Machine Learning for Anomaly Detection in Automotive Cyber-Physical Systems

Vipin Kumar Kukkala, Sooryaa Vignesh Thiruloga, and Sudeep Pasricha

1 Introduction

Today's vehicles are sophisticated cyber-physical systems (CPS) that consists of multiple interconnected embedded systems known as electronic control units (ECUs). The ECUs control various vehicular functions and communicate with each other using the in-vehicle network. In recent years, the number of ECUs along with the complexity of software running on these ECUs has been increasing rapidly, to enable advanced driver assistance systems (ADAS) features such as adaptive cruise control, collision avoidance, lane keep assist, and blind spot warning. This has resulted in an increase in the complexity of the in-vehicle network over which huge volumes of automotive sensor and real-time decision data, and control directives are communicated. This in turn has led to various challenges related to the reliability [1–4], security [5–9], and real-time control of automotive applications [10–13].

Recent developments in ADAS resulted in increased interaction with various external systems using advanced communication standards such as 5G technology and Vehicle-to-X (V2X) [14]. Unfortunately, this makes automotive embedded systems highly susceptible to various cybersecurity threats that can have catastrophic consequences. The vehicular attacks in [15–17] have presented different ways to gain access to the in-vehicle network and override vehicle controls by injecting anomalous messages. With the connected and autonomous vehicles (CAVs) on the horizon, these security concerns will get further aggravated. Therefore, it is crucial to prevent unauthorized access to in-vehicle networks by external attackers to ensure the security of automotive CPS.

V. K. Kukkala (✉) · S. V. Thiruloga · S. Pasricha
Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO, USA
e-mail: kvipin@rams.colostate.edu; sooryaa@colostate.edu; sudeep@colostate.edu

S. Pasricha, M. Shafique (eds.), *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing*, https://doi.org/10.1007/978-3-031-40677-5_11

Traditional computer networks utilized firewalls to defend the networks from external attackers. However, no firewall is flawless, and no network can be completely secure. Therefore, there is a need for an active monitoring system that continuously monitors the network to identify malicious messages in the system. An anomaly detection system (ADS) can be used to continuously monitor the in-vehicle network traffic and trigger alerts when suspicious messages or known threats are detected, which is typically the last line of defense in automotive CPSs.

At a high level, ADSs are categorized into two types: *(i) rule-based* and (ii) *machine learning-based*. Rule-based ADSs observe for traces of previously observed attack signatures whereas machine learning-based ADSs observe for the deviation from the known normal system behavior to detect the presence of an attacker. Rule-based ADS can have faster detection rates and very few false alarms (false positive rate) but are limited to detecting only known attacks. On the contrary, machine learning-based ADS can detect both previously observed and novel attacks but can suffer from comparatively slower detection times and higher false alarm rate. An efficient ADS needs to be robust, scalable, and incur minimal overhead (lightweight). Moreover, practical ADSs need to have a wide attack coverage (being able to detect both known and unknown attacks) with high confidence in detection and low false alarms as recovering from false alarms can be costly.

Obtaining the signature of every possible attack is impractical and would limit us to only detecting known attacks. Hence, we believe that machine learning-based ADSs provide a more pragmatic solution to this problem. Additionally, due to the ease of acquiring in-vehicle network data, large volumes of in-vehicle message data can be collected, which facilitates the use of advanced deep learning models for detecting anomalies in automotive CPS [9].

In this chapter, we propose a novel ADS framework called *INDRA*, first presented in [6], that monitors the messages in controller area network (CAN)-based automotive CPS for anomalies. During the offline phase, *INDRA* uses a deep learning-based model to learn the normal system behavior in an unsupervised manner. At runtime, *INDRA* continuously scans the network for anomalous messages in the network. *INDRA* aims to maximize the detection accuracy with minimal false alarms and overhead on the ECUs.

Our novel contributions in this work are as follows:

1. We introduced a gated recurrent unit (GRU)-based recurrent autoencoder network to learn the normal system behavior during the offline phase.
2. We presented an anomaly score (AS) metric to measure deviation from the normal system behavior.
3. We conducted a comprehensive analysis toward the selection of thresholds for the anomaly score metric.
4. We compare our proposed *INDRA* framework with the best-known prior works in the area, to show its effectiveness.

2 Related Work

Several techniques have been proposed to design ADS for protecting time-critical automotive CPS. These works try to detect multiple attacks by monitoring the in-vehicle network data.

Rule-based ADS detects known attacks by using the information about previously observed attack signatures. A language theory-based model [18] was proposed to derive attack signatures. However, this technique fails to detect attacks when it misses the packets transmitted during the early stages of an attack. The authors in [19], used transition matrices to detect attacks in a CAN bus. They were able to achieve a low false-positive rate for simple attacks but failed to detect advanced replay attacks. In [20], the authors identify notable attack signatures such as an increase in message frequency and missing messages to detect attacks. In [21], the authors proposed a specification-based approach to detect attacks; they analyze the behavior of the system and compare it with the predefined attack patterns to detect anomalies. However, their system fails to detect unknown attacks. The authors in [22] propose an ADS technique using the Myers algorithm [23] under the map-reduce framework. In [24], a time-frequency analysis of CAN messages is used to detect multiple anomalies. The authors analyzed message frequency at design time in [25] to derive regular operating mode region. This region is observed for deviations at runtime to detect anomalies. The sender ECU's clock skew and the messages are used to detect attacks [26] by observing for variations in the clock-skew at runtime. The authors in [27] performed a formal analysis on clock-skew-based ADS and evaluated on a real vehicle. In [28], a memory heat map is used to characterize the memory behavior of the operating system to detect anomalies. In [29], an entropy-based ADS is proposed, which observes for change in system entropy to detect anomalies. Nonetheless, the technique fails to detect small scale attacks for which the entropy change is minimal. In conclusion, rule-based ADSs offer a solution to the intrusion detection problem with lower false positive rates but cannot detect more complex and novel attacks. Moreover, obtaining signatures of every possible attack pattern is not practical.

Machine learning-based ADSs aim to learn the normal system behavior in an offline phase and observe for any deviation from the learned normal behavior to detect anomalies at runtime. In [30], the authors proposed a sensor-based ADS that utilizes attack detection sensors to monitor various system events to observe for deviations from normal behavior. However, this approach is expensive and suffers from poor detection rates. In [31], a one-class support vector machine (OCSVM)-based ADS was introduced, but it suffers from poor detection latency. An ensemble of different nearest neighbor classifiers was used in [32] to distinguish between a normal and an attack-induced CAN payload. The authors in [33] proposed a decision-tree-based detection model to monitor the physical features of the vehicle to detect attacks. However, this model is not practical and suffers from high anomaly detection latencies. In [34], a hidden Markov model (HMM)-based technique was proposed to monitor the temporal relationships between messages to detect attacks.

Table 1 Performance metrics comparison between our proposed *INDRA* framework and state-of-the-art machine learning-based anomaly detection works

	Performance metrics			
Technique	Lightweight model	Low false positive rate	High detection accuracy	Fast inference time
PLSTM [38]	X	✓	X	X
RepNet [39]	✓	X	X	✓
CANet [36]	X	✓	✓	X
INDRA	✓	✓	✓	✓

A deep neural network-based approach was proposed to scan the payload in the in-vehicle network in [35]. This approach is not scalable as it is fine-tuned for a low priority tire pressure monitoring system (TPMS), which makes it hard to adapt to high priority powertrain applications. In [36] a long-short-term memory (LSTM)-based ADS for multi-message ID detection was proposed. However, the model architecture is highly complex and incurs high overhead on the ECUs. An LSTM-based ADS to detect insertion and dropping attacks (explained in Sect. 4.3) is proposed in [37]. In [38], an LSTM-based predictor model is proposed to predict the subsequent time step message value at a bit level and observe for large variations to detect anomalous messages. A recurrent neural network (RNN)-based ADS to learn the normal CAN message pattern in the in-vehicle network is proposed in [39]. In [40], a hybrid ADS was proposed which utilizes a specification-based system in the first stage and an RNN-based model in the second stage to detect anomalies in time-series data. Several other machine models such as the stacked LSTMs and temporal convolutional neural networks (TCNs)-based techniques were proposed in [7, 8], respectively. However, none of these techniques provides a complete system-level solution that is scalable, reliable, and lightweight to detect various attacks for in-vehicle networks.

In this chapter, we introduce a lightweight recurrent autoencoder-based ADS using gated recurrent units (GRUs) that monitors the in-vehicle network messages at a signal level to detect multiple types of attacks with higher efficiency than various state-of-the art works in this area. A summary of some of the state-of-the-art works' performance under different metrics and our proposed *INDRA* framework is presented in Table 1. An exhaustive analysis of each metric and evaluation results are presented later in Sect. 6.

3 Sequence Learning Background

The availability of increased computing power from GPUs and custom accelerators training deep neural networks with many hidden layers became feasible and has led to the creation of powerful models for solving difficult problems in many domains.

One such problem is detecting anomalies in automotive CPS. In an automotive CPS, the communication between ECUs occurs in a time-dependent manner. Therefore, there is temporal relationship between the messages, which can be exploited in order to detect anomalies. However, this cannot be achieved using typical feedforward neural networks where the output of a specific input at an instance is independent of the other inputs. Sequence models can be an appropriate approach for such problems, as they inherently handle sequences and time-series data.

3.1 Sequence Models

A sequence model is a function that ensures that the outcome is reliant on both the current and prior inputs. The recurrent neural network (RNN), which was introduced in [41], is an example of such a sequence model. Moreover, other sequence models such as gated recurrent unit (GRU) and long-short-term memory (LSTM) have also been developed.

3.1.1 Recurrent Neural Networks (RNNs)

An RNN is a form of artificial neural network that takes the sequential data as input and tries to learn the relationships between the elements in the sequence. The hidden state in RNNs allows learned information from previous time steps to persist over time. An RNN unit with feedback is shown in Fig. 1a, and an unrolled RNN in time is shown in Fig. 1b.

