Real-Time Intrusion Detection in Automotive Cyber-Physical Systems with Recurrent Autoencoders

Vipin Kumar Kukkala, Sooryaa Vignesh Thiruloga, and Sudeep Pasricha

1 Introduction

Modern-day vehicles are highly sophisticated cyber-physical systems (CPS) that consist of multiple interconnected embedded systems known as Electronic Control Units (ECUs). The ECUs run various real-time automotive applications that control different vehicular subsystem functions. Moreover, ECUs are distributed across the vehicle and communicate with each other using the in-vehicle network. In recent years, the number of ECUs being integrated into the vehicles and the complexity of software running on these ECUs has been rapidly increasing to enable various state-of-the-art Advanced Driver Assistance Systems (ADAS) features such as adaptive cruise control, lane keep assist, collision avoidance, and blind spot warning. This 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 increased complexity of modern-day vehicles has led to various complex challenges that pose a serious threat to the reliability [1–4], security [5–9], and real-time control of automotive systems [10–13].

V. K. Kukkala (✉)
NVIDIA, Santa Clara, CA, USA
e-mail: vipin.kukkala@colostate.edu

S. V. Thiruloga
Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO, USA
e-mail: sooryaa@colostate.edu

S. Pasricha
Colorado State University, Fort Collins, CO, USA
e-mail: sudeep.pasricha@colostate.edu

V. K. Kukkala, S. Pasricha (eds.), *Machine Learning and Optimization Techniques for Automotive Cyber-Physical Systems*, https://doi.org/10.1007/978-3-031-28016-0_10

Today's vehicles heavily rely on information from various external systems that utilize advanced communication standards such as 5G technology and Vehicle-to-X (V2X) [14] to support various ADAS functionalities. Unfortunately, this makes automotive embedded systems highly vulnerable to various cyber-attacks that can have catastrophic consequences. The cyber-attacks on vehicles discussed in [15–17] have presented different ways to gain unauthorized access to the in-vehicle network and override the vehicle controls by injecting malicious messages. With connected and autonomous vehicles (CAVs) on the horizon, these security concerns will get further aggravated and become a serious threat to the safety of future autonomous vehicles. Therefore, it is crucial to prevent unauthorized access to in-vehicle networks by external attackers to ensure the security of automotive CPS.

Traditional computer networks utilized firewalls to defend the networks from external attackers. However, no firewall is foolproof, and no network can be fully secure from attackers. Thus, there is a need for an active monitoring system that continuously monitors the network to identify malicious messages in the system. These systems are commonly referred to as intrusion detection systems (IDS). An IDS that is deployed in a vehicle can be used to continuously monitor the in-vehicle network traffic and trigger alerts when suspicious messages or known threats are detected. Thus, IDS acts as the last line of defense in automotive CPSs.

At a high level, IDSs are categorized into two types: *(i) rule-based* and *(ii) machine learning based*. Rule-based IDSs look for traces of previously observed attack signatures in the network traffic, whereas machine learning-based IDSs observe for the deviation from the learned normal system behavior to detect cyber-attacks. Rule-based IDS can have faster detection rates and very few false alarms (false positive rate) but are limited to detecting only previously observed attacks. On the other hand, machine learning-based IDS can detect both previously observed and novel attacks but can suffer from relatively slower detection times and higher false alarm rates. An efficient IDS needs to be lightweight (have minimal overhead), robust, and highly scalable. More importantly, practical IDSs need to have comprehensive attack coverage (i.e., detect both known and unknown attacks) with high detection accuracy and low false alarms, as recovering from false alarms can be costly.

