design and implementation of an intrusion detection system (ids) for in-vehicle networks," Master's thesis, 2017.

[8] S. Rajapaksha, H. Kalutarage, M. O. Al-Kadri, G. Madzudzo, and A. V. Petrovski, "Keep the moving vehicle secure: Context-aware intrusion detection system for in-vehicle can bus security," in *2022 14th International Conference on Cyber Conflict: Keep Moving!(CyCon)*, vol. 700. IEEE, 2022, pp. 309–330.

[9] H. K. Kalutarage, M. O. Al-Kadri, M. Cheah, and G. Madzudzo, "Context-aware anomaly detector for monitoring cyber attacks on automotive can bus," in *ACM Computer Science in Cars Symposium*, 2019, pp. 1–8.

[10] A. Tomlinson, J. Bryans, and S. A. Shaikh, "Using internal context to detect automotive controller area network attacks," *Computers & Electrical Engineering*, vol. 91, p. 107048, 2021.

[11] C. Miller and C. Valasek, "Can message injection: Og dynamite edition," *Tech. Rep.*, 2016.

[12] M. E. Verma, M. D. Iannacone, R. A. Bridges, S. C. Hollifield, B. Kay, and F. L. Combs, "Road: The real ornl automotive dynamometer controller area network intrusion detection dataset (with a comprehensive can ids dataset survey & guide)," *arXiv preprint arXiv:2012.14600*, 2020.

[13] K.-T. Cho and K. G. Shin, "Error handling of in-vehicle networks makes them vulnerable," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1044–1055.

[14] J. Dürrwang, M. Braun, R. Kriesten, and A. Pretschner, "Enhancement of automotive penetration testing with threat analyses results," *To appear in SAE International Journal of Transportation Cybersecurity and Privacy*, 2018.

[15] Z. Cai, A. Wang, W. Zhang, M. Gruffke, and H. Schweppe, "0-days & mitigations: roadways to exploit and secure connected bmw cars," *Black Hat USA*, vol. 2019, p. 39, 2019.

[16] V. K. Kukkala, S. V. Thiruloga, and S. Pasricha, "Indra: Intrusion detection using recurrent autoencoders in automotive embedded systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3698–3710, 2020.

[17] V. K. Kukkala, S. V. Thiruloga, and S. Pasricha, "Latte: Lstm self-attention based anomaly detection in embedded automotive platforms," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 5s, pp. 1–23, 2021.

[18] 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 *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE, 2020, pp. 1–6.

[19] T.-N. Hoang and D. Kim, "Detecting in-vehicle intrusion via semi-supervised learning-based convolutional adversarial autoencoders," *Vehicular Communications*, vol. 38, p. 100520, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214209622000675

[20] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[21] S. Yang, X. Yu, and Y. Zhou, "Lstm and gru neural network performance comparison study: Taking yelp review dataset as an example," in *2020 International workshop on electronic communication and artificial intelligence (IWECAI)*. IEEE, 2020, pp. 98–101.

[22] N. James, L.-Y. Ong, and M.-C. Leow, "Exploring distributed deep learning inference using raspberry pi spark cluster," *Future Internet*, vol. 14, no. 8, p. 220, 2022.

[23] S. I. Mirzadeh, M. Farajtabar, R. Pascanu, and H. Ghasemzadeh, "Understanding the role of training regimes in continual learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 7308–7320, 2020.

[24] P. Droździel, S. Tarkowski, I. Rybicka, and R. Wrona, "Drivers 'reaction time research in the conditions in the real traffic," *Open Engineering*, vol. 10, no. 1, pp. 35–47, 2020. [Online]. Available: https://doi.org/10.1515/eng-2020-0004