The output h_t of an RNN unit is a function of both the input x_t and the previous output h_{t-1}:

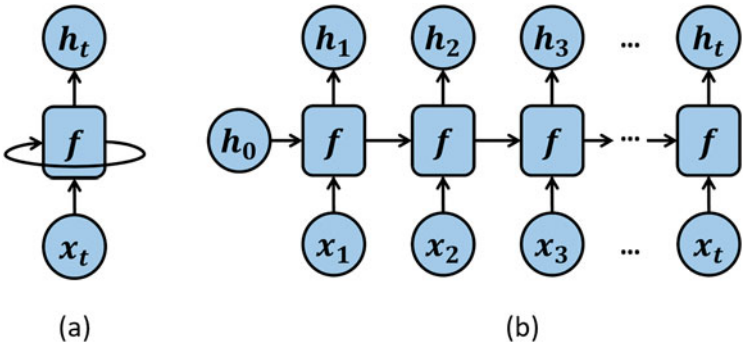


Fig. 1 (**a**) A single RNN unit and (**b**) RNN unit unrolled in time, where f is the RNN unit, x is the input, and h represents hidden states

$$h_t = f\left(W x_t + U h_{t-1} + b\right) \tag{1}$$

where f is a nonlinear activation function (e.g., sigmoid or tanh), U and W are weight matrices, and b is the bias term. One of the major limitations of RNNs is that they are very hard to train. Since RNNs and other sequence models handle sequences or time-series inputs, backpropagation occurs through various time steps (known as backpropagation through time). During this process, the feedback loop in RNNs causes the errors to expand or shrink rapidly, thereby creating exploding or vanishing gradients, respectively, which in turn destroys the information in backpropagation. This vanishing gradient problem prohibits RNNs from learning *long-term dependencies*. To solve this problem, additional states and gates were introduced in the RNN unit in [42] to remember long-term dependencies, which led to the development of LSTM Networks.

3.1.2 Long-/Short-Term Memory (LSTM) Networks

LSTMs unlike RNNs uses cell state and hidden state information along with multiple gates to remember long-term dependencies between messages. The cell state can be imagined as a freeway that carries relevant information throughout the processing of a sequence. The state stores information from previous time steps so that it can be used in subsequent time steps, reducing the effects of short-term memory. The gates modify the information in the cell state. As a result, the gates in LSTM assist the model in determining which information should be retained and which should be ignored.

An LSTM unit contains three gates: (i) input gate (f_t) (ii) forget gate (i_t), and (iii) output gate (o_t) as shown in Fig. 2a. The forget gate is a binary gate that determines which information from the previous cell state (c_{t-1}) to retain. The input gate adds relevant information to the cell state (c_t). Finally, the output gate uses information from the previous two gates to produce an output. An LSTM unit unrolled in time is shown in Fig. 2b.

LSTMs learn long-term dependencies in a sequence by using a combination of different gates and hidden states. However, they are not computationally efficient due to the addition of multiple gates, as the sequence path is more complicated than in RNNs, which in turn requires more memory at runtime. Moreover, training LSTMs have a high computation overhead even when the advanced training methods such as truncated backpropagation are employed. To overcome abovementioned limitations, a simpler recurrent neural network called gated recurrent unit (GRU) network was introduced in [43]. GRUs can be trained faster than LSTMs and also can remember dependencies in long sequences with minimal overhead (in both memory and runtime), while solving the vanishing gradient problem.

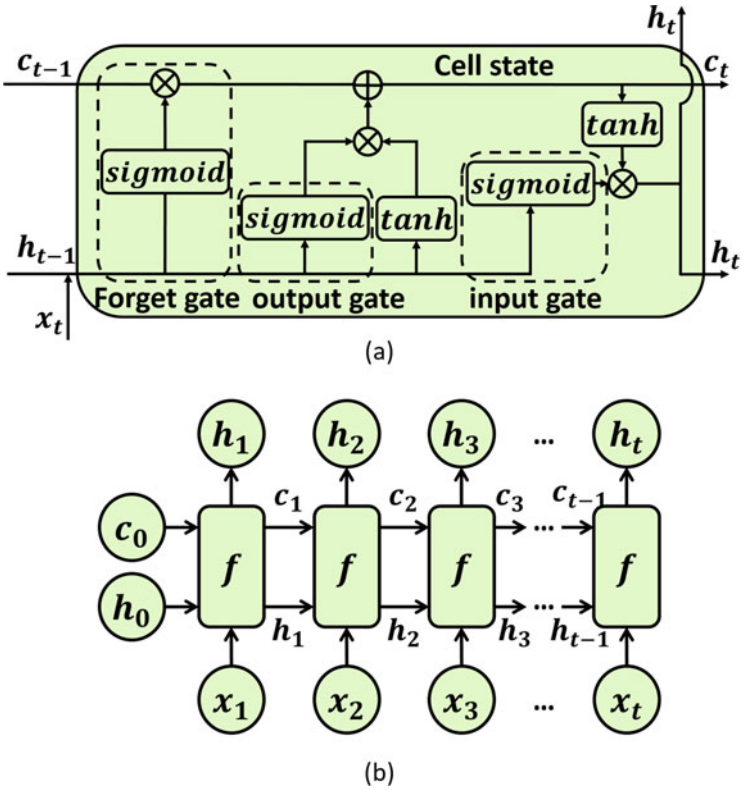


Fig. 2 (**a**) A single LSTM unit with different gates and (**b**) unrolled LSTM unit in time, where *f* is an LSTM unit, *x* is input, *c* is cell state, and *h* is the hidden state

3.1.3 Gated Recurrent Unit (GRU)

Unlike LSTMs, a GRU unit takes a different route for gating information. The input and forget gate of the LSTM is combined into a solitary *update* gate and in addition combines hidden and cell state, as shown in Fig. 3a, b.

A typical GRU unit contains two gates: (i) reset gate and (ii) update gate. The reset gate combines new input with previous memory, while the update layer determines how much relevant data should be stored. Thus, a GRU unit controls the data stream similar to an LSTM by uncovering its hidden layer contents. Moreover, GRUs are computationally more efficient than LSTMs as they achieve this using fewer gates and states, with low memory overhead. It is crucial to use lightweight machine learning models as real-time automotive ECUs are highly resource-constrained embedded systems with strict energy and power budgets. Thus, GRU-based networks are an ideal fit for inference in automotive systems. Hence, *INDRA* chose to use a lightweight GRU-based model to implement an ADS (explained in detail in Sect. 5).

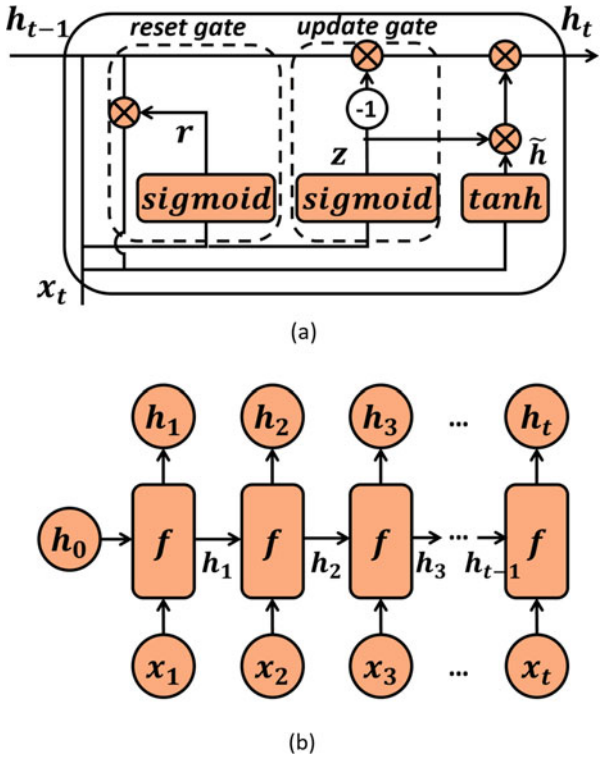


Fig. 3 (**a**) A single GRU unit with different gates and (**b**) GRU unit unrolled in time, where f is a GRU unit, x is input, and h is the hidden state

The major advantage of sequence models is that they can be trained in both supervised and unsupervised learning fashion. Due to the large volume of CAN message data in a vehicle, labeling all that data can become very tedious. Additionally, the variability in the messages between vehicle models from the same manufacturer and the proprietary nature of this information makes it even more challenging to label messages correctly. Nonetheless, due to the ease of obtaining CAN message data via onboard diagnostics (OBD-II), large amounts of unlabeled data can be collected easily. Thus, *INDRA* uses GRUs in an unsupervised learning setting.

3.2 Autoencoders

Autoencoders are unsupervised learning-based artificial neural networks who try to reconstruct the input by learning the latent input features. They accomplish this by encoding the input data (x) to a hidden layer and finally decoding it to produce a reconstruction $\tilde{x}$ (as shown in Fig. 4). This encoded information at the hidden

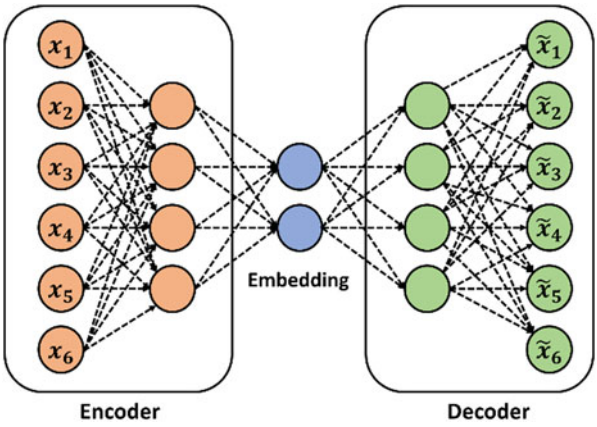


Fig. 4 A simple autoencoder network with encoder, decoder, and embedding layers of the network

layer is called an embedding. The layers that are used to create this embedding are called the encoder, and the layers that are used in reconstructing the embedding into the original input (decoding) are called the decoder. During the training process, the encoder attempts to learn a nonlinear mapping of the inputs, while the decoder tries to learn the nonlinear mapping of the embedding to the inputs. The encoder and decoder accomplish this with the help of nonlinear activation functions such as tanh and rectified linear unit (ReLU). Moreover, the autoencoder aims to recreate the input as closely as possible by extracting important features from the inputs with a goal of minimizing reconstruction loss. The most used loss functions in autoencoders include mean squared error (MSE) and Kullback-Leibler (KL) divergence.

Since autoencoders aim to reconstruct the input by learning the underlying distribution of the input data, they are an excellent choice for efficiently learning and reconstructing highly correlated time-series data by learning the temporal relations between messages. *Hence, our proposed INDRA framework uses lightweight GRUs in an autoencoder to learn latent representations of CAN message data in an unsupervised learning setting*.