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

In this chapter, we present a novel IDS framework called *INDRA*, first introduced in [6], that monitors the in-vehicle network messages in a Controller Area Network (CAN) based automotive CPS to detect various cyber-attacks. During the offline phase, *INDRA* uses a deep learning-based recurrent autoencoder model to learn the normal system behavior in an unsupervised manner. At runtime, *INDRA* continuously monitors the in-vehicle network for deviations from learned normal

system behavior to detect malicious messages. Moreover, *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 proposed an intrusion score (IS) metric to measure deviation from the normal operating system behavior;
3. We presented a comprehensive analysis of the selection of thresholds for the intrusion score metric;
4. Lastly, we compared our proposed *INDRA* framework with the best-known prior works in the area to demonstrate its effectiveness.

2 Related Work

Several techniques have been proposed in the literature to design IDS for protecting time-critical automotive CPS. These works try to detect various attacks by monitoring the in-vehicle network traffic. In this section, we first discuss the key rule-based IDSs and then discuss machine learning based IDSs.

Rule-based IDS detects known attacks by using the information from previously observed attack signatures. In [18], a language theory-based model was introduced to derive attack signatures. However, this technique fails to detect attacks when it misses the packets transmitted during the early stages of the attack interval. A transition matrix-based attack detection scheme for CAN bus systems was proposed in [19], but this approach only works for simple attacks and fails to detect advanced replay attacks. In [20], the authors identified key attack signatures such as increased message frequency and missing messages to detect cyber-attacks. In [21], the authors proposed a specification-based approach to detect cyber-attacks, which analyzes the system behavior and compares it with the predefined attack patterns to detect anomalies. However, their approach fails to detect unknown attacks. The authors in [22] propose an ADS technique using the Myers algorithm [23] under the map-reduce framework. A time-frequency analysis of CAN messages is used to detect multiple anomalies in [24]. In [25], the authors analyzed message frequency at design time to derive a regular operating mode region, which is used as a baseline during runtime to detect cyber-attacks. In [26], the sender ECU's clock skew, and the messages are fingerprinted at design time and used at runtime to detect attacks by observing for variations. The authors in [27] presented a formal analysis of clock-skew-based IDS and evaluated it on a real vehicle. In [28], a memory heat map is used to characterize the memory behavior of the operating system to detect anomalies. An entropy-based IDS that observes the change in system entropy to detect anomalies was proposed in [29]. Nonetheless, the technique fails to detect complex attacks for which the entropy change is minimal. In conclusion, rule-

based IDSs offer a fast solution to the intrusion detection problem with lower false positive rates but fail to detect more complex and novel attacks. Moreover, obtaining signatures of every possible attack pattern is not practical.

On the other hand, machine learning-based IDSs 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 IDS that utilizes attack detection sensors in the vehicle to monitor various system events and 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 IDS was introduced, but it suffers from poor detection latency. In [32], an ensemble of different nearest neighbor classifiers was used to distinguish between normal and an attack-induced CAN messages. A decision-tree-based detection model to monitor the physical features of the vehicle was proposed in [33] to detect cyber-attacks. However, this model is impractical 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 cyber-attacks. In [35], a deep neural network-based approach was proposed to scan the messages in the in-vehicle network to detect attacks. This approach is finetuned 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 IDS for multi-message ID detection was proposed. However, due to the high complexity of model architecture, this approach has a high computational overhead on the ECUs. In [37], an LSTM-based IDS was proposed to detect insertion and dropping attacks (explained in Sect. 4.3). In [38], an LSTM-based predictor model is proposed to predict the next time step message value at a bit level and observe for large variations to detect anomalous messages. A recurrent neural network (RNN) based IDS to learn the normal CAN message pattern in the in-vehicle network is proposed in [39]. A hybrid IDS was proposed in [40], 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 learning 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 IDS using gated recurrent units (GRUs) to monitor the in-vehicle network messages at a signal level to detect various 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 shown in Table 1.

