Real-Time Attack Detection in Modern Automobile Controller Area Networks

Edward Martin and Sujeet Shenoi(✉)

University of Tulsa, Tulsa, OK, USA
sujeet@utulsa.edu

Abstract. Modern automobiles have numerous sensors, actuators and electronic systems interconnected via internal sub-networks that are not designed with security in mind. This chapter describes a novel real-time system that employs long short-term memory networks to monitor automobile controller area networks, detect attacks and raise alerts. A repeatable design framework is employed to construct and train multiple long short-term memory networks to recognize normal controller area network message timing patterns. The framework lays out the computational resources as well as the data collection and preprocessing and long short-term memory network model development and training steps. Also, it enables new long short-term memory network models to be trained and updated for automobiles of different makes, models and years.

The attack detection system leverages a server-client configuration to monitor an automobile controller area network bus. The server is an inexpensive Raspberry Pi device connected directly to the automobile controller area network bus that captures, logs and transmits controller area network message traffic to a client via a Wi-Fi network. The client, a workstation located outside the automobile, provides the computational resources for real-time attack detection. Trained long short-term memory models executing on the client workstation analyze the received controller area network messages, identify attacks and send alerts via the Wi-Fi network. Experimental results using a 2010 Toyota Prius testbed and a fully-operational 2014 Toyota Prius automobile demonstrate the effectiveness of the real-time attack detection system.

Keywords: Automobiles · Controller Area Networks · Real-Time Attack Detection · Long Short-Term Memory Networks

1 Introduction

Modern automobiles incorporate numerous sensors, actuators and diverse electronic systems that are interconnected by sub-networks to provide safe, convenient and comfortable experiences to drivers and passengers. The principal internal sub-networks, High-Speed Controller Area Network (High-Speed CAN), Low-Speed Controller Area Network (Low-Speed CAN), Local Interconnect Network (LIN) and Media Oriented Systems Transport (MOST) network, support

Published by Springer Nature Switzerland AG 2024
J. Staggs and S. Shenoi (Eds.): ICCIP 2023, IFIP AICT 686, pp. 221–252, 2024.
https://doi.org/10.1007/978-3-031-49585-4_11

automobile functionality [20]. The CAN protocol is employed by all the interconnected automobile applications that provide safety, convenience and comfort [2,3].

Unfortunately, modern automobile networks are not designed with security in mind [4,18]. First, the four principal sub-networks are interconnected via an automobile gateway, which increases the attack surface significantly. The lack of network segmentation makes it possible to gain remote access to the MOST network via the telematics module, pivot to the High-Speed CAN and target critical automobile components such as engine control and brakes. Second, the CAN protocol does not employ message encryption and authentication. The lack of message encryption simplifies reverse engineering as well as message interception, modification and fabrication. The lack of message authentication enables a malicious actor with CAN access to inject harmful messages that interfere with or disable any critical automobile system while remaining undetected. Additionally, the message identifier priority feature enables a malicious actor to flood the High-Speed and Low-Speed CANs with high-priority messages to deny service to all the networked components.

The sorry state of automobile security persists as new safety, convenience and comfort components are installed in models every year – soon, autonomous driving systems will be the norm. Automobile manufacturers are reluctant to segment automobile networks and components for reasons of cost, complexity and practicality (mainly maintenance and repairs). The lack of CAN message encryption and authentication persists due to the cost of implementation and computational resources required by individual automobile components.

A feasible solution is to develop attack detection systems that are incorporated in automobiles as add-on components. The attack detection systems would scrutinize CAN messages in real time and report malicious and anomalous traffic to drivers. Eventually, these attack detection systems could inform attack mitigation systems. Real-time attack detection is imperative because there can be no attack mitigation without detection.

The CAN attack detection system described in this work employs long short-term memory (LSTM) networks [10] to monitor automobile CANs, detect attacks and raise alerts in real time. LSTM networks are leveraged because they can learn patterns with long sequences.

A repeatable design framework is presented for constructing and training multiple LSTM networks that learn normal CAN message timing patterns. The design framework lays out the computational resources as well as the data collection and preprocessing and LSTM model development and training steps. The framework enables new LSTM models to be trained and updated for automobiles of different makes, models and years.

Another key contribution is real-time attack detection. The attack detection system leverages a server-client configuration. The server is an inexpensive Raspberry Pi device connected directly to a monitored automobile CAN bus that captures, logs and transmits CAN message traffic via a Wi-Fi network to a client workstation located outside the automobile. The client workstation pro-

vides the computational resources needed for real-time attack detection. Trained LSTM models executing on the client workstation process the transmitted CAN messages, identify attacks and send alerts via the Wi-Fi network.

The attack detection system was evaluated using a 2010 Toyota Prius testbed and a fully-operational 2014 Toyota Prius automobile. An attack device was employed to inject random CAN message identifiers at random times. The attack detection results are very good – LSTM model sensitivity ranged from 0.864 to 1.000 and accuracy ranged from 0.980 to 1.000. Sensitivity and accuracy are the most important metrics because LSTM models must recognize normal traffic and detect as many attacks as possible with high accuracy.

2 Interconnected Automobile Network

An automobile network comprises multiple sub-networks with systems that support safety, convenience and comfort [20]. The interconnected sub-networks include the High-Speed CAN, Low-Speed CAN, LIN and MOST network:

- **High-Speed CAN:** A High-Speed CAN connects critical automobile electronic control units (ECUs) such as the drivetrain, power steering, transmission control, instrument cluster, revolutions per minute (RPM) management, engine control and braking systems. A modern automobile may have multiple High-Speed CANs. Because a High-Speed CAN is responsible for critical automobile functionality, it employs a high-speed version of the CAN protocol that operates at bit rates between 500 Kbps and 1 Mbps to support fast and reliable communications [20].
- **Low-Speed CAN:** A Low-Speed CAN connects convenience and comfort components such as ventilated seats, power windows, lights, heat and air conditioning, and door locks. A Low-Speed CAN employs a low-speed version of the CAN protocol that operates at bit rates in the hundreds of Kbps [20]. An On-Board Diagnostics (OBD-II) interface provides direct access to the High-Speed and Low-Speed CANs via a specialized device. The OBD-II interface, which is located by the steering wheel or instrument cluster, is used to obtain diagnostic information required for automobile service and repair.
- **LIN:** A LIN connects ECUs in a Low-Speed CAN to peripheral components such as lights and door locks. The LIN protocol complements the CAN protocol.
- **MOST Network:** A MOST network connects multimedia components such as an infotainment system and cellular, Bluetooth and Wi-Fi modules in a ring network topology [20]. Telematics service providers such as OnStar interact with a MOST network via its cellular module.

3 Attack Vectors, Vulnerabilities and Attacks

This section lists the attack vectors that target automobile CANs. Also, it describes CAN vulnerabilities and attacks.

3.1 Attack Vectors

A malicious actor would be interested in accessing a High-Speed CAN because it contains critical automobile components. Several attack vectors can be leveraged to target the High-Speed and Low-Speed CANs.

Figure 1 shows the attack vectors that can be leveraged to access High-Speed and Low-Speed CAN components (targets). The attack vectors include the High-Speed CAN, OBD-II interface, Low-Speed CAN, Wi-Fi module, Bluetooth module, cellular module and infotainment system:

- **High-Speed CAN:** A malicious actor can gain direct access to High-Speed CAN targets via a physical connection to the High-Speed CAN (position 1 in Fig. 1). Upon gaining access to the High-Speed CAN via the physical connection, the malicious actor can gain indirect access to Low-Speed CAN targets via the automobile gateway.
- **OBD-II Interface:** A malicious actor can gain direct access to High-Speed and Low-Speed CAN targets via a physical connection to the OBD-II interface (position 2). This is because the OBD-II interface connects directly to the High-Speed and Low-Speed CANs.
- **Low-Speed CAN:** A malicious actor can gain direct access to Low-Speed CAN targets via a physical connection to the Low-Speed CAN (position 3). Upon gaining access to the Low-Speed CAN via the physical connection, the malicious actor can gain indirect access to High-Speed CAN targets via the automobile gateway.
- **Wi-Fi Module:** A malicious actor can gain indirect access to High-Speed and Low-Speed CAN targets via a remote connection to the Wi-Fi module in the MOST network (position 4). This is because the MOST network connects to the High-Speed and Low-Speed CANs via the automobile gateway.
- **Bluetooth Module:** A malicious actor can gain indirect access to High-Speed and Low-Speed CAN targets via a remote connection to the Bluetooth module in the MOST network (position 5). This is because the MOST network connects to the High-Speed and Low-Speed CANs via the automobile gateway.
- **Cellular Module:** A malicious actor can gain indirect access to High-Speed and Low-Speed CAN targets via a remote connection to the cellular module in the MOST network (position 6). This is because the MOST network connects to the High-Speed and Low-Speed CANs via the automobile gateway.
- **Infotainment System:** A malicious actor can gain indirect access to High-Speed and Low-Speed CAN targets via a physical connection to the infotainment system in the MOST network (position 7). This is because the MOST network connects to the High-Speed and Low-Speed CANs via the automobile gateway.