4 Problem Definition

4.1 System Model

In this chapter, we consider a generic automotive *system* consisting of multiple ECUs connected using a CAN-based in-vehicle network, as shown in Fig. 5. Each ECU connected in the network is responsible for running a specific set of automotive applications that are hard real time in nature (i.e., have strict timing and deadline

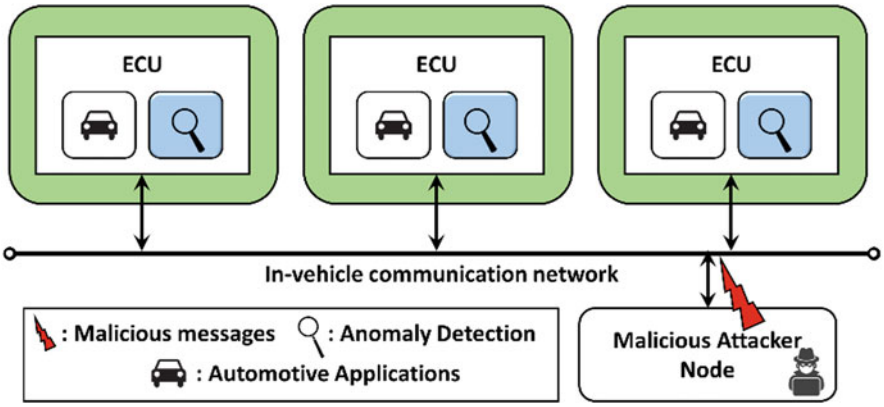


Fig. 5 Overview of the automotive system model considered in *INDRA*

constraints). Moreover, we assume that each ECU also runs anomaly detection applications (ADS), which are responsible for monitoring and detecting anomalies in the in-vehicle network. *INDRA* considers a distributed ADS approach (anomaly detection application is collocated with automotive applications) as opposed to a centralized ADS approach in which one central ECU handles all anomaly detection tasks due to the following reasons:

- A centralized ADS approach is susceptible to single-point failures, which can completely expose the system to the attacker.
- In the worst-case scenarios such as during a flooding attack (explained in Sect. 4.3), the centralized system might not be able to communicate with the victim ECUs due to highly congested in-vehicle network.
- If an attacker successfully tricks the centralized ADS ECU, the attacks can go undetected by the other ECUs, compromising the entire system; however, in a distributed ADS scenario, it requires fooling multiple ECUs (which is more difficult) to compromise the system. Moreover, in a distributed ADS scenario, even if one of the ECU is compromised, the attacks can still be detected by the decentralized intelligence.
- In a distributed ADS, ECUs can stop accepting messages as soon as an anomaly is detected rather than having to wait for a centralized system to notify them, resulting in faster reaction times.
- With a distributed ADS, the computation load of ADS is split among the ECUs and monitoring can be limited to only required messages. As a result, multiple ECUs can independently monitor a subset of messages with lesser overhead.

For the abovementioned reasons, many prior works such as [18, 25] also consider a distributed ADS approach. Furthermore, with increasing computation power of automotive ECUs, the collocation of ADS applications with real-time automotive

applications in a distributed manner should not be a problem, if the ADS has a minimal overhead. *INDRA* framework is not only lightweight but also scalable, and achieves high anomaly detection performance, as discussed in Sect. 6.

An ideal ADS should have low susceptibility to noise, low cost, and a low power/energy footprint. The following are some of the key characteristics of an efficient IDS, which were taken into consideration when designing our *INDRA* ADS:

- *Lightweight*: Anomaly detection tasks can incur additional overhead on ECU, which could result in poor application performance and missed deadlines for real-time applications, which is catastrophic. Therefore, *INDRA* aims to have a lightweight ADS that incurs minimal overhead on the ECU.
- *Few false positives*: This is a highly desired quality in any type of ADS (even outside of the automotive domain), as dealing with false positives can quickly become costly. Thus, a good ADS is expected to have few false positives or false alarms.
- *Coverage*: This defines the range of attacks that an ADS can detect. A good ADS must be capable of detecting more than one type of attack. Moreover, a high coverage for ADS will make the system resistant to multiple attack surfaces.
- *Scalability*: This is an important requirement as the number of ECUs in emerging vehicles is growing along with software and network complexity. A good ADS should be highly scalable and capable of supporting multiple system sizes.

4.2 Communication Model

This subsection discusses vehicle communication model that was considered for *INDRA* framework. *INDRA* primarily focuses on detecting anomalies in controller area network (CAN) bus-based automotive CPS. CAN is the most commonly used in-vehicle network protocol in modern automotive systems. CAN offers a low cost, lightweight, event-triggered communication where messages are transmitted in the form of frames. A typical standard CAN frame structure is shown in Fig. 6, and the length of each field (in bits) is shown on the top. The standard CAN frame consists of a header, payload, and trailer segment. The header contains information of the message identifier (ID) and the length of the message, whereas the payload segment contains the actual data that needs to be transmitted. The trailer section is mainly used for error checking at the receiver. A variation of the CAN protocol, called CAN-extended or CAN 2.0B, is also being deployed increasingly in modern vehicles. The key difference being that CAN extended has a 29-bit identifier, which allows for a greater number of messages IDs.

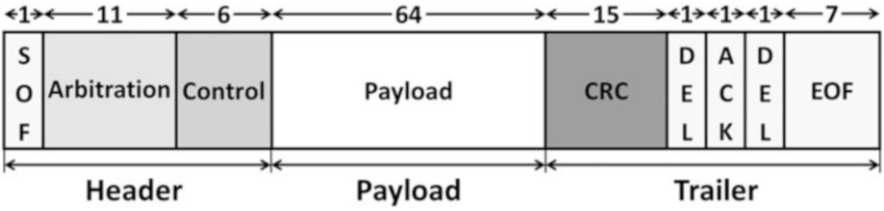


Fig. 6 Standard frame format of a CAN message

Signal Name	Message	Start bit	Length	Byte Order	Value Type
Battery_Current	Status	0	16	Intel	Signed
Battery_Voltage	Status	16	16	Intel	Unsigned
Motor_Current	Status	32	16	Intel	Signed
Motor_Speed	Status	48	8	Intel	Signed
Motor_Direction	Status	56	8	Intel	Unsigned

Fig. 7 An example real-world CAN message with signal information [44]

Our proposed *INDRA* ADS focuses on monitoring the payload segment of the CAN frame and observes for anomalies within the payload to detect cyberattacks. This is because most modern-day attacks involve an attacker modifying the payload to accomplish malicious activities. An attacker can also target the header or trailer segments, but the message would get rejected at the receiver. The payload segment comprises of multiple data entities called signals. An example real-world CAN message with the list of signals within the message is shown in Fig. 7. Each signal has a fixed size (in bits), assigned a particular data type, and a start bit that specifies its location in the 64-bit payload segment of the CAN message.

INDRA focuses on monitoring individual signals within CAN payload to observe for anomalies and detect attacks. During training, *INDRA* learns the temporal dependencies between the messages at a signal level and observes for deviations at runtime to detect attacks. The ability to detect attacks at a signal level enables *INDRA* to not only detect the presence of an attacker but also help in identifying the signal within the message that is under attack. This can be valuable information for understanding the intentions of the attacker, which can be used for developing appropriate countermeasures. The details about the signal level monitoring of *INDRA* ADS are discussed in Sect. 5.2. *Note*: Even though our proposed *INDRA* framework focuses on detecting attacks by monitoring CAN messages, our approach is protocol-agnostic and can be used with other in-vehicle network protocols (such as FlexRay and LIN) with minimal changes.

4.3 Attack Model

Our proposed *INDRA* ADS aims to protect the vehicle from various types of attacks that are most commonly seen and difficult to detect attacks in the domain of automotive CPS. Moreover, these attacks have been widely used in literature to evaluate ADSs.

1. *Flooding attack*: This is the most common and simple to launch attack, and it requires no knowledge of the system. In this attack, the attacker continuously floods the in-vehicle network with a random or specific message with the goal of preventing other ECUs from accessing the bus and rendering the bus unusable. These attacks are typically detected by the vehicle's network bridges and gateways and often do not reach the last line of defense (the ADS). However, it is crucial to consider these attacks as they can have serious consequences when not handled correctly.
2. *Plateau attack*: In this attack, an attacker overwrites a signal value with a constant value for the entirety of the attack interval. The severity of this attack is determined by the magnitude of the jump (increase in signal value) and the duration for which it is held. Larger jumps in signal values are easier to detect compared with shorter jumps.
3. *Continuous attack*: In this attack, an attacker gradually overwrites the signal value with the goal of achieving some target value while avoiding the activation of an ADS. This attack is difficult to detect and can be sensitive to the ADS parameters (discussed in Sect. 5.2).
4. *Suppress attack*: In this attack, the attacker suppresses the signal value(s) by either disabling the target ECU's communication controller or shutting down the ECU. These attacks are easy to detect because they disrupt message transmission for long durations but are harder to detect for shorter durations.
5. *Playback attack*: In this attack, the attacker attempts to trick the ADS by replaying a valid series of message transmissions from the past. This attack is hard to detect if the ADS lacks the ability to capture the temporal relationships between messages and detect when they are violated.

Moreover, in this work, we assume that the attacker can gain access to the vehicle using the most common attack vectors such as connecting to V2X systems that communicate with the outside world (e.g., infotainment and connected ADAS systems), connecting to the OBD-II port, probe-based snooping on the in-vehicle bus, and by replacing an existing ECU. We also assume that the attacker has access to the network parameters (such as parity, flow control, and BAUD rate) that can further assist in gaining access to the in-vehicle network.

Problem objective: The goal of our proposed *INDRA* framework is to implement a lightweight ADS that can detect a variety of attacks (mentioned above) in a CAN-based automotive CPS, with a high detection accuracy and low false positive rate while maintaining a large attack coverage.

5 *INDRA* Framework Overview

INDRA framework enables a machine learning-based signal level ADS for monitoring CAN messages in automotive embedded CPS. An overview of the proposed framework is depicted in Fig. 8. The *INDRA* framework is divided into design-time and runtime steps. During design time, *INDRA* uses trusted CAN message data to train a recurrent autoencoder-based model that learns the normal system behavior. At runtime, the trained recurrent autoencoder model is used to detect anomalies based on deviations from normal system behavior computed using the proposed anomaly score metric. These steps are described in greater detail in the subsequent subsections.