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

Technique	Performance metrics			
	Lightweight model	Low false positive rate	High detection accuracy	Fast inference time
PLSTM [25]	X	✓	X	X
RepNet [26]	✓	X	X	✓
CANet [23]	X	✓	✓	X
INDRA	✓	✓	✓	✓

3 Background on Sequence Learning

The availability of increased compute power from GPUs, and custom hardware accelerators enabled the training of deep neural networks with many hidden layers, which led to the creation of powerful models for solving complex problems in many domains. One such problem is detecting cyber-attacks in automotive CPS. In an automotive CPS, the communication between ECUs occurs in a time-dependent manner. Therefore, the temporal relationship between the messages in the system can be exploited in order to detect cyber-attacks. However, this cannot be achieved using traditional feedforward neural networks as the output of any input at any instance is independent of the other inputs. This makes sequence models appropriate 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 current and prior inputs. The recurrent neural network (RNN), which was introduced in [41], is an example of such a sequence model. Other sequence models, such as gated recurrent unit (GRU) and long short-term memory (LSTM), have also become popular in recent years.

3.1.1 Recurrent Neural Networks (RNN)

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

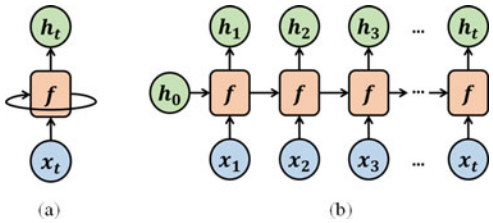


Fig. 1 (**a**) A single RNN unit and (**b**) RNN unit unrolled in time; f is the RNN unit, x is the input, and h represents hidden states

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

$$h_t = f\left(Wx_t + Uh_{t-1} + b\right) \tag{1}$$

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. As RNNs and other sequence models handle time-series data or sequences as inputs, backpropagation happens through various time steps (commonly known as backpropagation through time (BPTT)). During the BPTT step, the feedback loop in RNNs causes the errors to expand or shrink rapidly thereby creating exploding or vanishing gradients respectively. This destroys the information in backpropagation and makes the training process obsolete. Moreover, the 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 use cell state, hidden state information, and multiple gates to capture long-term dependencies between messages. The cell state can be visualized as a freeway that carries relevant information throughout the processing of an input sequence. The cell state stores information from previous time steps and passes it to the subsequent time steps to reduce the effects of short-term memory. Moreover, the information in the cell state is modified by the gates in the LSTM unit, which helps the model in determining which information should be retained and which should be ignored.

An LSTM unit contains 3 gates: *(i)* forget gate (f_t) *(ii)* output gate (o_t), and *(iii)* input gate (i_t) as shown in Fig. 2a. The forget gate is a binary gate that determines which information to retain from the previous cell state (c_{t-1}). The output gate uses information from the previous two gates to produce an output. Lastly, the input gate

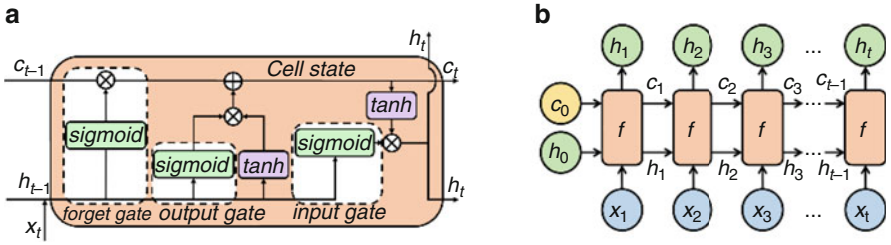


Fig. 2 (**a**) A single LSTM unit with different gates and activations, and (**b**) LSTM unit unrolled in time; f is an LSTM unit, x is input, c is cell state, and h is the hidden state