3.2 Vulnerabilities

CAN vulnerabilities arise from the lack of message authentication and message encryption, message identifier priority feature and absence of network segmentation:

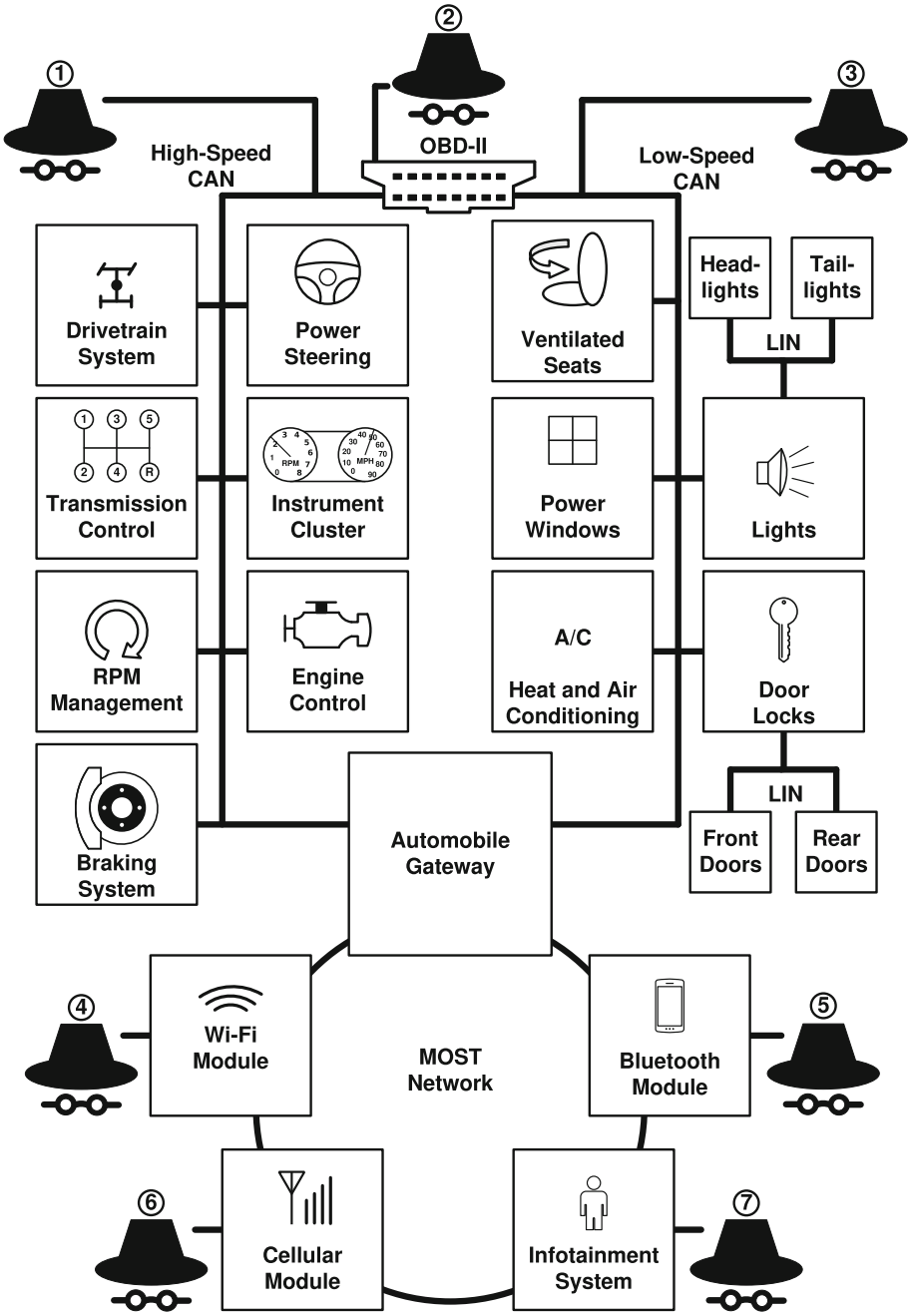

Fig. 1. High-Speed and Low-Speed CAN attack vectors.

- **Message Authentication:** The CAN protocol lacks message authentication. A CAN message only contains an identifier without source and destination addresses [14]. Therefore, receiving nodes cannot verify message source and distinguish between real and fake messages. Thus, a malicious actor with CAN access can transmit fake messages without being detected.
- **Message Encryption:** The CAN protocol lacks message encryption. All CAN messages are transmitted in plaintext [14]. Connecting to a CAN bus directly or via its OBD-II interface provides direct access to all CAN messages. The lack of message encryption simplifies reverse engineering as well as message interception, modification and fabrication. Although automobile manufacturers keep CAN message content proprietary, thorough analysis of CAN traffic can reveal message details. The lack of encryption also enables CAN message replay.
- **Message Identifier Prioritization:** A message identifier with a low binary value has high priority on a CAN bus. A malicious actor can flood a CAN bus with high priority messages to prevent the transmission of legitimate messages. Such denial-of-service attacks are easy to execute and can render critical automobile systems non-operational [14].
- **Network Segmentation:** CANs are not segmented adequately. Nodes in different CANs can communicate with each other via the automobile gateway. A malicious actor with access to an automobile gateway can target nodes in all the connected CANs [14].

3.3 Attacks

Numerous attacks have been demonstrated on CANs in modern automobiles. Absent custom security measures, these attacks are expected to impact practically every High-Speed and Low-Speed CAN.

Hoppe and Dittman [11] describe novel attacks on simulated CANs. They employed CANoe software to create a virtual network comprising connected High-Speed and Low-Speed CAN buses. The CAN buses were connected to a virtual automobile power window system. In one experiment, they captured CAN message frames on the CAN buses and recorded the message frames that opened and closed the power window; the recorded frames were subsequently replayed to control the power window. In another experiment, they used an ECU in a Low-Speed CAN bus to obtain information from the High-Speed CAN bus, demonstrating the lack of segmentation in the virtual CANs. These experiments stimulated research in automobile security.

Koscher et al. [13] employed a custom CarShark CAN bus analyzer and packet injection tool to perform experiments with stationary and moving automobiles. CarShark was used to sniff CAN frames and the message identifiers were subsequently reverse engineered via fuzzing techniques. CAN messages were then injected to control the radio, instrument cluster, engine components, brakes, heating, ventilation and air conditioning, and body control module functions. Denial-of-service attacks were successfully executed on the engine control module of a stationary automobile. Several attacks were executed on a moving

automobile, including sounding the horn, killing the engine and preventing the automobile from restarting, and disabling the brakes. The experiments demonstrated that a malicious actor with physical CAN bus access could wreak havoc on stationary and moving automobiles.

Hoppe et al. [12] describe experiments involving a CAN testbed with an electric window lift, instrument cluster, automobile gateway, warning lights and airbag control system. They used a laptop connected to the testbed via the OBD-II interface to directly access the CAN. Fabricated messages were transmitted from the laptop to tamper with various CAN systems.

Checkoway et al. [6] demonstrated several remote exploits that leveraged automobile mechanic tools, media interfaces and wireless communications. In particular, they used the OBD-II interface and infotainment system to obtain indirect physical network access, and Bluetooth and cellular channels to access automobile systems. The research demonstrated that the attack surface expands considerably as an automobile becomes more connected.

Valasek and Miller [24] leveraged custom tools to interact with automobile networks in a 2010 Ford Escape and 2010 Toyota Prius. The automobiles were targeted by connecting a laptop to the OBD-II interfaces. CAN messages were captured and replayed. Also, messages were modified and injected to control the behavior of the automobiles.

Valasek and Miller [25] also demonstrated physical and remote attacks on a 2014 Jeep Cherokee. They targeted its Harman Kardon Uconnect infotainment system that bundles Wi-Fi connectivity, navigation, apps and cellular communications. Specifically, they gained physical access to the infotainment system via a USB connection and were able to jailbreak the system. Next, they gained remote access to the telematics system and exploited it by leveraging an open diagnostics port in the CAN. Using the diagnostics port access, they uploaded modified firmware to the microcontroller connected to the CAN. The resulting direct access to the High-Speed and Low-Speed CAN buses enabled them to send commands to several critical systems. A viral video [8] shows Valasek and Miller remotely turning on the air conditioner of a moving Jeep Cherokee, activating wiper fluid release and even disabling the brakes. The research of Valasek and Miller led Fiat Chrysler to recall 1.4 million automobiles in 2015 [16].

4 Related Work

This section discusses LSTM networks and their applications to CAN attack and anomaly detection.

4.1 LSTM Networks

Creating an anomaly-based detection model for automobile CANs requires a neural network that can learn normal network traffic patterns. This research employs a type of recurrent neural network called a long short-term memory (LSTM) network.

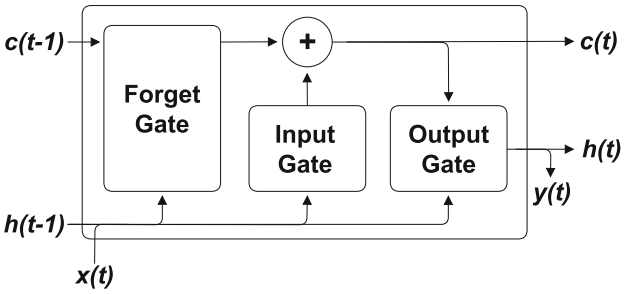

Fig. 2. LSTM cell.

A recurrent neural network comprises connected artificial neurons called cells. Each cell processes inputs using a mathematical function and produces outputs that are sent as inputs to other connected cells. Learning is built into the recurrent neural network cell in the form of a feedback loop [5]. The loop enables information to be passed to the next iteration (time step) of the network loop.