5.1 *Recurrent Autoencoder*

Recurrent autoencoders are powerful neural networks that are designed to behave similar to an encoder–decoder structure but can handle time-series or sequence data as inputs. They can be represented as regular feed-forward neural network-based autoencoders, with neurons that are RNN, LSTM, or GRU units (discussed in Sect. 3). Recurrent autoencoders, like regular autoencoders, have an encoder and a decoder stage. The encoder generates a latent representation of the input data in an n-dimensional space. The decoder uses this latent representation from the encoder output and attempts to reconstruct the input data with minimal reconstruction loss. In *INDRA*, we propose a new lightweight recurrent autoencoder model, which is tailored for the design of ADS to detect cyberattacks in the in-vehicle network data.

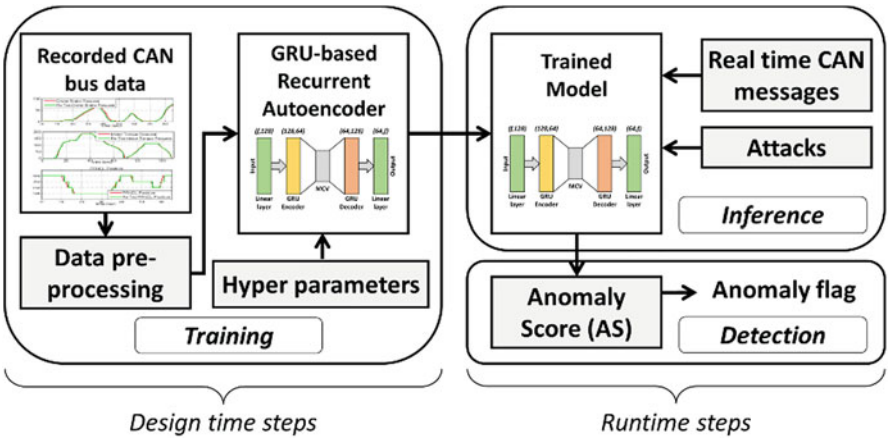


Fig. 8 Overview of *INDRA* ADS framework

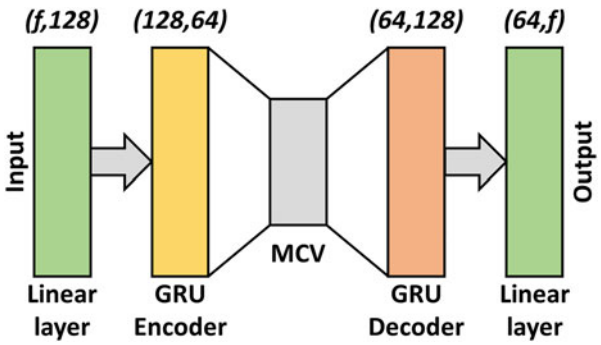


Fig. 9 Proposed model architecture of the recurrent autoencoder used in *INDRA* (f is number of features, i.e., number of signals in the input CAN message, and MCV is message context vector)

The details of the proposed model architecture and the various steps involved in its training and evaluation are discussed in the subsequent sections.

5.1.1 Model Architecture

Our proposed recurrent autoencoder model architecture with the input and output dimensions of each layer is shown in Fig. 9. The model comprises of a linear layer at the input, a GRU-based encoder, a GRU-based decoder, and a linear layer before the final output. The first linear layer receives the input time-series CAN message data with signal level values with f features (where f is the number of signals in the message). The linear layer output is passed to the GRU-based encoder, which generates the latent representation of the time-series signal inputs. This latent representation is referred to as a message context vector (MCV). The MCV captures the context of various signals in the input message in the form of a vector. Each value in the MCV can be viewed as a point in an n-dimensional space containing the context of the series of signal values provided as input. The MCV is fed into a GRU-based decoder, which is then followed by a linear layer to produce the reconstruction of the input CAN message data with individual signal values. The loss between the input and the reconstructed input is calculated using mean square error (MSE), and the weights are updated using backpropagation through time. *INDRA* designs a recurrent autoencoder model for each message ID.

5.1.2 Training Process

The training procedure starts with the preprocessing of the recorded CAN message data from a trusted vehicle. Each sample in the dataset consists of a message ID and the corresponding signal values contained within that message ID. In some cases, the range of signal values can be very large, which can make the training process

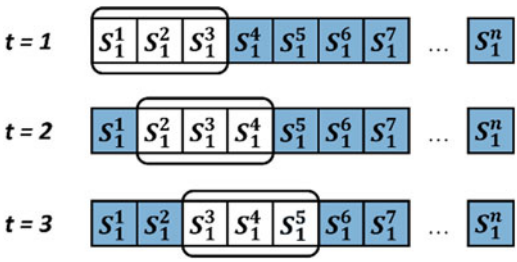


Fig. 10 Example of a rolling window approach

extremely slow or unstable. To prevent this, we scale the signal values between 0 and 1 for each signal type. Moreover, scaling signal values also helps to avoid the problem of exploding gradients (as discussed in Sect. 3).

The preprocessed CAN data is divided into training data (85%) and validation data (15%), which is then prepared for training using a rolling window-based approach. This involves choosing a fixed size window and rolling it to the right by one sample every time step. Figure 10 illustrates a rolling window of size three samples and its movement for the three consecutive time steps. The term S_i^j represents the *i*th signal value at *j*th sample. The elements in the rolling window are referred to as a subsequence, and the size of the subsequence is equal to the size of the rolling window. Our proposed recurrent autoencoder model attempts to learn the temporal relationships that exist between the series of signal values because each subsequence consists of a set of signal values over time. These signal level temporal relationships aid in detecting more complex attacks such as continuous and playback (as discussed in Sect. 4.3). The process of training using subsequences is done iteratively until the end of the training data.

Each iteration during the training process consists of a forward pass and a backward pass (using backpropagation through time to update the weights and biases of the neurons-based on the error value (as discussed in Sect. 3)). The model's performance is evaluated (forward pass only) at the end of the training using the validation data, which was not seen by the model during the training. The model has seen the complete dataset once by the end of validation, which is known as an epoch. The model is trained for a set number of epochs until the model reaches convergence. Moreover, the process of training and validation using subsequences is sped up by training the input subsequences data in groups known as mini-batches. Each mini-batch is made up of several consecutive subsequences that are given as the input to the model in parallel. The size of each mini-batch is referred to as batch size. Finally, a learning rate is defined to control the rate of update of the model parameters during backpropagation phase. These hyperparameters such as subsequence size, batch size, and learning rate are covered in detail in Sect. 6.1.

5.2 Inference and Detection

The trained model is set to evaluation mode at runtime, meaning that only forward passes are performed, and the weights are not updated. During this phase, the trained model is tested under multiple attack scenarios (mentioned in Sect. 4.3), by simulating appropriate attack condition in the CAN message dataset.

Every data sample that passes through the model is reconstructed, and the reconstruction loss is sent to the detection module, which then computes a metric called *anomaly score* (AS). The AS helps in determining whether a signal is anomalous or normal. The AS is calculated at a signal level to predict which signal is under attack. AS is computed as a squared error during each iteration of the inference to estimate the prediction deviation from the input signal value, as shown in (2).

$$\mathrm{AS}_i = \left(S_i^j - \hat{S}_i^j\right)^2 \forall i \in [1, m] \tag{2}$$

where, S_i^j denotes the ith signal value at jth sample, $\hat{S}_i^j$ represents its reconstruction, and m is the number of signals in the message. We observe a large deviation for predicted value from the input signal value (i.e., large AS value), when the current signal pattern is not seen during the training phase and a minimal AS value otherwise. This serves as the foundation for our detection phase.

Since the dataset lacks a signal level anomaly label information, *INDRA* combines the signal level AS information into a message-level AS, by calculating the maximum AS of the signals in that message as shown in (3).

$$\mathrm{MIS} = \max\left(\mathrm{AS}_1, \mathrm{AS}_2, \ldots, \mathrm{AS}_m\right) \tag{3}$$

To achieve adequate detection accuracy, the anomaly threshold (AT) for flagging messages is carefully chosen. *INDRA* investigates multiple choices for AT, using the best model from the training process. The model with the lowest running validation loss, from the training process, is defined as the best model. From this model, multiple metrics such as maximum, mean, median, 99.99%, 99.9%, 99%, and 90% validation loss are logged across all iterations as the choices for the AT. The analysis for selection of the AT metric is presented in detail in Sect. 6.2.

A working snapshot of *INDRA* ADS working in an environment with attacks is illustrated in Fig. 11a, b, with a plateau attack on a message with three signals, between time 0 and 50. Figure 11a compares the input (true) vs ADS predicted signal value comfort three signals. The attack interval is represented by the blue highlighted area. It can be observed that the reconstruction is close for almost all signals except during the attack interval for majority of the time. Signal 3 is subjected to a plateau attack in which the attacker maintains a constant value until the end of attack interval as illustrated in the third subplot of Fig. 11a (note the larger difference between the predicted and actual input signal values in that subplot,