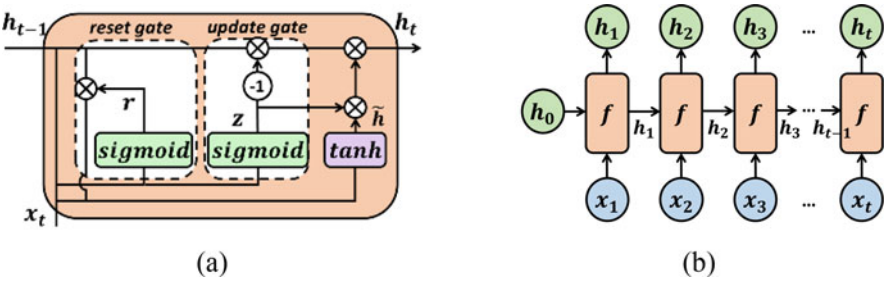


Fig. 3 (**a**) A single GRU unit with different gates and activations, and (**b**) GRU unit unrolled in time; f is a GRU unit, x is input, and h is the hidden state.

adds relevant information to the cell state (c_t). An illustration of 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 computationally expensive due to the complex sequence path from having multiple gates (compared to RNNs), and require more runtime memory. Moreover, training LSTMs have a high computation overhead even when advanced training methods such as truncated backpropagation are employed. To overcome the above-mentioned limitations, a simpler sequence model called gated recurrent unit (GRU) was introduced in [43]. GRUs can be trained faster than LSTMs and also capture dependencies in long sequences with minimal overhead (in both memory and runtime) while solving the vanishing gradient problem.

3.1.3 Gated Recurrent Unit (GRU)

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

A GRU unit consists of 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 information 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 and have a low memory overhead than LSTMs as they achieve this using fewer gates and states. It is highly crucial to use lightweight machine learning models when working with automotive systems, as real-time automotive ECUs are highly resource-constrained embedded systems with strict energy and power budgets. This makes GRU-based networks an ideal fit for inference in resource-constrained automotive systems. Thus, *INDRA* utilizes a lightweight GRU-based model to implement the IDS (explained in detail in Sect. 5).

One of the significant advantages 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, high variability in the messages between vehicle models from the same manufacturer, and the proprietary nature of this information make it highly challenging and tedious to label messages correctly. However, 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 a special class of neural networks that try to reconstruct the input by learning the latent input features in an unsupervised fashion. They achieve this by encoding the input data (x) to a hidden layer which produces the embedding, and finally decoding the embedding to produce a reconstruction $\tilde{x}$ (as shown in Fig. 4). The layers used to create this embedding are called the encoder, and the

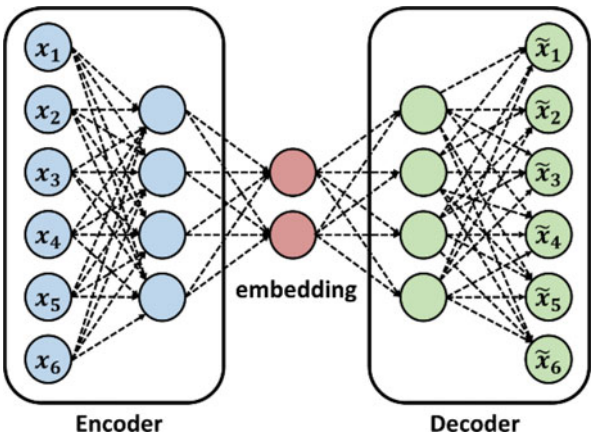


Fig. 4 An autoencoder network with encoder, decoder, and embedding layers

layers used in reconstructing the embedding into the original input (decoding) are called the decoder. During the training process, the encoder tries 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 achieve this using various 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 the key features from the inputs with the goal of minimizing reconstruction loss. The most commonly used loss functions in autoencoders include mean squared error (MSE) and Kullback-Leibler (KL) divergence.

As autoencoders aim to reconstruct the input by learning the underlying distribution of the input data, they are an excellent choice for efficiently learning and re-constructing 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 an in-vehicle network, as shown in Fig. 5. Each ECU in the system runs a specific set of automotive applications that are hard-real time in nature (i.e., they have strict timing and deadline constraints). Moreover, we assume that each ECU also runs intrusion detection applications (IDS) that are responsible for monitoring and detecting cyber-attacks in the in-vehicle network.