Unfortunately, traditional recurrent neural networks can only learn short patterns. As data propagates through a recurrent neural network, information about older data is discarded. This is problematic when attempting to construct a model that attempts to learn long sequences [7]. Traditional recurrent neural networks have trouble learning patterns in applications such as speech recognition, language translation and network traffic analysis.

Hochreiter and Schmidhuber [10] introduced LSTM networks to address the long-term memory limitation of traditional recurrent neural networks. Their design uses gates within cells that allow certain data to flow through cells. Each cell also has a long-term state called the cell state and a short-term state called the hidden state. These innovations enable an LSTM network to learn patterns with long sequences.

Figure 2 shows a single LSTM cell. All the cell inputs and outputs are vectors. At time step t, input $x(t)$ and the previous hidden state $h(t-1)$ are fed to the cell, which outputs $y(t)$ and the updated hidden state $h(t)$. The horizontal line on the top of the cell (running between $c(t-1)$ and $c(t)$) denotes the cell state, which gives the LSTM cell its long-term memory capability. In an LSTM network implementation, the cell state runs through an entire chain of cells and is updated as needed. Information is added to and removed from the cell state using gates. A cell has three types of gates, forget gate, input gate and output gate:

- **Forget Gate:** A forget gate decides how much information in inputs $x(t)$ and $h(t-1)$ is discarded from the cell state. The output updates the cell state.
- **Input Gate:** An input gate decides how much information from inputs $x(t)$ and $h(t-1)$ is added to the cell state. The output is added to the cell state.

– **Output Gate:** The output gate decides how much information should be read from the cell inputs and cell state, and produces the cell outputs $h(t)$ and $y(t)$. The output yields the cell outputs $h(t)$ and $y(t)$.

The cell state enables an LSTM cell to store important inputs for a period of time. The forget gate in an LSTM cell determines how much information should be discarded. The input gate enables an LSTM cell to learn the important inputs. The output gate produces the LSTM outputs at a specific time step.

An LSTM cell is the fundamental building block of an LSTM network. Each layer in an LSTM network contains tens to hundreds of LSTM cells. The number of layers in an LSTM network depends on the complexity of the problem.

4.2 Attack and Anomaly Detection

Several researchers have employed LSTM networks to detect attacks and anomalies in automobile CANs. The approaches differ in their LSTM model architectures and features used to detect attacks and anomalies.

Taylor et al. [23] were the first to employ an LSTM model to detect attacks leveraging CAN message data field values. An LSTM model was trained to predict data in the next message corresponding to a given CAN identifier. The trained model recognized fabricated CAN messages as anomalous and indicators of attacks.

Zhu et al. [28] developed a multi-dimensional LSTM model for detecting anomalies in CANs. They combined the values in the CAN timestamp and data fields to produce samples with a single feature that were used to train an LSTM model to identify anomalous CAN traffic. The trained model was positioned in a mobile edge computing server to collect and analyze CAN traffic.

Xiao et al. [26] developed a convolutional LSTM model to examine spatiotemporal relationships in CAN traffic. The model was trained using samples with CAN timestamp and data field values. Correlation coefficients between predicted data and real data were computed. A specific range of correlation coefficient values indicated anomalous behavior. Tariq et al. [22] also developed a convolutional LSTM model for predicting normal and abnormal CAN message sequences. Their model is similar to the model of Xiao et al. [26], but it focused on transfer learning, which enabled it to detect novel attacks based on knowledge about previously-seen attacks.

Yang et al. [27] developed an LSTM model to learn the fingerprints of analog CAN signals emanating from ECUs. Their approach differed from others in that it examined CAN message fields. The LSTM model, which was trained using analog CAN signals from ECUs as they processed messages, was implemented using field-programmable gate array hardware for real-time detection.

Hanselmann et al. [9] developed an unsupervised anomaly-based intrusion detection system using LSTM models. An LSTM model was assigned to each CAN identifier and each model was trained to learn the temporal features associated with its CAN identifier. The outputs of all the LSTM models were input to an autoencoder, which produced an anomaly score.

Sun et al. [21] developed a convolutional LSTM model with attention for anomaly-based detection in CANs. The model incorporated a one-dimensional convolution layer for feature extraction and a bidirectional LSTM layer for time characteristic learning in the forward and backward directions. The model was trained using analog CAN signals instead of CAN message fields. Anomaly-based detection was tested on a scaled-down CAN with three ECUs.

Aldhyani and Alkahtani [1] developed a convolutional LSTM model for classifying attacks on automobile CANs. The model, which was trained using CAN message timestamp, identifier, data length and data fields, classified CAN traffic as spoofing, flooding, replay or benign.

Several research efforts have trained LSTM models to learn traffic sequence patterns using the CAN message identifier and data fields, but not the timestamp field. Four efforts stand out as exceptions. Hanselmann et al. [9] focused on learning temporal features of CAN identifiers, but they developed an LSTM model for each CAN identifier, which resulted in a large system. Xiao et al. [26] considered the CAN message timestamp field in addition to other fields, but little information is provided about their implementation. Zhu et al. [28] combined one-bit timestamp and 64-bit data fields. Aldhyani and Alkahtani [1] also combined the timestamp with other features.

Most of the related research efforts did not focus on live systems that monitored CANs in operating automobiles. Three efforts are exceptions. Yang et al. [27] implemented their model using field-programmable gate array hardware for real-time detection. Zhu et al. [28] proposed a mobile-edge computing architecture for real-time detection. However, neither Yang et al. nor Zhu et al. tested their systems on live CAN traffic. The work of Sun et al. [21] stands out because their model was tested on an automobile CAN, although it incorporated only three ECUs.

The anomaly-based CAN attack detection approach described in this work advances previous research by focusing on message timing in live automobile CAN traffic. An unsupervised machine learning framework is employed to train LSTM models to recognize normal CAN traffic patterns based on the timestamp and identifier fields. The resulting LSTM models identify mistimed CAN messages as attack indicators. Other unique features of the approach are real-time attack detection in operating automobiles and adaptability to multiple automobile CANs.

5 Attack Detection Design Framework

The attack detection design framework covers the five steps in the machine learning workflow: data collection, data preprocessing, model development and training, model testing, and model enhancement and deployment.

5.1 Data Collection

Tens of thousands of CAN data samples are required to develop LSTM networks for detecting attacks in automobile CANs. During the research, CAN data was

Timestamp	Interface	Message ID#Data
(1645210861.287672)	can0	127#0000000000200050
(1645210861.299934)	can0	245#000000004C
(1645210861.304063)	can0	127#0000000000200050
(1645210861.304252)	can0	247#0200FF0000
(1645210861.304496)	can0	3F9#553C5601000000EC
(1645210861.309178)	can0	4A8#0000004000400000
(1645210861.310218)	can0	499#0000000000000000
(1645210861.311234)	can0	49A#0000000000000000
(1645210861.312228)	can0	49B#00A0006010BE0000
(1645210861.313258)	can0	49D#6666005EBD0238C0
(1645210861.316763)	can0	45C#5F02002007000000
(1645210861.320444)	can0	127#0000000000200050
(1645210861.323457)	can0	245#000000004C

Fig. 3. Sample CAN log file.

collected by connecting a Raspberry Pi with a PiCAN interface board to the monitored CAN bus and logging the data.

The Raspberry Pi device captured CAN traffic logs. The `can-utils` package, specifically its `candump` utility [20], facilitated the logging of CAN messages. The Raspberry Pi was set up with the `can0` SocketCAN interface. SocketCAN provides CAN drivers as network devices in a Linux operating system [20]. Application access to the CAN bus was enabled by a network socket programming interface. The `can0` interface provided direct access to the connected automobile CAN bus.

The CAN log files were saved to Google Drive for data preprocessing. Figure 3 shows a sample CAN log file. Each row depicts a single message broadcasted on the connected CAN bus. The first column lists the timestamps, the absolute times connected to the Raspberry Pi system clock. The second column lists the `can0` interface. The third column specifies the CAN message identifiers and associated data. The CAN message identifier is the hexadecimal string to the left of the hash symbol and the data field is the hexadecimal string to the right of the hash symbol. The data field size is between one and eight bytes.

5.2 Data Preprocessing

CAN log data must be preprocessed before it can be used for training and testing. The CAN message timestamp and identifier were selected as features for developing LSTM network models. The feature values are shown in the first column and the left half of the third column in Fig. 3. The CAN data field was not used because attack detection focuses on the timing patterns of CAN messages.

The `pandas` Python library was primarily used for data preprocessing [17]. The library uses high-level data structures called DataFrames and various methods to simplify data conversion. A DataFrame is a tabular data structure with an ordered collection of columns, potentially with different types. Several built-in methods were employed to manipulate DataFrames during data preprocessing.

The CAN log file was read and the feature data was stored in a `pandas` DataFrame using the `read_csv` method. Only the CAN message timestamp and identifier features were considered in this research. The timestamp feature was converted from a string to numeric value using the `to_numeric` method.

The LSTM networks were trained using Δtimestamp values corresponding to CAN identifiers. A Δtimestamp value denotes the frequency at which a CAN identifier is transmitted on a CAN bus. The composite `groupby(df.ID).diff` method computes the difference between the current and previous timestamps of a given CAN identifier. The difference value replaces the timestamp in the timestamp column in the DataFrame. Because there is no previous timestamp before the first row of a respective CAN identifier, null values are stored in the first differenced rows. The `dropna` method eliminates rows with null values from the DataFrame.