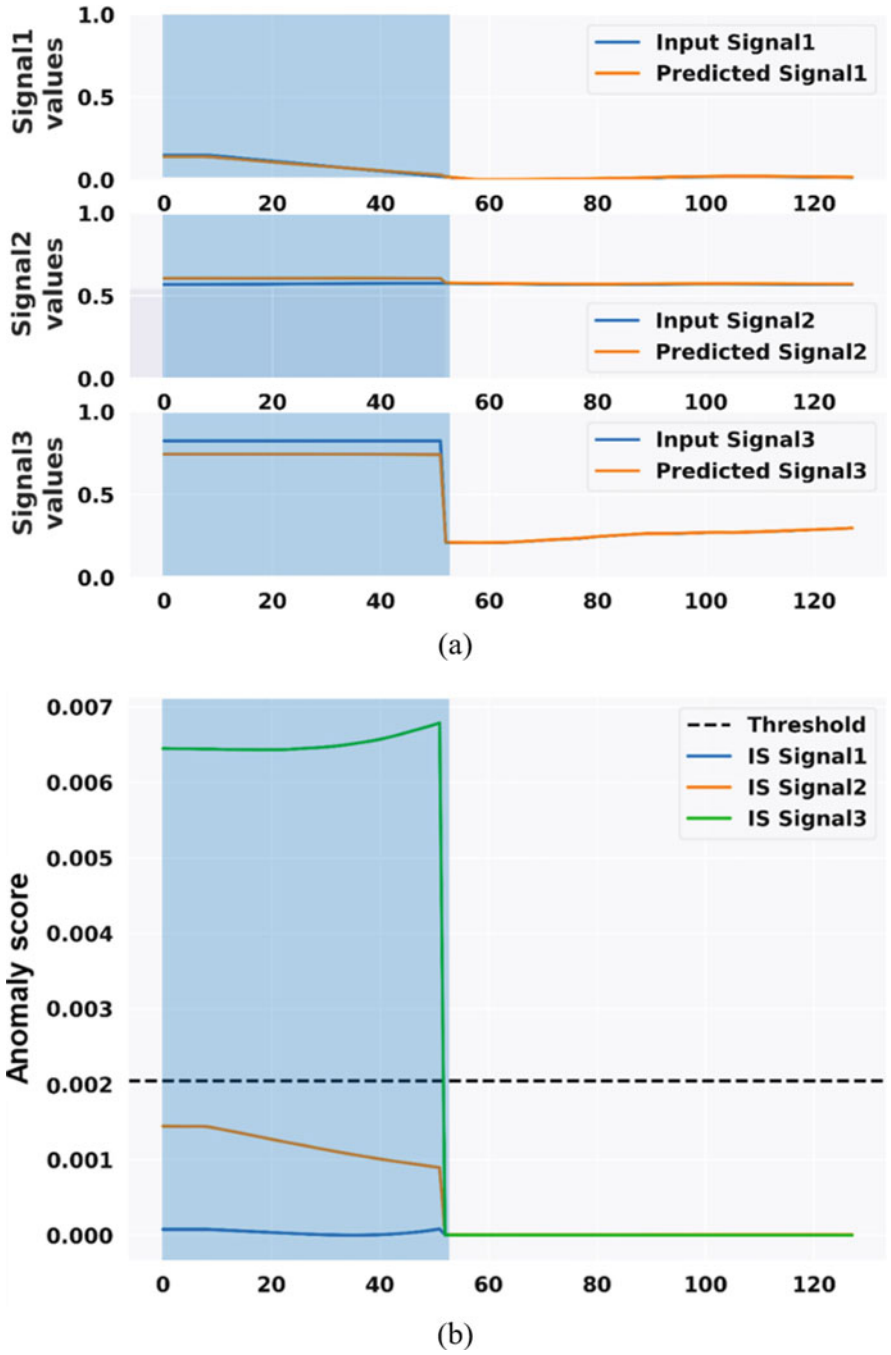


Fig. 11 Working of *INDRA* ADS checking a message with three signals under a plateau attack, where (**a**) shows the signal comparisons and (**b**) shows IS for signals and IS for the message and Anomaly flag

compared to for signals 1 and 2). Figure 11b depicts multiple signal anomaly scores for the three signals. The dotted black line represents the anomaly threshold (AT). As previously stated, the maximum of signal anomaly scores is chosen as message anomaly score (MAS), which in this case is the AS of signal 3. The anomaly score of signal 3 is above the AT, for the entire duration of the attack interval as shown in Fig. 11b, highlighting *INDRA's* ability to detect such attacks. The value of AT (equal to 0.002) in Fig. 11b is calculated using the method described in Sect. 6.2. It is important to note that this value is specific to the example case shown in Fig. 11 and is not the threshold value used for our remaining experiments. The details of AT selection technique are discussed in detail in Sect. 6.2.

6 Experiments

6.1 Experimental Setup

A series of experiments have been conducted to evaluate the performance of our proposed *INDRA* ADS. We begin by presenting an analysis for the selection of anomaly threshold (AT). The derived AT is used to contrast against two variants of the same framework known as *INDRA*-LED and *INDRA*-LD. The former removes the linear layer before the output, essentially leaving the task of decoding the context vector to GRU-based decoder. The abbreviation LED stands for (L) linear layer, (E) encoder GRU, and (D) decoder GRU. The second variation substitutes a series of linear layers for the GRU and the linear layer at the decoder (LD stands for linear decoder). These experiments were carried to assess the importance of different layers in the network. However, the encoder side of the network is not changed because it is required to generate an encoding of the time-series data. *INDRA* investigates other variants as well, but they were not included in the discussion as their performance was lesser compared with that of LED and LD variants.

Subsequently, the best *INDRA* variant is compared with three prior works: predictor LSTM (PLSTM [38]), replicator neural network (RepNet [39]), and CANet [36]. The first comparison work (PLSTM) employs an LSTM-based network that has been trained to predict the signal values in the following message transmission. PLSTM accomplishes this by taking the 64-bit CAN message payload as the input and learning to predict the signal at a bit-level granularity by minimizing prediction loss. The bit level deviations between the real and the predicted next signal values are monitored using a log loss or binary cross-entropy loss function. PLSTM uses the prediction loss values during runtime to decide whether a particular message is anomalous or not. The second comparison work (RepNet) employs a series of RNN layers to increase the dimensionality of the input data and reconstruct the signal values by decreasing back to the original dimensionality. RepNet accomplishes this by reducing the mean squared error between the input and the reconstructed signal values. At runtime, large deviations between the input received signal and

the reconstructed signal values are used to detect attacks. Finally, CANet uses a quadratic loss function to minimize the signal reconstruction error by combining multiple LSTMs and linear layers in an autoencoder architecture. All experiments conducted with *INDRA* and its variants and prior works are discussed in subsequent subsections.

The SynCAN dataset developed by ETAS and Robert Bosch GmbH [36] was used to evaluate INDRA framework with its variants and against prior works. The dataset contains CAN message data for ten different IDs that have been modeled after real-world CAN message data. Furthermore, the dataset consists of both training and test data with multiple attacks (discussed in Sect. 4.3). Each row in the dataset contains a timestamp, message ID, and individual signal values. In addition, the test data contains a label column with either 0 or 1 values indicating normal or anomalous messages. The label information is available per message basis and does not specify which signal within the message is under attack. This label information is used to evaluate the proposed ADS over several metrics such as detection accuracy and false positive rate and is discussed in detail in the next subsections. Moreover, to simulate a more realistic attack scenario in the in-vehicle networks, the test data also contains normal CAN traffic between the attack injections. *Note*: The label information in the training data is not used to train *INDRA* model, as INDRA model learns the patterns in the input data in an unsupervised manner.

All the machine learning-based frameworks including the *INDRA* framework and its variants as well as comparison works are implemented using Pytorch 1.4. *INDRA* conducts various experiments to select the best performing model hyperparameters (number of layers, hidden unit sizes, and activation functions). The final model discussed in Sect. 5.1 was trained using the SynCAN data set, with 85% of train data used for training and the remaining for validation. The validation data is primarily used to assess the model performance at the end of each epoch. The model is trained for 500 epochs, using a rolling window approach (as discussed in Sect. 5.1.2) with the subsequence size of 20 messages and the batch size of 128. Moreover, an early stopping mechanism is implemented to monitor the validation loss across epochs and stop the training process if there is no improvement after 10 (patience) epochs. The initial learning rate is chosen as 0.0001, and tanh activations are applied after each linear and GRU layers. Furthermore, ADAM optimizer is used with the mean squared error (MSE) as the loss criterion. The trained model parameters were used during testing and considered multiple test data inputs to simulate attack scenarios. The anomaly score metric (as stated in Sect. 5) was used to calculate the anomaly threshold to flag the message as anomalous or normal. To evaluate the model performance, several performance metrics such as detection accuracy and false positive rate were considered. All the simulations were executed on an AMD Ryzen-9 3900X server with an Nvidia GeForce RTX 2080Ti GPU.

Finally, before the experimental results section, we present the following definitions in the context of ADS:

- True positive (TP) – when the ADS detects an actual anomalous message as an anomaly

- False negative (FN) – when the ADS detects an actual anomalous message as normal
- False positive (FP) – when the ADS detects a normal message as an anomaly (aka false alarm)
- True negative (TN) – when the ADS detects an actual normal message as normal.

INDRA framework focuses on two key performance metrics: (i) *detection accuracy*, a measure of ADS ability to detect anomalous messages correctly, and (ii) *false positive rate*, also known as false alarm rate. These metrics are computed as shown in (4) and (5):

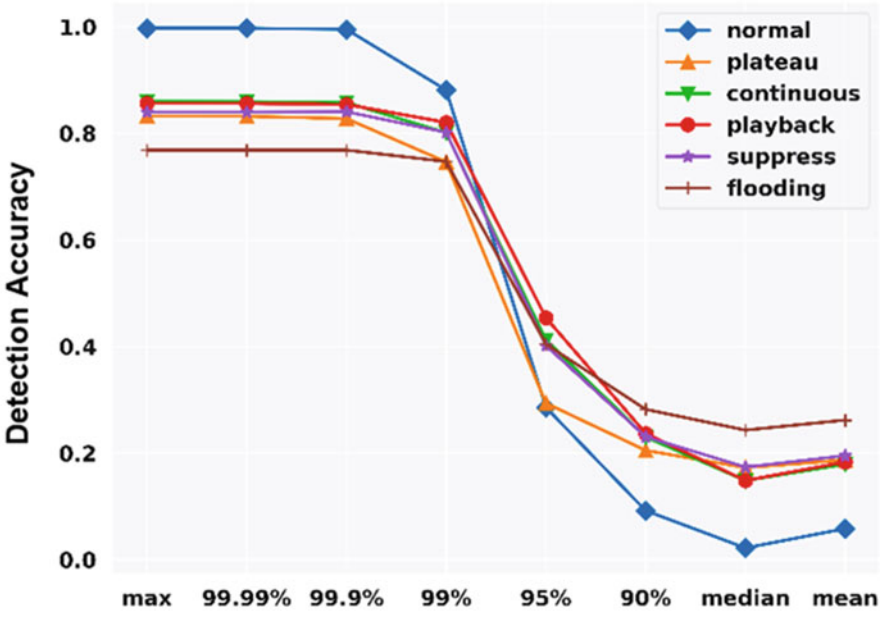
$$\text{Detection accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FN} + \text{FP} + \text{TN}} \tag{4}$$

$$\text{False positive rate} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{5}$$