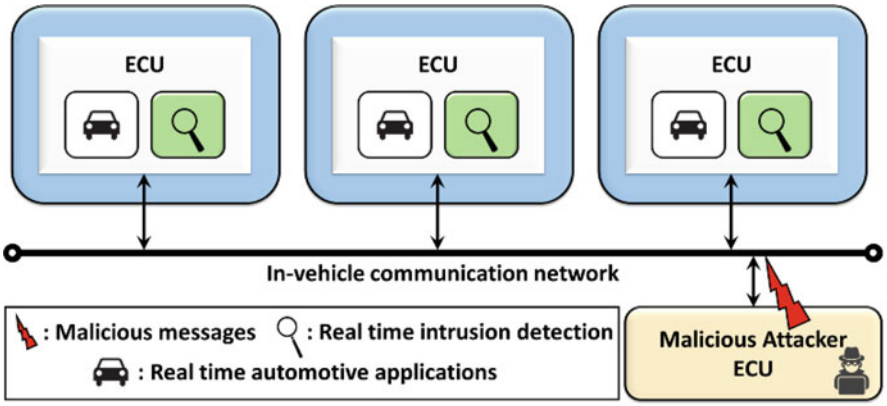


Fig. 5 Overview of the system model considered in *INDRA*

In the *INDRA* framework, we consider a distributed IDS approach (where intrusion detection applications are collocated with automotive applications) as opposed to a centralized IDS approach in which one central ECU handles all intrusion detection tasks due to the following reasons:

- A centralized IDS approach is prone to single-point failures, which can completely expose the system to the attacker.
- In extreme scenarios such as during a flooding attack (explained in Sect. 4.3), the in-vehicle network would get highly congested, and the centralized system might not be able to communicate with the victim ECUs.
- If an attacker successfully tricks the centralized IDS ECU, the attacks can go undetected by the other ECUs, compromising the entire system; however, in the case of a distributed IDS, it requires fooling multiple ECUs (which is more difficult) to compromise the system. Moreover, the decentralized intelligence in a distributed IDS scenario can still detect the attacks, even if one of the ECU is compromised.
- In a distributed IDS, ECUs can stop accepting messages as soon as an intrusion is detected. This results in significantly faster reaction times as there is no need for a notification from a centralized system.
- Lastly, in a distributed IDS, the computation load of IDS is divided 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.

Distributed IDS approach has been adopted in many state-of-the-art works, such as [18, 25], for the above-mentioned reasons. Moreover, with the increasing computation power of automotive ECUs, the collocation of IDS applications with real-time automotive applications in a distributed manner should not be a problem if the IDS has minimal overhead. *INDRA* framework is not only lightweight but also highly scalable and achieves superior intrusion detection performance (discussed in detail in Sect. 6).

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

- *Lightweight*: Intrusion detection tasks can incur additional overhead on ECU, which can have a broad range of impact ranging from poor application performance to catastrophic events due to missed deadlines for real-time applications. Therefore, *INDRA* aims to have a lightweight IDS that incurs minimal overhead on the ECU.
- *Coverage*: This is defined as the range of attacks that an IDS can detect. A good IDS must be capable of detecting more than one type of attack. Moreover, high coverage for IDS will make the system resilient to multiple attack surfaces.
- *Few false positives*: This is a highly desired quality in any IDS (even outside of the automotive domain), as dealing with false positives can quickly become costly. Thus, a good IDS is expected to have few false positives or false alarms.

- *Scalability*: As the number of ECUs in emerging vehicles is growing along with software and network complexity, this is an essential requirement. A good IDS should be highly scalable and capable of supporting multiple system sizes.