After the data is loaded in a DataFrame, groups of CAN messages must be binned into smaller DataFrames based on the frequency of CAN identifiers. This is necessary because of potential biasing in a machine learning model. Specifically, CAN message identifiers that appear more frequently in a log file induce bias during model training. Biasing causes a model to make incorrect assumptions, which hinders effective machine learning [7].

The k-means clustering algorithm [15] was used to place CAN messages into bins based on their frequency. The algorithm was selected because it quickly and efficiently clusters datasets [7]. The k-means algorithm clustered the data samples into bins based on the timestamp feature in a loaded DataFrame. A separate DataFrame was subsequently created for each bin.

Figure 4 shows an example of the binning process. A DataFrame is accepted as input to the k-means clustering algorithm. Three bins are generated as outputs by the algorithm. The highest frequency messages are stored in Bin 0, medium frequency messages in Bin 1 and lowest frequency messages in Bin 2. Each bin is treated separately throughout the rest of the design process and a separate LSTM model is trained using each bin.

Feature data is required to be in a numeric format [5]. However, the CAN identifier feature values are hexadecimal strings that correspond to categorical data (i.e., labels that describe their semantics). Therefore, the CAN identifiers were converted to numeric values using an integer encoding that gives a unique integer value to each CAN identifier. Conversions of timestamp feature values were not required because timestamps have a numeric format.

Feature data must be scaled [5]. This was accomplished by normalizing the numeric CAN identifier feature values between zero and one. The timestamp feature values were rescaled so that the mean of the values was zero and standard deviation was one. This was done because the timestamp feature values had a well-behaved mean and standard deviation. The CAN identifier feature values did not have this property, which is why they were normalized.

The data conversion process invokes the `LabelEncoder`, `MinMaxScaler` and `StandardScaler` methods in the Scikit-Learn library [7]. The input is the DataFrame with the CAN timestamp and identifier features. The CAN identifier

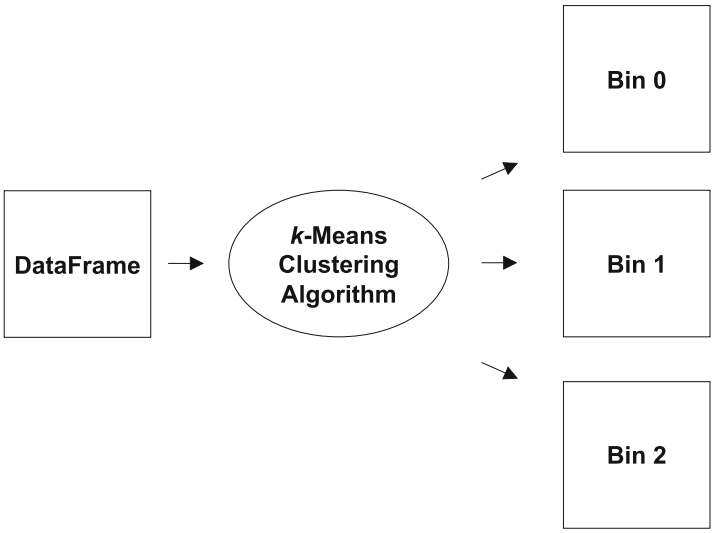

Fig. 4. Binning process.

feature values are encoded as integers and subsequently normalized. Finally, the CAN timestamp feature values are standardized to produce the converted DataFrame.

Note that the encoder objects created during data conversion must be saved for future use. This is because all data input to the attack detection system must be encoded and scaled in the same format.

LSTM networks require three-dimensional data [7]. Therefore, the final data preprocessing step converts the data to a batch of sequences in three dimensions, batch size, time step and input dimensionality:

- **Batch Size:** The batch size is the number of CAN message sequences input to an LSTM network. When training a machine learning model, the batch size is of the order of tens to hundreds of thousands of sequences.
- **Time Step Size:** The time step is the window size used to train an LSTM network. It represents the memory of an LSTM network – given n time steps, the LSTM network is trained to remember the previous n observations.
- **Input Dimensionality:** The input dimensionality is the number of features used to train an LSTM network. Two features, CAN message timestamp and identifier, are employed to develop the LSTM-based attack detection system in this dissertation research.

Figure 5 shows an example of the sequence creation process. In the example, the time step is five, number of samples is seven and number of features is two. The process employs a sliding window approach, where the time step window moves across the rows of the dataset. Sequence 0 takes the first five samples shown in gray. Sequence 1 moves down one row and takes the next five samples.

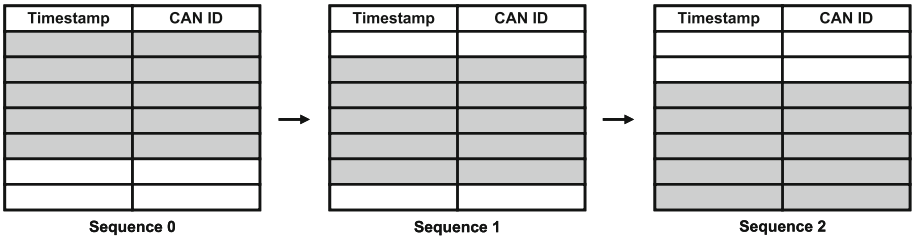

Fig. 5. Sequence creation process.

Sequence 2 moves down another row and takes the next five samples. The input dimensionality in this example is $3 \times 5 \times 2$, which is fed to an LSTM network.

5.3 Model Development and Training

The CAN network attack detection model employs LSTM networks with autoencoder architectures. The LSTM network layers capture temporal relationships between CAN message timestamp and identifier features. The autoencoder architectures capture dense representations of the training data.

An autoencoder learns to copy inputs to outputs using data compression and reconstruction [7]. Specifically, it uses data compression and reconstruction to learn the important characteristics of data. Given CAN data, an autoencoder learns normal CAN traffic patterns.

An autoencoder incorporates an encoder and decoder. The encoder compresses the input data to a fixed-sized vector with less dimensionality. The decoder reconstructs the data from the fixed-size vector of less dimensionality to produce an output. A well-trained autoencoder produces output values close to its input values. However, due to network constraints and data complexity, training an autoencoder is not a trivial task. The autoencoder must learn efficient ways to represent the training data in order to have its outputs resemble its inputs.

Figure 6 shows an LSTM network with an autoencoder architecture. The architecture has two LSTM layers, a repeat vector layer and a time distributed layer.

The first LSTM layer is the encoder. This layer accepts the input with a time step of 15 and input dimensionality of two. It is designed for an arbitrary batch size as input, so the batch size is labeled none. The layer outputs a compressed feature vector of size 1×30.

The repeat vector layer serves as the bridge between the encoder and decoder [19]. The layer accepts a feature vector of size 1×30 as input and replicates it 15 times. It outputs a 15×30 vector that is input to the second LSTM layer.

The second LSTM layer is the decoder layer, which reconstructs the encoding [19]. The layer accepts a 15×30 vector as input and outputs a reconstruction of the encoding with size 15×30.

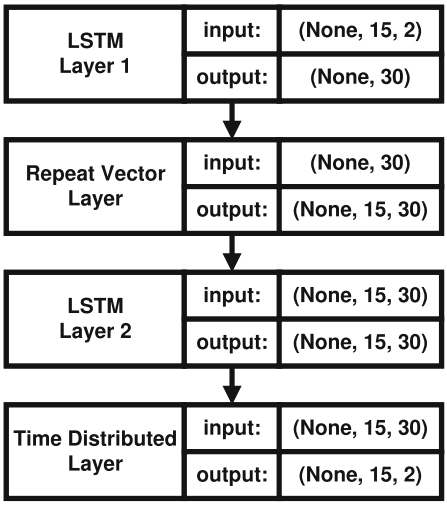

Fig. 6. LSTM network with an autoencoder architecture.

The time distributed layer converts the reconstructed output to the same size as the input. It creates a 30×2 output with input dimensionality of two. The output of the second LSTM layer (15×30) is matrix-multiplied with the output of the time distributed layer (30×2). The resulting output size of 15×2 is the same as the input size.

After an LSTM network is constructed, the model must be compiled and fitted. The compilation process, which is a pre-fitting step for the model, converts the sequences of layers in Fig. 6 to matrix transforms that can be executed on central processing units (CPUs), graphics processing units (GPUs) or tensor processing units (TPUs) [5].

A constructed LSTM model is compiled using the optimizer and loss function parameters. The optimizer is an algorithm that updates the model weights during training. Adaptive moment estimation, a popular optimization algorithm due to its overall performance [5], was employed. Additionally, the algorithm does not require constant tuning.

The loss function determines the performance of an LSTM model. Since a model predicts numerical values based on given input, it attempts to solve a regression problem. The mean-squared error (MSE) loss function employed is given by:

$$\text{MSE loss} = \sum_{i=1}^{n} \frac{(x_i - y_i)^2}{n}$$

where x_i is the model input, y_i is the predicted model output and n is the number of data samples over which the loss is computed.

An LSTM model is fitted after the compilation step. This is the step where the LSTM model is trained. To solve the regression problem, training data is fed

to the model and the weights are adjusted to make the model fit the training data to the desired extent [5].

The following command from the Keras ML library [7] fits a constructed LSTM model:

```
model.fit(x, y, epochs=20, batch_size=15, validation_split=0.2)
```

where `x` is the preprocessed input training data and `y` is the expected output. This is equivalent to an autoencoder architecture because the model is trained to make the outputs close to or equal to the inputs.

An epoch corresponds to one pass through the entire training dataset. The batch size specifies how many samples the model processes before the weights are updated.