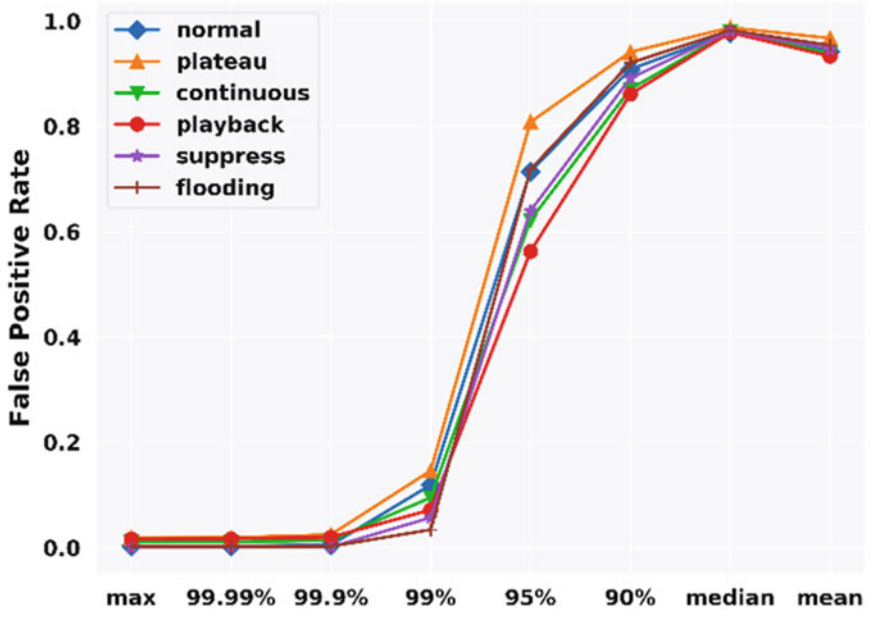
6.2 Anomaly Threshold Selection

This subsection presents a detailed analysis on the selection of anomaly threshold (AT) by considering various options such as max, median, mean, and different quantile bins of validation loss of the final model. The idea is that the model reconstruction error for the normal message should be much smaller than the error for anomalous messages. Hence, *INDRA* explores several candidate options to achieve this goal that would work across multiple attack and no-attack scenarios. A high threshold value can make it harder for the model to detect the attacks that change the input pattern minimally (e.g., continuous attack). On the other hand, having a small threshold value can cause multiple false alarms, which is highly undesirable. Hence, it becomes crucial to select an appropriate threshold value to optimize the performance of the model.

Figure 12a, b shows the detection accuracy and false positive rate, respectively, for various candidate options to calculate AT under different attack scenarios. The results from the Fig. 12 indicate that selecting higher validation loss as the AT can lead to a high accuracy and low false alarm rate. However, selecting a very high value (e.g., "max" or "99.99 percentile") may result in missing small variations in the input patterns that are found in more sophisticated attacks. We empirically conclude that the maximum and 99.99 percentile values to be very close. To capture attacks that produce small deviations, a slightly smaller threshold value is selected that would still perform similar to max and 99.99 percentile thresholds on all of the current attack scenarios. Therefore, *INDRA* chooses the 99.9th percentile value of the validation loss as the value of the anomaly threshold (AT) and uses the same AT value for the remainder of the experiments discussed in the next subsections.



(a)



(b)

Fig. 12 Comparison of (**a**) detection accuracy and (**b**) false positive rate for various choices of anomaly threshold (AT) as a function of validation loss under different attack scenarios (% refers to percentile not percentage)

6.3 Comparison of INDRA Variants

After selecting the correct anomaly threshold from the previous subsection, we use that same criterion for evaluating against two other variants: *INDRA*-LED and *INDRA*-LD. The main intuition behind evaluating different variants of *INDRA* is to investigate the impact of different types of layers in the model on the performance metrics discussed in Sect. 6.1.

Figure 13a illustrates the detection accuracy for *INDRA* framework and its variants on *y*-axis with multiple types of attacks and for a no-attack scenario (normal) on the *x*-axis. It can be clearly seen that *INDRA* outperforms the other two variants and has high accuracy in most of the attack scenarios. It should be noted that the high accuracy is achieved by monitoring at a signal level as opposed to prior works that monitors at the message level.

Figure 13b illustrates the false positive rate or false alarm rate of *INDRA* and other variants under different attack scenarios. When compared with other variants, *INDRA* has the lowest false positive rate and highest detection accuracy. Moreover, *INDRA*-LED, which is just short of a linear layer at the decoder end, is the second-best performing model after *INDRA*. The ability of *INDRA*-LED to use a GRU-based decoder helps in reconstructing the MCV back to original signals. It can be clearly seen in both Fig. 13a, b that the absence of GRU layers on the output decoder end for *INDRA*-LD results in significant performance degradation. As a result, *INDRA* is chosen as the candidate model for subsequent experiments.

6.4 Comparison with Prior Works

Our proposed *INDRA* framework is compared with some of the best-known prior works in the ADS area such as PLSTM [38], RepNet [39], and CANet [36]. Figure 14a, b shows the detection accuracy and false positive rate, respectively, for the various techniques under different attack scenarios.

From the Fig. 14a, b, it is evident that *INDRA* achieves high accuracy for each attack scenario while also achieving low positive rates. The ability to monitor signal level variations combined with more cautious selection of anomaly threshold gives *INDRA* an advantage over comparison works. PLSTM and RepNet use the maximum validation loss in the final model as the threshold, whereas CANet uses interval-based monitoring to detect anomalous messages. Choosing a higher threshold helped PLSTM to achieve slightly lower false positive rates for some scenarios, but it hurt the ability of both PLSTM and RepNet to detect attacks with minor variations in the input data. This is because the deviations produced by some of the complex attacks are small and the attacks go undetected due to the large thresholds. Moreover, CANet's interval-based monitoring struggles to find an optimal value for the thresholds. Lastly, the false positive rates of *INDRA* remain significantly low with the maximum of 2.5% for plateau attacks. It should be noted

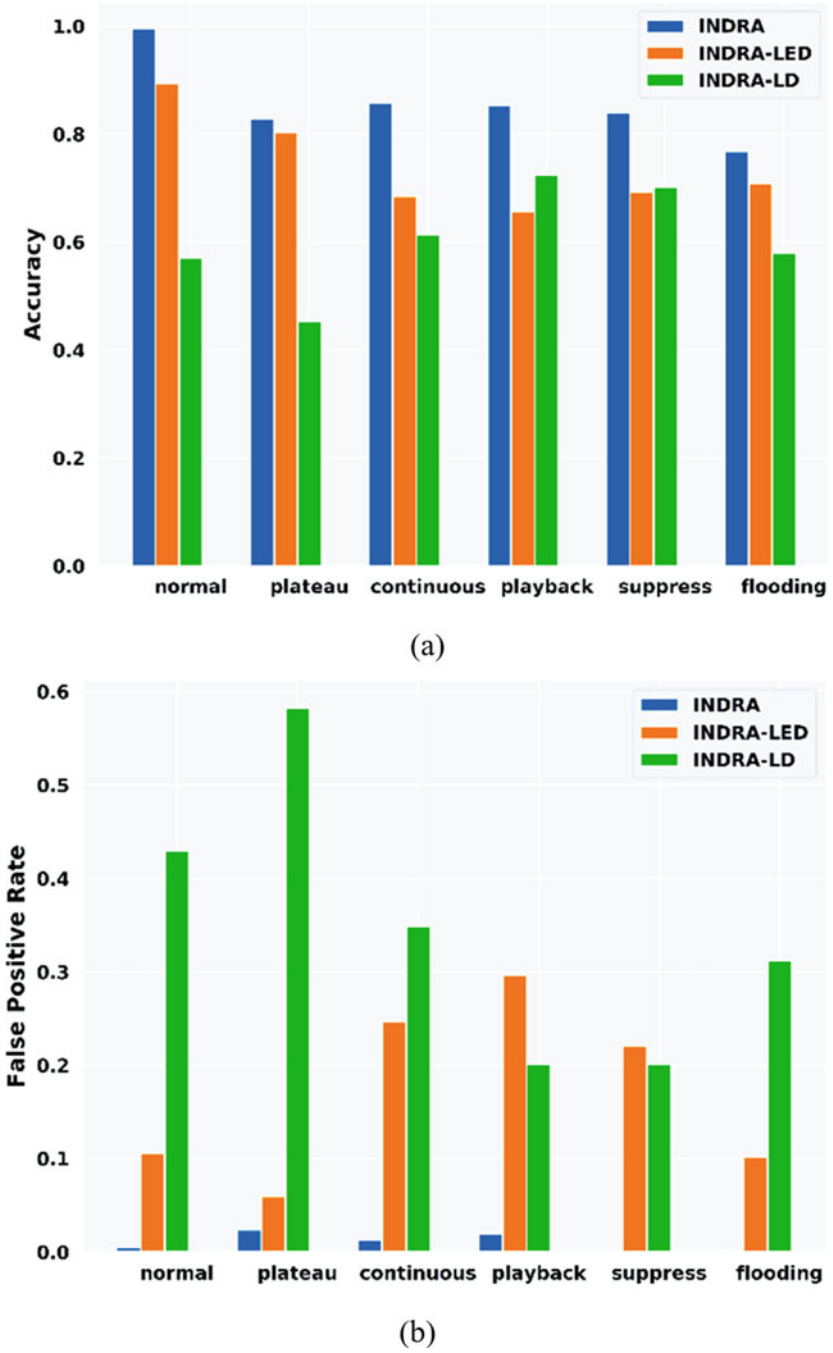


Fig. 13 Comparison of (**a**) detection accuracy and (**b**) false positive rate under different attack scenarios for *INDRA* and its variants (*INDRA*-LED and *INDRA*-LD)