4.2 Communication Model

In this subsection, we discuss the communication model that was considered for the *INDRA* framework. *INDRA* primarily focuses on detecting anomalies in Controller Area Network (CAN) bus-based automotive CPS, as CAN is the most commonly used in-vehicle network protocol. CAN offers a low-cost, lightweight, event-triggered communication where messages are transmitted in the form of frames. A standard CAN frame structure with the length of each field (in bits) on the top is shown in Fig. 6. The standard CAN frame consists of *(i)* header, *(ii)* payload, and *(iii)* trailer segments. The header contains information about 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. More recently, a new variation of the CAN protocol, called CAN-extended or CAN 2.0B, is also being deployed increasingly in modern vehicles. The key difference is that CAN-extended has a 29-bit identifier which allows for a greater number of message IDs.

The *INDRA* IDS focuses on monitoring the payload of the CAN frame and observes for anomalies within the payload segment to detect cyberattacks. This is because most modern-day attacks involve an attacker modifying the payload to accomplish malicious activities. On the other hand, if an attacker targets the header or trailer segments, the message would get rejected at the receiver. The typical payload segment of a CAN message comprises of multiple data entities called signals. Figure 7 illustrates a real-world example CAN message with the list of signals within the message. Each signal has a particular data type, fixed size (in bits), and a start bit which specifies the signal's location in the 64-bit payload segment of the CAN frame.

INDRA focuses on monitoring individual signals within CAN payload to observe for anomalies and detect attacks. During training, *INDRA* learns the temporal relationships between the messages at a signal level and observes for deviations at runtime to detect attacks. This ability to detect attacks at a signal level enables

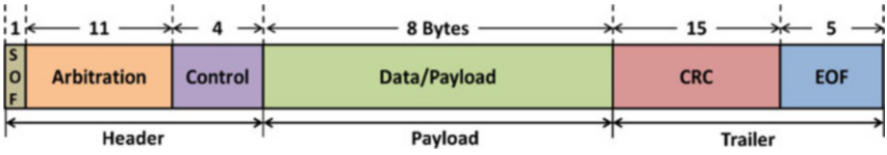


Fig. 6 Standard frame format of a Controller Area Network (CAN) message

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 A real-world example CAN message with signal information [44]

INDRA to not only detect the presence of an attacker but also help in identifying the signal under attack. This can provide valuable information related to the attack and help in understanding the intentions of the attacker, which can be used to initiate appropriate countermeasures. The signal level monitoring technique employed in *INDRA* IDS is discussed in detail in Sect. 5.2.

Note: Even though the *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* IDS aims to protect the vehicle from various types of state-of-the-art attacks that are most commonly seen and difficult to detect in automotive CPS. Moreover, these attacks have been widely studied in literature to evaluate IDSs.

1. *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 change in signal value and the duration for which the signal magnitude is changed. Larger changes in signal values are easier to detect compared to shorter changes.
2. *Flooding attack*: This is the most common and simple to launch attack, as it requires no knowledge of the system. In this attack, the attacker continuously floods the in-vehicle network with random or specific messages with the goal of preventing other ECUs from accessing the bus and rendering the bus unusable. These attacks are typically detected by the gateways and network bridges in the vehicle and often do not reach the last line of defense (the IDS). However, it is crucial to consider these attacks as they can have serious security and safety consequences when poorly handled.
3. *Playback attack*: In this attack, the attacker attempts to trick the IDS by replaying a valid series of message transmissions from the past. This attack is hard to detect if the IDS lacks the ability to capture the temporal relationships between messages and detect when they are violated.

4. *Continuous attack*: In this attack, an attacker gradually overwrites the signal value to some target value while avoiding the activation of an IDS. These attacks are difficult to detect and can be sensitive to the IDS parameters (discussed in Sect. 5.2).
5. *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 when they disrupt message transmission for long durations but are harder to detect for shorter durations.

Moreover, in this work, we assume that the attacker can gain access to the in-vehicle network using the most common attack vectors, such as connecting to the OBD-II port, connecting to V2X systems that communicate with the outside world (for e.g., infotainment and connected ADAS systems), probe-based snooping on the in-vehicle bus, and by replacing an existing ECU with a malicious 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.