The validation split holds out a portion of the training data to create the validation dataset. The `model.fit` command holds out 20% of the training data for validation to improve the problem generalization ability of the LSTM model [7]. The validation dataset is used during the training phase to tune the model.

Model learning curves created during training provide visual indications of how the LSTM model is learning [5]. The learning curves plot the MSE losses with the training and validation datasets during model training. Most well-trained models have curves with exponential decays. Training must be terminated when the training and validation curves taper off because overtraining can cause problems. It is also important to repeat the training process multiple times to verify that the model works well. Neural networks are stochastic, meaning that different predictions are made when the same model is trained on the same data over multiple runs [5].

Data pertaining to a trained model is saved in two files for use in the final two steps of the machine learning workflow, model testing, and model enhancement and deployment. The model architecture is saved in a JSON file. The model weights are saved in an HDF5-formatted file.

5.4 Model Testing

A trained model is tested and verified using unseen CAN data. Data in the testing dataset is preprocessed in the exact same way as the training data in order to enhance accuracy.

Table 1 shows the files created for model testing that are subsequently updated after model enhancement. The model architecture and weights are loaded from the saved JSON and HDF5 files. The encodings as well as the standardizing and normalizing parameters are loaded from the saved PKL files.

The maximum MSE loss during testing is computed as follows. Each test data sequence passes through the LSTM model and an output sequence is generated. The MSE loss is computed between the input and output sequences. The maximum MSE loss is output at the end of the loop. A model with good generalization ability has a low maximum MSE loss, similar to the MSE loss of the training data.

Table 1. Files created for model testing and updated after model enhancement.

File	Description
`model.json`	Stores the model architecture
`weights.h5`	Stores the model weights
`standardizeScaler.pkl`	Stores the standardizing parameters for the timestamp feature
`labelEncoder.pkl`	Stores the integer encodings of the CAN identifier feature
`normalizeScaler.pkl`	Stores the normalizing parameters for the CAN identifier feature

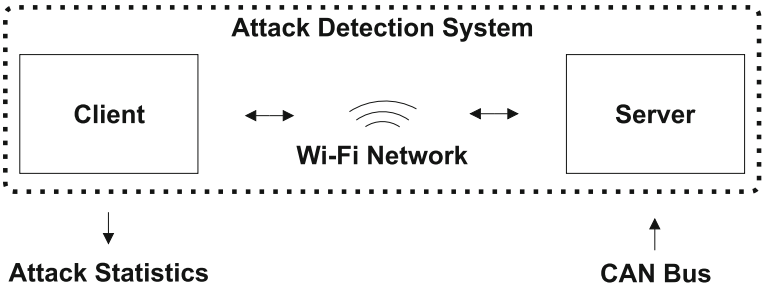

Fig. 7. Attack detection system configuration.

5.5 Model Enhancement and Deployment

A trained and tested LSTM model can always be enhanced. Model enhancement starts with the LSTM model files produced after testing and proceeds to update the LSTM model files for deployment in a production environment. Model enhancement and deployment require updated versions of the files used for testing (Table 1).

6 Real-Time Attack Detection System

This section describes the deployment of the LSTM models in the real-time CAN attack detection system.

6.1 System Configuration

The attack detection system leverages a server-client configuration on a monitored automobile CAN bus. This configuration eliminates the need to have a large system connected to the CAN bus and supports remote attack detection. The attack detection system uses commercial off-the-shelf components.

Figure 7 shows the attack detection system configuration. The server is connected to the monitored CAN bus. The server and client are connected via a

Wi-Fi network. The client processes CAN data received from the server and presents attack statistics.

Server Configuration. The server is an embedded system that is designed to connect to an automobile CAN bus. The embedded hardware was chosen for its lightweight, powerful computing and low cost features.

The server executes on a Raspberry Pi 4 Model B running the 64-bit Raspberry Pi OS Lite. The Raspberry Pi incorporates a PiCAN3 board, which contains an MCP2562 CAN transceiver, MCP2515 CAN controller and 3 A switch mode power supply board. The Raspberry Pi is mounted inside the automobile to be monitored for CAN attacks.

Client Configuration. The client is a rack-mounted computer workstation located outside the automobile being monitored. The workstation was selected to provide the computational resources and processing speed needed for real-time attack detection. The client executes on a Macintosh Pro Rack version 2019 running an Ubuntu virtual machine. The virtual machine is used for all client command and control.

6.2 System Processes

Real-time attack detection involves CAN message logging, processing and prediction. Python scripts were written for the server and client to implement the necessary tasks.

Server Process. The server is responsible for receiving CAN messages, storing the messages in queues and handling client message requests. CAN messages are split into queues depending on the number of LSTM models trained for the CAN bus. A queue is reserved for each LSTM model (bin) created during the model training phase. The server transmits CAN messages from relevant queues upon requests from the client. The server process includes the CAN message handling subprocess and client handling subprocess.

The CAN message handling subprocess determines the number of bins and the CAN messages that go in the bins. Each bin corresponding to an LSTM model contains the CAN identifiers associated with a queue. The server listens for messages on the CAN bus. When a CAN message is received, the difference between the timestamp of the current message and previous message with the same CAN identifier is computed. Following this, the CAN message is placed in the appropriate queue.

The client handling subprocess handles the queues generated by the CAN message handling subprocess. The server listens for a message request from the client. When a request is received from the client, the server parses the request to determine the number of requested messages and the queue in which they exist. If the number of messages requested is greater than the number of messages

in the queue, the server waits for the queue to hold the requested number of messages. The server then sends the requested messages to the client.

The CAN message and client handling subprocesses operate concurrently. This ensures that no CAN messages or message requests are dropped. The queues contain data that is shared by the CAN message handling and client handling subprocesses.

Client Process. The client is responsible for examining the received CAN messages and detecting attacks. First, the client requests a set of messages from the server. After receiving the messages, the client preprocesses the messages and feeds them to the appropriate LSTM model. The LSTM model compresses and reconstructs the messages, and computes the mean-squared error. The client process includes the client message requesting and message prediction subprocesses.

An attack detection system user specifies the number of iterations of requests at client startup. The client message requesting subprocess sends a request to the server that includes the number of messages requested and LSTM model (bin number) for processing the messages. The number of requested messages is equal to the time step in the corresponding LSTM model.

Messages received from the server are encoded and scaled for input to the LSTM model. This is an important step because all the messages have to be encoded and scaled in exactly the same manner as the training data. The encoding and scaling files saved after LSTM model training are processed to produce tensors of transformed data. A tensor is a three-dimensional vector of size batch size $\times$ time step $\times$ input dimensionality. The batch size is one, the time step is dependent on the LSTM model and the input dimensionality is two (CAN timestamp and identifier features). The tensor is placed in a shared data queue for the client message prediction subprocess.

An attack detection system user may specify the number of iterations required for attack prediction at client startup. Alternatively, the client message prediction subprocess can run continuously. The client message prediction subprocess receives a tensor of messages from the shared queue. The tensor is passed to the appropriate LSTM model and returns a reconstructed tensor. Finally, the client message prediction subprocess computes the MSE loss between the input and output tensors.

The client message requesting and prediction subprocesses execute concurrently to support real-time attack detection. The tensor queue is shared by the two subprocesses.

6.3 Client Operation Modes

While the server has a single operation mode, the client has three modes, threshold testing mode, attack detection mode and default execution mode:

– **Threshold Testing Mode:** This mode enables an attack detection system user to determine the appropriate MSE loss threshold for detecting attacks.

The client executes thousands of iterations. Upon completing the iterations, statistics are printed for a user to determine an appropriate MSE loss threshold for attack detection.
- **Attack Detection Mode:** This mode is used for attack detection statistics generation after threshold testing. An attack is indicated when the MSE loss value is greater than the set threshold. The attack detection statistics include the true-positive, false-positive, true-negative and false-negative error rates.
- **Default Execution Mode:** This mode alerts an attack detection system user to attacks. It does not print attack detection statistics.

Multiple client processes must execute concurrently to analyze all the messages transmitted on a CAN bus. A separate client process executes for each LSTM model used in attack detection. A bash script initiates a client process for each model. The client processes leverage the central processing unit cores on the Macintosh Pro workstation to run the LSTM models concurrently.

7 Experimental Testbeds and Results

This section describes the experimental testbeds and the results of evaluating the performance of the real-time attack detection system.

7.1 Experimental Testbeds

Two experimental testbeds were employed to evaluate the performance of the real-time attack detection system for CANs, a 2010 Toyota Prius testbed and a fully-operational 2014 Toyota Prius automobile.

2010 Toyota Prius Testbed. Figure 8 shows the 2010 Toyota Prius testbed. The testbed comprises a CAN test bench with ECUs and a Raspberry Pi attack device (mounted on the wooden board on the table), a black Raspberry Pi server for attack detection (just to the left of the test bench on the table) and a Macintosh Pro Rack version 2019 client running an Ubuntu 22.04 LTS virtual machine using VMware Fusion 12 Pro (on the floor).

The test bench comprises a CAN bus connecting four ECUs from a wrecked 2010 Toyota Prius. The ECUs include the smart key, transmission control, power management control and instrument cluster modules. The accelerator pedal and gear shift mechanism are connected to the power management control and transmission control modules. The Raspberry Pi on the bottom-left of the testbed is the attack device. The black Raspberry Pi just to the top-left of the test bench is the attack detection server. The Raspberry Pi attack device and the Raspberry Pi attack detection server are connected directly to the High-Speed CAN bus in the testbed.