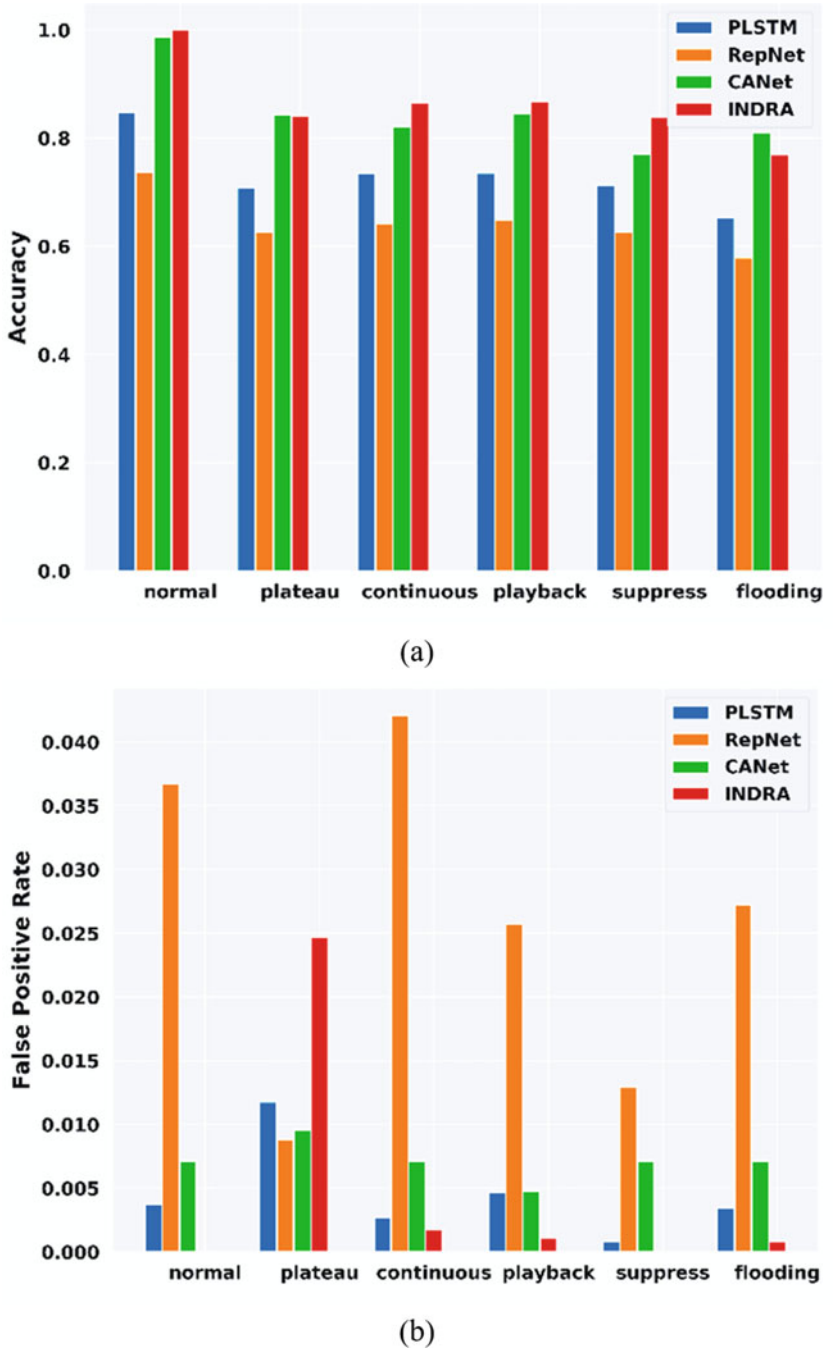


Fig. 14 Comparison of (**a**) detection accuracy and (**b**) false positive rate of *INDRA* and the prior works PLSTM [38], RepNet [39] and CANet [36]

Table 2 Memory footprint comparison between our proposed *INDRA* framework and the prior works PLSTM [38], REPNET [39], and CANET [36]

ADS framework	Memory footprint (KB)
PLSTM [38]	13,417
RepNet[39]	55
CANet [36]	8718
INDRA	443

that the *y*-axis in Fig. 14b has a much smaller scale than in Fig. 14a, and the magnitude of the false positive rate is very small.

6.5 ADS Overhead Analysis

A detailed analysis of the overhead incurred by our proposed *INDRA* ADS is discussed in this subsection. The overhead is quantified in terms of both memory footprint and time taken to process an incoming message, i.e., inference time. The former metric is important because the automotive ECUs are highly resource constrained and have limited memory and compute capacities. Therefore, having a low memory overhead is crucial to avoid interference with real-time automotive applications. The inference time metric not only provides important information about the time it takes to detect the attacks but also can be used to compute the utilization overhead on the ECU. Thus, the abovementioned two metrics are used to analyze the overhead and quantify the lightweight nature of INDRA ADS.

To accurately capture the overhead of our proposed *INDRA* framework and the prior works, we implemented the ADSs on an ARM Cortex- A57 CPU on a Jetson TX2 board, which has similar specifications to the state-of-the-art multi-core ECUs. Table 2 shows the memory footprint of *INDRA* framework and the prior works mentioned in the previous subsections. It is clear that *INDRA* framework has a low memory footprint compared with the prior works, except for the RepNet [39]. However, it is important to observe that even though *INDRA* framework has slightly higher memory footprint compared with the RepNet [39], *INDRA* outperforms all the prior works including RepNet [39] in all performance metrics under multiple attack scenarios, as shown in Fig. 14. The heavier (high memory footprint) models can capture a wide range of system behaviors; however, they are not an ideal choice for resource constrained automotive CPS. On the contrary, a much lighter model (such as RepNet) fails to capture crucial details about the system behavior due to its limited model parameters, which in turn suffers from performance issues.

In order to understand the inference overhead, we benchmarked the different ADS frameworks on an ARM Cortex- A57 CPU. In this experiment, different system configurations are considered to encompass a wide variety of ECU hardware that is available in the state-of-the-art vehicles. Based on the available hardware resources, a single core (employs only one CPU core) and dual core (employs

Table 3 Inference time comparisons between our proposed *INDRA* framework and the prior works PLSTM [38], REPNET [39], and CANET [36] using single and dual core configurations

	Average inference time (μs)	
ADS framework	Single core ARM Cortex A57 CPU	Dual core ARM Cortex A57 CPU
PLSTM [38]	681.18	644.76
RepNet [39]	19.46	21.46
CANet [36]	395.63	378.72
INDRA	80.35	72.91

two CPU cores) system configurations were selected on the Jetson TX2. The ADS frameworks are executed ten times for the different CPU configurations, and the average inference time (in μs) are recorded in Table 3. From the results in Table 3, it is evident that the *INDRA* framework has significantly faster inference times compared with the prior works (excluding RepNet) under all configurations. This is partly due to the lower memory footprint of *INDRA* framework. As previously stated, even though RepNet has a lower inference time, it has the worst performance of any compared framework, as shown in Fig. 14. The large inference times for the better performing frameworks can have an impact on the real-time performance of the control systems in the vehicle and can result in catastrophic deadline misses. We also believe that using a dedicated deep learning accelerator (DLA) further enhance the performance of the ADS models.

Thus, from Fig. 14 and Tables 2 and 3, it is clear that *INDRA* achieves a clear balance of having superior anomaly detection performance while maintaining low memory footprint and fast inference times, making it a powerful and lightweight ADS solution.

6.6 Scalability Results

In this subsection, an analysis on the scalability of *INDRA* framework is presented by studying the system performance using the ECU utilization metric as a function of increasing system complexity (number of ECUs and messages). Each ECU in the system has a real-time utilization (U_{RT}) and an ADS utilization (U_{ADS}) from running real-time and ADS applications, respectively. We primarily focus on analyzing the ADS overhead (U_{ADS}), as it is a measure of the compute efficiency of the ADS. Since the safety-critical messages monitored by the ADS are periodic in nature, the ADS can be modeled as a periodic application with period that is the same as the message period [5]. Thus, monitoring an ith message m_i results in an induced ADS utilization ($U_{\text{ADS}, mi}$) at an ECU, which can be calculated as:

$$U_{\text{IDS},m_i} = \left(\frac{T_{\text{IDS}}}{P_{m_i}} \right) \tag{6}$$

where T_{ADS} and P_{mi} denote the time taken by the ADS to process one message (inference time) and the period of the monitored message, respectively. Moreover, the sum of all ADS utilizations as a result of monitoring different messages is the overall ADS utilization at that ECU (U_{ADS}) and is given by:

$$U_{\text{IDS}} = \sum_{i=1}^{n} U_{\text{IDS},m_i} \tag{7}$$

To evaluate the scalability of INDRA, six different system sizes were considered. Moreover, a pool of commonly used message periods {1, 5, 10, 15, 20, 25, 30, 45, 50, 100} (all periods in ms) in automotive CPS is considered to sample uniformly, when assigning periods to the messages in the system. These messages are distributed evenly among different ECUs and the ADS utilization is calculated using (6) and (7). *INDRA* assumes a pessimistic scenario where all the ECUs in the system have only a single core. This would allow us to analyze the worst case overhead of the ADS.

Figure 15 shows the average ECU utilization for different system sizes denoted by $\{p, q\}$, where p is the number of ECUs and q is the number of messages in the system. In this work, a very pessimistic estimate of 50% real-time ECU utilization for real-time automotive applications ("RT Util", as shown in the dotted bars) is

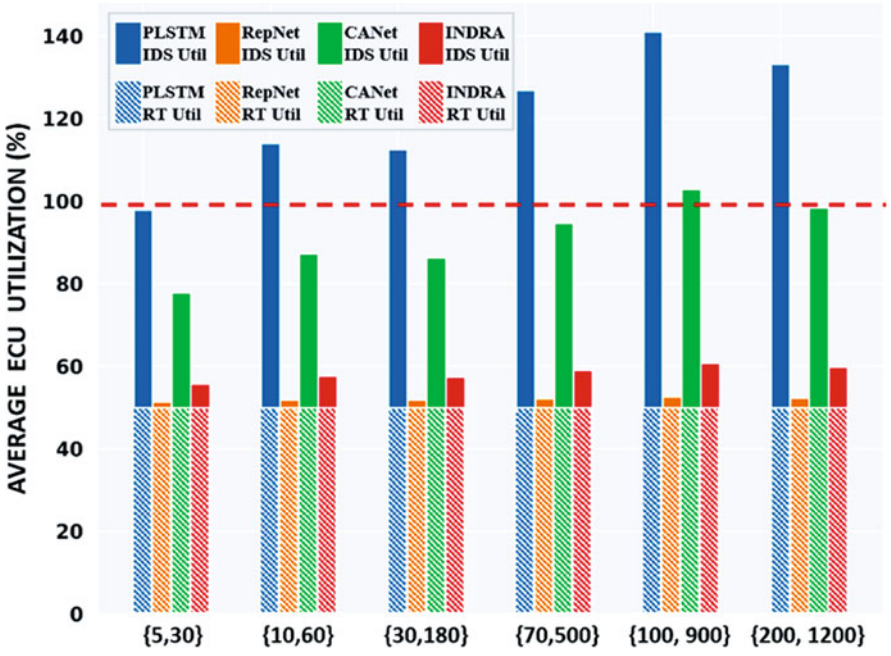


Fig. 15 Scalability analysis of our proposed *INDRA* ADS for different system sizes and the prior works PLSTM [38], RepNet [39], and CANet [36]

assumed. The solid bars on top of the dotted bars represent the overhead incurred by the ADS executing on the ECUs, and the red horizontal dotted line represents the 100% ECU utilization mark. It is critical to avoid exceeding the 100% ECU utilization limit under any scenario, as it could create undesired latencies resulting in missing deadlines, for time-critical automotive applications that can be catastrophic. It is clear from the results that the prior works such as PLSTM and CANet incur heavy overhead on the ECUs while RepNet and our proposed *INDRA* framework have very minimal overhead that is favorable to increasing system sizes. From the results in this section (Figs. 14 and 15; Tables 2 and 3), it is apparent that not only does *INDRA* achieve better performance in terms of both accuracy and low false positive rate for anomaly detection than state-of-the-art prior work but also is lightweight and highly scalable.