Objective The goal of our proposed *INDRA* framework is to implement a lightweight IDS that can detect a variety of attacks (discussed above) in a CAN bus-based automotive CPS, with a high detection accuracy and low false positive rate while having a large attack coverage.

5 *INDRA* Framework Overview

INDRA framework utilizes a machine learning-based signal level IDS for monitoring real-time CAN messages in automotive CPS. An overview of the *INDRA* framework is depicted in Fig. 8. The *INDRA* framework consists of design-time and runtime steps. During design time, *INDRA* uses CAN message data from a trusted vehicle to train a recurrent autoencoder-based model to learn the normal system behavior. At runtime, the trained recurrent autoencoder model observes for the deviation from the learned normal system behavior using the proposed intrusion score metric to detect cyberattacks. These steps are described in detail in the subsequent subsections.

5.1 Recurrent Autoencoder

Recurrent autoencoders are powerful neural networks that are similar to an encoder-decoder structure but can handle time-series or sequence data inputs. They typically consist of units such as RNNs, LSTMs, or GRUs (discussed in Sect. 3). Similar to regular autoencoders, recurrent autoencoders have an encoder and a decoder stage. The encoder generates a latent representation of the input data in an n-dimensional

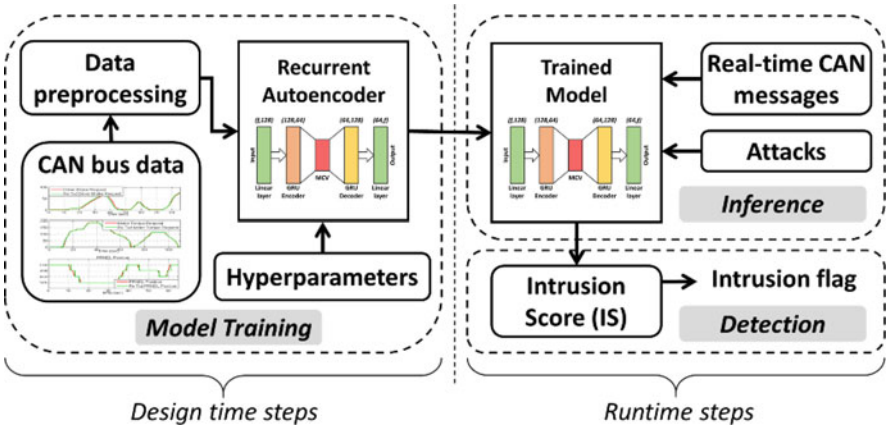


Fig. 8 Overview of the *INDRA* IDS framework

space, and the decoder uses this latent representation from the encoder output and tries to reconstruct the input data with minimal reconstruction loss. In *INDRA*, we propose a novel lightweight recurrent autoencoder model that is tailored for the design of IDS to detect cyberattacks in the in-vehicle network. The details of the proposed neural network 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 is illustrated in Fig. 9, with each layer's input and output dimensions on the top. 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 input time-series CAN message data with signal level information consisting of f features (where f is the number of signals in the message) is given as input to the first linear layer. The output from the first linear layer is then passed to the GRU-based encoder, which generates the latent representation of the time-series signal inputs, which is referred to as a message context vector (MCV) in this chapter. The MCV captures the context of various signals in the input message as a vector. Each 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 generate 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.