Table 2 specifies the 2010 Toyota Prius test bench LSTM model architecture. The architecture comprises four LSTM models trained to analyze the message

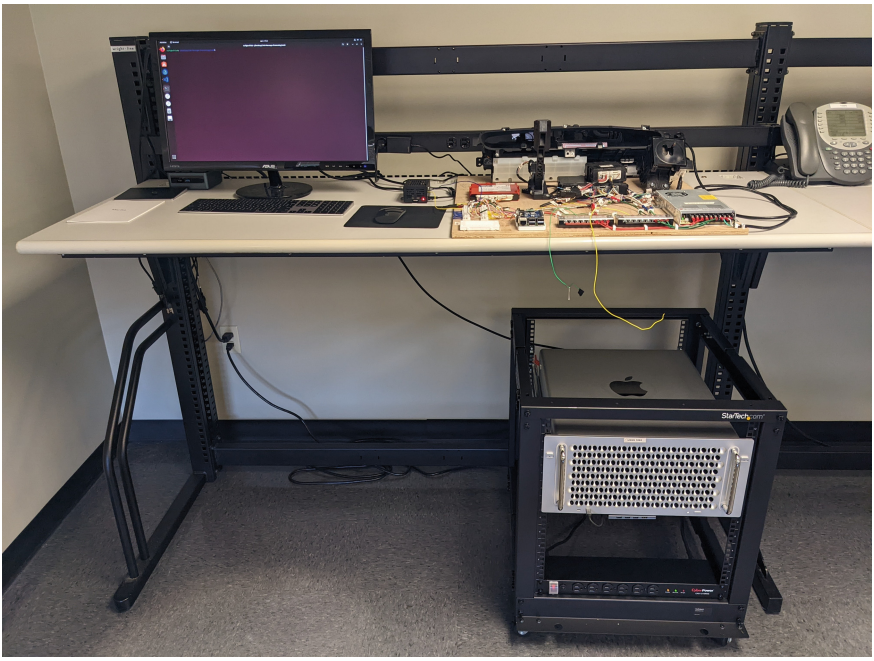

Fig. 8. 2010 Toyota Prius testbed.

timing patterns of the 28 CAN message identifiers listed in the table. Four models were constructed for the 28 CAN message identifiers to prevent model bias towards higher frequency identifiers. CAN message identifiers 612, 613, 616, 619 and 61A were not included due to their low transmission frequencies.

The time step specifies the memory window used by an LSTM model. The LSTM cell configuration corresponding to each model specifies the number of cells used in the LSTM encoder and decoder layers. Each LSTM model has a repeat vector layer between the LSTM encoder and decoder layers, and a time-distributed layer positioned after the LSTM decoder layer. The epochs and batch sizes used for training the LSTM models are also listed.

2014 Toyota Prius Automobile. A fully-operational 2014 Toyota Prius automobile with all the connected ECUs was also employed in the experimental evaluation. Figure 9 shows the Raspberry Pi attack device located below the steering column (left) and the Raspberry Pi server for attack detection located on the center console (right). The attack device is connected directly to the High-Speed CAN bus via the OBD-II diagnostics interface whereas the server is connected to the High-Speed CAN bus via the twisted pair. The Macintosh Pro Rack version 2019 in the 2010 Toyota Prius testbed is also used as the attack detection client for this testbed.

Table 2. 2010 Toyota Prius testbed LSTM model architecture.

LSTM Model	CAN Identifiers	Time Step	Cell Config.	Epoch	Batch Size
0	127, 245, 247	5	10–10	5	5
1	3F9, 6C0, 45C, 45F, 442, 44D	10	20–20	10	10
2	4A8, 499, 49A, 49B, 49D, 3B3, 610	10	20–20	10	10
3	630, 632, 633, 635, 399, 3BB, 3BC, 4A6, 421, 3B6, 611, 3BD	15	30–30	20	15

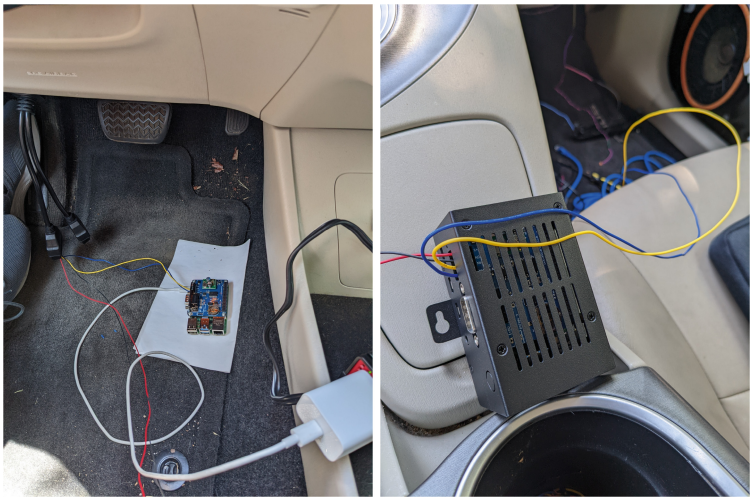

Fig. 9. Attack device (left) and attack detection server (right).

Table 3 specifies the 2014 Toyota Prius automobile LSTM model architecture. The architecture comprises six LSTM models trained to analyze the message timing patterns of the 79 CAN message identifiers listed in the table. Six models were constructed for the 79 CAN message identifiers to prevent model bias towards higher frequency identifiers. CAN message identifiers 383, 381, 382, 3B6, 612, 613, 616, 619 and 61A were not included due to their low transmission frequencies.

The time step corresponds to the memory window used by an LSTM model. The LSTM cell configuration corresponding to each LSTM model specifies the number of LSTM cells used in the LSTM encoder and decoder layers. Each LSTM model has a repeat vector layer between the LSTM encoder and decoder layers, and a time-distributed layer positioned after the LSTM decoder layer. The epochs and batch sizes used for training the LSTM models are also listed.

Table 3. 2014 Toyota Prius automobile LSTM model architecture.

LSTM Model	CAN Identifiers	Time Step	Cell Config.	Epoch	Batch Size
0	0AA, 127, 020, 025, 024, 245, 260, 1C4, 224, 247, 230, 0B4, 235	15	30–30	10	15
1	1AA, 32A, 320, 0B6, 262, 361, 351	10	20–20	15	10
2	6C0, 3F9, 394, 3B7, 620	5	10–10	20	5
3	63B, 4A0, 4A1, 4A2, 610, 4A8, 499, 49A, 49B, 49D, 4A7, 498, 49C, 3B3, 3D3	20	40–40	50	20
4	44D, 45C, 45F, 440, 442, 443	10	40–40	50	10
5	4A6, 4C1, 3B0, 626, 611, 3B1, 420, 4C8, 423, 621, 622, 624, 638, 639, 680, 3B9, 4C3, 3BC, 38E, 4C7, 387, 4C6, 38F, 4DD, 3BD, 3BB, 630, 399, 632, 421, 42F, 633, 635	35	200–200	100	35

All the parameters are the same as those used in the LSTM models for the 2010 Toyota Prius testbed.

Test Environment

Figure 10 shows the test environment. The attack device and attack detection server are connected to the monitored CAN bus. The client and server are connected via a Wi-Fi network. During testing, random synthetic attacks were injected by the attack device into the monitored CAN bus.

The synthetic CAN message injection process implemented by the attack device executes for a duration specified in seconds. This process chooses a random CAN message identifier from a specified bin. If no bin number is specified, then the process chooses a bin at random to obtain a CAN message identifier. The attack device injects the CAN message identifier along with a flag that

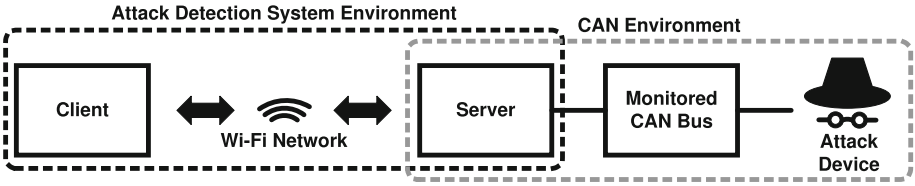

Fig. 10. Test environment.

indicates the absolute truth corresponding to an attack. At this point, the process displays that a particular message was injected along with the bin associated with its identifier. The process then waits for a randomly-generated time in seconds between a specified lower bound and an upper bound.

Table 4. Attack detection data.

MSE Loss Above Threshold	Flag Set	Result
Yes	Yes	TP
Yes	No	FP
No	No	TN
No	Yes	FN

The attack detection system uses the attack detection mode to raise alerts about attacks and collect historical real-time attack data. Table 4 shows the results generated by the attack detection system for High-Speed CAN message sequences. A true positive (TP) output is received when the MSE loss is above the set threshold and a CAN message is received with an attack flag set. A false positive (FP) output is received when the MSE loss is above the set threshold and a CAN message is received with no attack flag set. A true negative (TN) output is received when the MSE loss is below the set threshold and a CAN message is received with no attack flag set. A false negative (FN) output is received when the MSE loss is below the set threshold and a CAN message is received with an attack flag set. When the attack detection system is terminated, the historical real-time attack data is presented to the user.

Note that error propagation often occurs when an attack message is injected. Specifically, when an attack occurs, two message sequences that result in MSE loss above the set threshold are received. The first corresponds to the attack message sequence (true positive) and the second is the next message sequence, which is recognized as part of the attack.