7 Conclusion

In this chapter, we presented a novel recurrent autoencoder-based lightweight anomaly detection system called *INDRA* for distributed automotive cyber-physical systems. *INDRA* framework uses a metric called anomaly score (AS) to measure the deviation of the prediction signal from the actual input. *INDRA* also presents a thorough analysis of our anomaly threshold selection process and compared with the best-known prior works in this area. The promising results indicate a compelling potential for utilizing our proposed approach in emerging automotive platforms.

Acknowledgments This work was supported by the National Science Foundation (NSF), through grant CNS-2132385.

References

1. Kukkala, V.K., Bradley, T., Pasricha, S.: Priority-based multi-level monitoring of signal integrity in a distributed powertrain control system. In: Proceedings of IFAC Workshop on Engine and Powertrain Control, Simulation and Modeling. IEEE (2015)
2. Kukkala, V.K., Bradley, T., Pasricha, S.: Uncertainty analysis and propagation for an auxiliary power module. In: Proceedings of IEEE Transportation Electrification Conference (TEC). IEEE (2017)
3. Kukkala, V.K., Pasricha, S., Bradley, T.: JAMS: Jitter-aware message scheduling for FlexRay automotive networks. In: Proceedings of IEEE/ACM International Symposium on Network-on-Chip (NOCS). IEEE (2017)
4. Kukkala, V.K., Pasricha, S., Bradley, T.: JAMS-SG: a framework for Jitter-aware message scheduling for time-triggered automotive networks. ACM Trans. Des. Autom. Electron. Syst. **24**(6), 1–31 (2019)
5. Kukkala, V., Pasricha, S., Bradley, T.: SEDAN: security-aware design of time-critical automotive networks. IEEE Trans. Veh. Technol. **69**(8), 9017–9030 (2020)
6. Kukkala, V.K., Thiruloga, S.V., Pasricha, S.: INDRA: intrusion detection using recurrent autoencoders in automotive embedded systems. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **39**(11), 3698–3710 (2020)

7. Kukkala, V.K., Thiruloga, S.V., Pasricha, S.: LATTE: LSTM self-attention based anomaly detection in embedded automotive platforms. ACM Trans. Embed. Comput. Syst. **20**(5s, Article 67), 1–23 (2021)
8. Thiruloga, S.V., Kukkala, V.K., Pasricha, S.: TENET: temporal CNN with attention for anomaly detection in automotive cyber-physical systems. In: Proceedings of IEEE/ACM Asia & South Pacific Design Automation Conference (ASPDAC). IEEE (2022)
9. Kukkala, V.K., Thiruloga, S.V., Pasricha, S.: Roadmap for cybersecurity in autonomous vehicles. In: IEEE Consumer Electronics Magazine (CEM). IEEE (2022)
10. Tunnell, J., Asher, Z., Pasricha, S., Bradley, T.H.: Towards improving vehicle fuel economy with ADAS. SAE Int. J. Connected Autom. Veh. **1**(2), 81 (2018)
11. Tunnell, J., Asher, Z., Pasricha, S., Bradley, T.H.: Towards improving vehicle fuel economy with ADAS. In: Proceedings of SAE World Congress Experience (WCX). SAE Technical Paper (2018)
12. Asher, Z., Tunnell, J., Baker, D.A., Fitzgerald, R.J., Banaei-Kashani, F., Pasricha, S., Bradley, T.H.: Enabling prediction for optimal fuel economy vehicle control. In: Proceedings of SAE World Congress Experience (WCX). SAE Technical Paper (2018)
13. Dey, J., Taylor, W., Pasricha, S.: VESPA: a framework for optimizing heterogeneous sensor placement and orientation for autonomous vehicles. IEEE Consum. Electron. Mag. **10**(2), 16 (2021)
14. Kukkala, V.K., Pasricha, S., Bradley, T.: Advanced driver-assistance systems: a path toward autonomous vehicles. IEEE Consum. Electron. Mag. **7**(5), 18 (2018)
15. Koscher, K., Czeskis, A., Roesner, F., Patel, S., Kohno, T., Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S.: Experimental security analysis of a modern automobile. In: Proceedings of IEEE Symposium on Security and Privacy (SP). IEEE (2010)
16. Miller, C., Valasek, C.: Remote exploitation of an unaltered passenger vehicle. Black Hat USA (2015)
17. Izosimov, V., Asvestopoulos, A., Blomkvist, O., Törngren, M.: Security-aware development of cyber-physical systems illustrated with automotive case study. In: Proceedings of IEEE/ACM Design, Automation & Test in Europe & Exhibition (DATE). IEEE (2016)
18. Studnia, I., Alata, E., Nicomette, V., Kaâniche, M., Laarouchi, Y.: A language-based intrusion detection approach for automotive embedded networks. Int. J. Embed. Syst. **10**(8), 1–12 (2018)
19. Marchetti, M., Stabili, D.: Anomaly detection of CAN bus messages through analysis of ID sequences. In: Proceedings of IEEE Intelligent Vehicle Symposium (IV). IEEE (2017)
20. 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 (2011)
21. Larson, U.E., Nilsson, D.K., Jonsson, E.: An approach to specification-based attack detection for in-vehicle networks. In: Proceedings of IEEE Intelligent Vehicles Symposium (IV). IEEE (2008)
22. Aldwairi, M., Abu-Dalo, A.M., Jarrah, M.: Pattern matching of signature-based IDS using Myers algorithm under MapReduce framework. EURASIP J. Inf. Secur. **2017**(1), 1–11 (2017)
23. Myers, E.W.: An O(ND) difference algorithm and its variations. Algorithmica. **1**, 251–266 (1986)
24. Hoppe, T., Kiltz, S., Dittmann, J.: Applying intrusion detection to automotive IT-early insights and remaining challenges. J. Inf. Assur. Secur. **4**(6), 226–235 (2009)
25. Waszecki, P., Mundhenk, P., Steinhorst, S., Lukasiewycz, M., Karri, R., Chakraborty, S.: Automotive electrical and electronic architecture security via distributed in-vehicle traffic monitoring. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **36**(11), 1790–1803 (2017)
26. Cho, K.T., Shin, K.G.: Fingerprinting electronic control units for vehicle intrusion detection. In: Proceedings of USENIX. USENIX Association (2016)
27. Ying, X., Sagong, S.U., Clark, A., Bushnell, L., Poovendran, R.: Shape of the cloak: formal analysis of clock skew-based intrusion detection system in controller area networks. IEEE Trans. Inf. Forensics Secur. **14**(9), 2300–2314 (2019)

28. Yoon, M.K., Mohan, S., Choi, J., Sha, L.: Memory heat map: anomaly detection in real-time embedded systems using memory behavior. In: Proceedings of IEEE/ACM/EDAC Design Automation Conference (DAC). IEEE (2015)
29. Müter, M., Asaj, N.: Entropy-based anomaly detection for in-vehicle networks. In: Proceedings of IEEE Intelligent Vehicles Symposium (IV). IEEE (2011)
30. Müter, M., Groll, A., Freiling, F.C.: A structured approach to anomaly detection for in-vehicle networks. In: Proceedings of IEEE International Conference on Intelligent and Advanced System (ICIAS). IEEE (2010)
31. Taylor, A., Japkowicz, N., Leblanc, S.: Frequency-based anomaly detection for the automotive CAN bus. In: Proceedings of World Congress on Industrial Control Systems Security (WCICSS). IEEE (2015)
32. Martinelli, F., Mercaldo, F., Nardone, V., Santone, A.: Car hacking identification through fuzzy logic algorithms. In: Proceedings of IEEE International Conference on Fuzzy Systems (FUZZ-IEEE). IEEE (2017)
33. Vuong, T.P., Loukas, G., Gan, D.: Performance evaluation of cyber-physical intrusion detection on a robotic vehicle. In: Proceedings of IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM). IEEE (2015)
34. Levi, M., Allouche, Y., Kontorovich, A.: Advanced analytics for connected car cybersecurity. In: Proceedings of IEEE Vehicular Technology Conference (VTC). Springer (2018)
35. Kang, M.J., Kang, J.W.: A novel intrusion detection method using deep neural network for in-vehicle network security. In: IEEE Proceedings of Vehicular Technology Conference (VTC). Springer (2016)
36. Hanselmann, M., Strauss, T., Dormann, K., Ulmer, H.: CANet: an unsupervised intrusion detection system for high dimensional CAN bus data. In: IEEE Access, vol. 8, p. 58194 (2020)
37. Loukas, G., Vuong, T., Heartfield, R., Sakellari, G., Yoon, Y., Gan, D.: Cloud-based cyber-physical intrusion detection for vehicles using deep learning. IEEE Access. **6**(1), 3491–3508 (2018)
38. Taylor, A., Leblanc, S., Japkowicz, N.: Anomaly detection in automobile control network data with long short-term memory networks. In: Proceedings of IEEE International Conference on Data Science and Advanced Analytics (DSAA). IEEE (2016)
39. Weber, M., Wolf, G., Sax, E., Zimmer, B.: Online detection of anomalies in vehicle signals using replicator neural networks. In: Proceedings of ESCAR USA (2018)
40. Weber, M., Klug, S., Sax, E., Zimmer, B.: Embedded hybrid anomaly detection for automotive can communication. In: Embedded Real Time Software and Systems (ERTS) (2018)
41. Schmidhuber, J.: Habilitation thesis: system modeling and optimization (1993)
42. Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J.: Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In: A Field Guide to Dynamical Recurrent Neural Networks. IEEE Press (2001)
43. Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint, arXiv:1406.1078 (2014)
44. DiDomenico, G.C., Bair, J., Kukkala, V.K., Tunnell, J., Peyfuss, M., Kraus, M., Ax, J., Lazarri, J., Munin, M., Cooke, C., Christensen, E.: Colorado State University EcoCAR 3 final technical report. In: SAE World Congress Experience (WCX). SAE Technical Paper (2019)