7.2 Experimental Results

This section presents the real-time attack detection results. The true positive, false positive, true negative and false negative detection values are used to compute the attack detection performance metrics.

Performance Metrics. Five metrics are employed to characterize attack detection system performance, precision, sensitivity, specificity, accuracy and F1 score.

The precision metric P describes how well LSTM models detect actual attacks relative to the total number of predicted attacks:

$$\mathrm{P} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

The sensitivity (recall) metric Se describes the ability of LSTM models to correctly determine attacks:

$$\mathrm{Se} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

The specificity metric Sp describes how well LSTM models recognize normal activity without attacks:

$$\mathrm{Sp} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}}$$

The accuracy metric A describes how well LSTM models can determine all observations correctly:

$$\mathrm{A} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}$$

The F1 score metric is the harmonic mean of precision and sensitivity (recall):

$$\mathrm{F1} = \frac{\mathrm{TP}}{\mathrm{TP} + \frac{1}{2}(\mathrm{FP} + \mathrm{FN})}$$

While it is desirable for all five metrics to be high during testing, sensitivity and accuracy are the principal metrics for evaluating LSTM models. This is because it is most important that LSTM models detect as many attacks as possible with high accuracy.

2010 Toyota Prius Testbed Attack Detection Results. Real-time attack testing was conducted on the 2010 Toyota Prius testbed. During the one-hour test, the attack device injected CAN message identifiers at random (28 different identifiers). The tests evaluated the four LSTM models concurrently using the specified MSE loss thresholds.

Table 5 shows the precision, sensitivity, specificity, accuracy and F1 score for each LSTM model. All the LSTM models, except for LSTM Model 0, achieved the maximum precision, sensitivity, specificity, accuracy and F1 score. LSTM

Table 5. 2010 Toyota Prius testbed concurrent test results.

LSTM Model	MSE Loss Threshold	Messages Analyzed	Attacks Injected	Attacks Detected	P	Se	Sp	A	F1
0	0.005	490,015	192	192	0.985	1.000	1.000	1.000	0.992
1	0.300	62,890	209	209	1.000	1.000	1.000	1.000	1.000
2	1.000	50,770	174	174	1.000	1.000	1.000	1.000	1.000
3	1.000	44,010	205	205	1.000	1.000	1.000	1.000	1.000

Model 0 generated three false positives, which reduced its precision and F1 score. Accuracy and specificity were affected negligibly due to the small number of false positives. LSTM Model 0 achieved the maximum sensitivity.

The MSE loss threshold for LSTM Model 0 could be increased to reduce the number of false positives. Also, the model could be improved by model tuning and retraining, or model restructuring. Overall, the four models were able to distinguish between normal CAN traffic and attack traffic with the 28 CAN message identifiers.

2014 Toyota Prius Automobile Attack Detection Results. Four tests were conducted on the 2014 Toyota Prius automobile. The first and second tests evaluated the six LSTM models concurrently for half-hour periods. These two real-time tests involving CAN attacks were performed when the automobile was stationary for safety and network configuration reasons.

The third real-time test evaluated the six LSTM models concurrently when the automobile was moving forward and reversing. The fourth test, which was not performed in real time, evaluated the six LSTM models separately under normal driving conditions. No attacks were launched during the third and fourth tests for safety reasons.

Table 6 shows the results of the first test, which evaluated all six LSTM models executing concurrently. During the half-hour test, the attack device injected CAN message identifiers selected at random (79 different identifiers). The concurrency test evaluated the six LSTM models using the specified MSE loss thresholds. The precision, sensitivity, specificity, accuracy and F1 score are presented for each LSTM model.

In the first test, LSTM Model 0 generated 121 false positives, which reduced its precision, specificity, accuracy and F1 score. Note that the precision (0.142) and F1 score (0.248) are very low. This is due to the decreased rate of true positives caused by the attack device selecting identifiers from random bins instead of identifiers only from the bin associated with LSTM Model 0. Nevertheless, LSTM Model 0 yielded good attack detection accuracy of 0.997. The model achieved maximum attack detection sensitivity at the cost of detecting false positives.

LSTM Models 2 and 4 generated six and two false positives, respectively, which reduced their precision, specificity, accuracy and F1 scores. However, the models achieved the maximum sensitivity. LSTM Models 1, 3 and 5 achieved the maximum precision, sensitivity, specificity, accuracy and F1 scores.

Table 6. 2014 Toyota Prius automobile concurrent test results (Run 1).

LSTM Model	MSE Loss Threshold	Messages Analyzed	Attacks Injected	Attacks Detected	P	Se	Sp	A	F1
0	0.025	707,520	20	20	0.142	1.000	0.997	0.997	0.248
1	0.250	256,620	48	48	1.000	1.000	1.000	1.000	1.000
2	0.500	31,415	43	43	0.878	1.000	0.999	0.999	0.935
3	0.450	55,220	42	42	1.000	1.000	1.000	1.000	1.000
4	0.300	18,420	45	45	0.978	1.000	0.999	0.999	0.989
5	0.100	60,655	46	46	1.000	1.000	1.000	1.000	1.000

Overall, LSTM Models 1 through 5 were able to distinguish between normal CAN traffic and attack traffic with the 79 CAN message identifiers. LSTM model 0 achieved high accuracy for attack detection, but it generated several false positives. The MSE loss threshold for LSTM Model 0 could be increased in an attempt to reduce the number of false positives. Also, LSTM Model 0 could be improved with model tuning and retraining, or model restructuring.

Table 7. 2014 Toyota Prius automobile concurrent test results (Run 2).

LSTM Model	MSE Loss Threshold	Messages Analyzed	Attacks Injected	Attacks Detected	P	Se	Sp	A	F1
0	0.025	703,800	19	19	0.158	0.864	0.998	0.998	0.268
1	0.250	257,200	47	47	0.979	1.000	1.000	1.000	0.959
2	0.750	31,490	48	48	0.941	1.000	1.000	1.000	0.970
3	0.450	55,180	56	56	0.982	1.000	1.000	1.000	0.991
4	0.500	18,470	38	38	1.000	1.000	1.000	1.000	1.000
5	0.100	60,795	35	35	0.854	1.000	0.996	0.996	0.921

Table 7 shows the results of the second test with all six LSTM models executing concurrently. During the half-hour test, the attack device injected CAN message identifiers at random (79 different identifiers). The concurrency test evaluated the six LSTM models using the specified MSE loss thresholds. The precision, sensitivity, specificity, accuracy and F1 score were computed for each LSTM model.

In the second test, LSTM Model 0 generated 101 false positives, which reduced its precision, specificity, accuracy and F1 score. Note that its precision (0.158) and F1 score (0.268) are very low. This is due to the decreased rate of true positives caused by the attack device selecting identifiers from random bins instead of only identifiers from the bin associated with LSTM Model 0. The model generated three false negatives, which reduced its sensitivity to 0.864. However, LSTM Model 0 yielded good attack detection accuracy of 0.998 despite the false positives.

LSTM Models 1, 2, 3 and 5 generated one, three, one and six false positives, respectively, which reduced their precision and F1 scores. Specificity and accuracy were affected negligibly due to the small numbers of false positives. All four

Table 8. 2014 Toyota Prius automobile forward/reverse motion test results.

LSTM Model	MSE Loss Threshold	Messages Analyzed	Sp
0	0.025	40,680	0.998
1	0.250	13,910	0.999
2	0.750	1,635	0.997
3	0.450	3,340	1.000
4	0.500	1,170	1.000
5	0.100	60,795	0.772

models achieved the maximum sensitivity. LSTM Model 4 fared best, achieving the maximum precision, sensitivity, specificity, accuracy and F1 score.

Overall, LSTM Models 1 through 5 were able to distinguish between normal CAN traffic and attack traffic with the 79 CAN message identifiers. LSTM Model 0 achieved high accuracy for attack detection, but generated several false positives. Because the model generated false positives and false negatives, adjusting the MSE loss threshold would likely not improve its performance. However, LSTM Model 0 could be improved with model tuning and retraining, or model restructuring.

Although attacks could not be launched when the automobile was moving, it was important to test the attack detection system under normal operating conditions. Table 8 shows the results of the forward/reverse motion test. During the two-minute test, the automobile moved forward ten feet and reversed ten feet. Because no attacks were launched, only the specificity metric was computed using the true negatives and false positives.

LSTM Models 0, 1, 2 and 5 generated five, one, one and 18 false positives, respectively, negatively affecting their specificity. LSTM Models 3 and 4 achieved the maximum specificity. Overall, the LSTM models were able to recognize normal CAN traffic. LSTM Model 5 likely yielded a lower specificity of 0.772 due to CAN traffic interruptions when engaging the gear shift mechanism.

The final test involved the execution of the six LSTM models under normal driving conditions. CAN traffic was logged while driving around campus and the log file was input to the six LSTM models. Table 9 shows the results of the normal driving test. Because no attacks were launched, only the specificity metric was computed using the true negatives and false positives.

LSTM Models 0, 2, 3 and 5 generated 28, 1, 2 and 27 false positives, respectively, which reduced their specificity. LSTM Models 1 and 4 achieved the maximum specificity. Overall, the six LSTM models were able to recognize normal CAN traffic. Model 5 likely yielded a lower specificity of 0.892 due to CAN traffic interruptions during normal driving conditions.

Table 9. 2014 Toyota Prius automobile normal driving test results.

Model	MSE Loss Threshold	Messages Analyzed	Sp
0	0.025	171,615	0.998
1	0.250	32,670	1.000
2	0.750	4,040	0.999
3	0.450	6,940	0.994
4	0.500	2,330	1.000
5	0.100	7,770	0.892

Potential Model Improvements. For the 2010 Toyota Prius testbed, precision ranged from 0.985 to 1.000, sensitivity, specificity and accuracy were a perfect 1.000, and the F1 score ranged from 0.992 to 1.000. For the 2014 Toyota Prius automobile, precision ranged from 0.142 to 1.000, sensitivity ranged from 0.864 to 1.000, specificity ranged from 0.772 to 1.000, accuracy ranged from 0.980 to 1.000, and the F1 score ranged from 0.248 to 1.000. The low precision and F1 scores were due to high false positive to true positive rates in just the LSTM Model 0 in the 2014 Toyota Prius automobile tests. Nevertheless, the attack detection results are very good. Specifically, sensitivity ranged from 0.864 to 1.000 and accuracy ranged from 0.980 to 1.000 for the LSTM models in the 2014 Toyota Prius automobile tests. Sensitivity and accuracy are the most important metrics because the LSTM models must recognize normal traffic and detect as many attacks as possible with high accuracy.

LSTM model performance could be improved by reducing the false positives and/or false negatives. One approach is to adjust the MSE loss threshold to make the model more or less sensitive; this may be done using a trial-and-error procedure or using a receiver operating characteristic (ROC) curve. Another approach is to redesign the LSTM model or retrain it using additional data and epochs, but this would be more time consuming than MSE loss threshold adjustment. Alternatively, the LSTM model may be divided into smaller models to simplify the message sequences to be learned. This would involve dividing a CAN message identifier bin into smaller bins. Bin subdivision, while effective, would be more time consuming than the MSE loss threshold adjustment and model redesign/retraining approaches.

It is possible that some CAN message identifiers do not have periodic timing patterns. Since an LSTM model cannot not be trained effectively with aperiodic timing patterns, the corresponding CAN message identifiers would have to be discarded.

8 Conclusions

The CAN attack detection system described in this work employs LSTM networks to monitor automobile CANs, detect attacks and raise alerts in real

time. LSTM networks are leveraged because they can learn patterns with long sequences. To address LSTM network bias, binning is employed as a novel concept in LSTM model development for automobile CANs. In this process, CAN message identifiers are separated into bins depending on their relative message frequencies. A separate LSTM model is trained to recognize the traffic patterns of the CAN message identifiers in each bin.

A repeatable design framework is presented for constructing and training multiple LSTM networks that learn normal CAN message timing patterns. The design framework lays out the computational resources as well as the data collection and preprocessing and LSTM model development and training steps. The framework enables new LSTM models to be trained and updated for automobiles of different makes, models and years.

Another key contribution is the implementation of real-time attack detection. The attack detection system leverages a server-client configuration on a monitored automobile CAN bus. The server is an inexpensive Raspberry Pi device connected directly to an automobile CAN bus that captures, logs and transmits CAN message traffic to a client via a Wi-Fi network. The client, a workstation located outside the automobile, provides the computational resources needed for real-time attack detection. Trained LSTM models executing on the client workstation process the transmitted CAN messages, identify attacks and send alerts via the Wi-Fi network.

The attack detection system was evaluated using a 2010 Toyota Prius testbed and a fully-operational 2014 Toyota Prius automobile. An attack device was employed to inject random CAN message identifiers at random times. For the 2010 Toyota Prius testbed, the attack detection precision ranged from 0.985 to 1.000, sensitivity, specificity and accuracy were perfect 1.000, and the F1 score ranged from 0.992 to 1.000. For the 2014 Toyota Prius automobile, the attack detection precision ranged from 0.142 to 1.000, sensitivity ranged from 0.864 to 1.000, specificity ranged from 0.772 to 1.000, accuracy ranged from 0.980 to 1.000, and the F1 score ranged from 0.248 to 1.000. The low precision and F1 scores were due to high false positive to true positive rates in just one of the six LSTM models in the attack detection system used in the 2014 Toyota Prius automobile experiments. Nevertheless, the attack detection results are very good – LSTM model sensitivity ranged from 0.864 to 1.000 and accuracy ranged from 0.980 to 1.000. Sensitivity and accuracy are the most important metrics because LSTM models must recognize normal traffic and detect as many attacks as possible with high accuracy.

Acknowledgment. This research was supported by the National Science Foundation under Grant no. DGE 1501177.

References

1. Aldhyani, T., Alkahtani, H.: Attacks on autonomous vehicles: a deep learning algorithm for cybersecurity. Sensors **22**(1), article no. 360 (2022)
2. Bosch, CAN Specification Version 2.0, Technical Specification, Stuttgart, Germany (1991)
3. Bosch, CAN with Flexible Data-Rate Version 1.0, Technical Specification, Gerlingen, Germany (2011)
4. Bozdal, M., Samie, M., Aslam, S., Jennions, I.: Evaluation of CAN bus security challenges. Sensors **20**(8), article no. 2364 (2020)
5. Brownlee, J.: Long Short-Term Memory Networks with Python. Machine Learning Mastery, San Juan, Puerto Rico (2020)
6. Checkoway, S., et al.: Comprehensive experimental analysis of automotive attack surfaces. In: Proceedings of the Twentieth USENIX Security Symposium (2011)
7. Geron, A.: Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow: Concepts, Tools and Techniques to Build Intelligent Systems. O'Reilly Media, Sebastopol (2019)
8. Greenberg, A.: Hackers remotely kill a jeep on the highway – with me in it, Wired (2015)
9. Hanselmann, M., Strauss, T., Dormann, K., Ulmer, H.: CANet: an unsupervised intrusion detection system for high-dimensional CAN bus data. IEEE Access **8**, 58194–58205 (2020)
10. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
11. Hoppe, T., Dittman, J.: Sniffing/replay attacks on CAN buses: a simulated attack on the electric window lift classified using an adapted CERT taxonomy. In: Proceedings of the Second Workshop on Embedded Systems Security (2007)
12. Hoppe, T., Kiltz, S., Dittmann, J.: Security threats to automotive CAN networks - practical examples and selected short-term countermeasures. Reliab. Eng. Syst. Saf. **96**(1), 11–25 (2011)
13. Koscher, K., et al.: Experimental security analysis of a modern automobile. In: Proceedings of the IEEE Symposium on Security and Privacy, pp. 447–462 (2010)
14. Liu, J., Zhang, S., Sun, W., Shi, Y.: In-vehicle network attacks and countermeasures: challenges and future directions. IEEE Network **31**(5), 50–58 (2017)
15. Lloyd, S.: Least squares quantization in PCM. IEEE Trans. Inf. Theory **28**(2), 129–137 (1982)
16. Matthews, C.: Jeep hack: Fiat recalls 1.4 million vehicles for software fix, Fortune (2015)
17. McKinney, W.: Python for Data Analysis - Data Wrangling with Pandas, NumPy and IPython. O'Reilly Media, Sebastopol (2018)
18. Möller, D.P.F., Haas, R.E.: Automotive cybersecurity. In: Guide to Automotive Connectivity and Cybersecurity. CCN, pp. 265–377. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-73512-2_6
19. Ranjan, C.: Step-by-step understanding of LSTM autoencoder layers. Towards Data Science (2019). https://towardsdatascience.com/step-by-step-understanding-lstm-autoencoder-layers-ffab055b6352
20. Smith, C.: The Car Hacker's Handbook: A Guide for the Penetration Tester. No Starch Press, San Francisco (2016)
21. Sun, H., Chen, M., Weng, J., Liu, Z., Geng, G.: Anomaly detection in in-vehicle networks using CNN-LSTM with attention mechanism. IEEE Trans. Veh. Technol. **70**(10), 10880–10893 (2021)

22. Tariq, S., Lee, S., Woo, S.: CANtransfer: transfer-learning-based intrusion detection in a controller area network using a convolutional LSTM network. In: Proceedings of the Thirty-Fifth Annual ACM Symposium on Applied Computing, pp. 1048–1055 (2020)
23. Taylor, A., Leblanc, S., Japkowicz, N.: Anomaly detection in automobile control network data with long short-term memory networks. In: Proceedings of the IEEE International Conference on Data Science and Advanced Analytics, pp. 130–139 (2016)
24. Valasek, C., Miller, C.: Adventures in Automotive Networks and Control Units. Technical White Paper, IOActive, Seattle, Washington (2014)
25. Valasek, C., Miller, C.: Remote Exploitation of an Unaltered Passenger Vehicle. Technical White Paper, IOActive, Seattle, Washington (2015)
26. Xiao, J., Wu, H., Li, X.: Robust and self-evolving IDS for in-vehicle networks by enabling spatiotemporal information. In: Proceedings of the Twenty-First IEEE International Conference on High Performance Computing and Communications, Seventeenth IEEE International Conference on Smart City and Fifth IEEE International Conference on Data Science and Systems, pp. 1390–1397 (2019)
27. Yang, Y., Duan, Z., Tehranipoor, M.: Identifying a spoofing attack on an in-vehicle CAN bus based on the deep features of an ECU fingerprint signal. Smart Cities **3**(1), 17–30 (2020)
28. Zhu, K., Chen, Z., Peng, Y., Zhang, L.: Mobile edge-assisted literal multi-dimensional anomaly detection in an in-vehicle network using LSTM. IEEE Trans. Veh. Technol. **68**(5), 4275–4284 (2019)