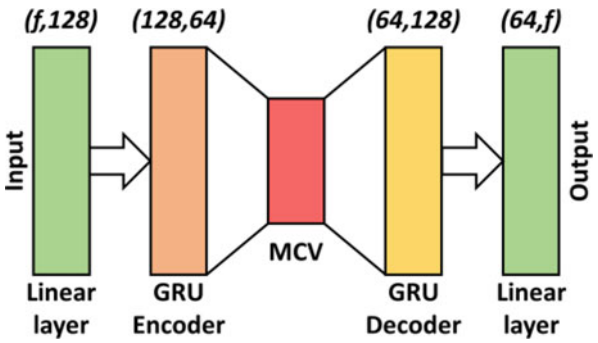


Fig. 9 Proposed recurrent autoencoder model used in *INDRA* (*f* is the number of features, i.e., number of signals in the input CAN message, MCV is message context vector)

5.1.2 Training Process

The training procedure starts with pre-processing the CAN message data collected from a trusted vehicle. Each sample in the CAN message dataset consists of a message ID and the corresponding signal values contained within that message ID. As signals represent a wide variety of information in the vehicle, the range of signal values can also be very large. This can make the training process extremely slow or unstable. To prevent this, we scale the signal values between 0 to 1 for each signal type. Moreover, scaling signal values also helps to avoid the problem of exploding gradients (as discussed in Sect. 3).

The pre-processed CAN dataset is divided into training data (85%) and validation data (15%), which is then prepared for training. We use a rolling window-based approach, which involves choosing a fixed-size window and rolling it to the right by one sample every time step. An example rolling window approach with a window size of three samples and its movement for the three consecutive time steps is illustrated in Fig. 10. The term S_i^j represents the *ith* signal value at *jth* sample. The elements in the rolling window are referred to as a subsequence, and the size of the rolling window is defined as the subsequence length. As each subsequence consists of a set of signal values over time, our proposed recurrent autoencoder model attempts to learn the temporal relationships between the series of signal values. 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 training iteration 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)). At the end of the training, the model's performance is evaluated (forward pass only) using the validation data, which was not seen by the model during the training. The end of the validation step marks the completion of one epoch during which the model has seen the complete dataset once. The model is trained for a set number of epochs until the model reaches

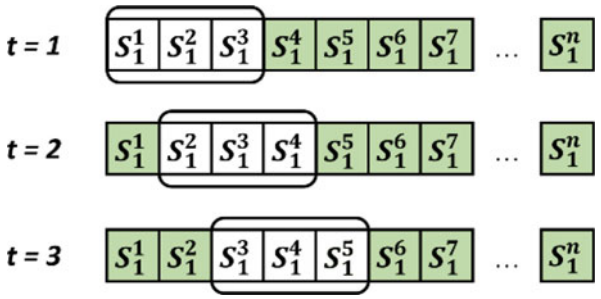


Fig. 10 An example of a rolling window approach and its movement for three consecutive time steps

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 a batch size. Lastly, to control the rate of update of the model parameters during the backpropagation phase, a learning rate is defined. These hyperparameters, such as subsequence size, batch size, learning rate, etc., are covered in detail in Sect. 6.1.

5.2 Inference and Detection

At runtime, the trained model is set to evaluation mode, where only forward passes are performed, and the weights are not updated. During this phase, various attack conditions are simulated in the CAN message dataset, and the trained model is tested under multiple attack scenarios (mentioned in Sect. 4.3).

During inferencing, each data sample that passes through the model is reconstructed, and the reconstruction loss is computed. This reconstruction loss is sent to the detection module to compute the proposed *intrusion score* (IS) metric, which helps in determining whether a signal is malicious or normal. The IS is calculated at a signal level to predict which signal is under attack. The IS is calculated as a squared error during each iteration of the inference to estimate the prediction deviation from the input signal value, as shown in (2).

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

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 IS value) when the current

signal pattern is not seen by the model during the training phase and a smaller IS value otherwise. This serves as the foundation for our detection phase.

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

$$MIS = \max(IS_1, IS_2 \ldots, IS_m) \tag{3}$$

To achieve adequate detection accuracy, it is critical to choose the intrusion threshold (IT) for flagging messages. *INDRA* investigates multiple choices for IT, using the best model (model with the lowest running validation loss) from the training phase. From this model, multiple metrics such as maximum, mean, median, 99.99%, 99.9%, 99%, and 90% validation loss are recorded across all iterations as the potential choices for the IT. The analysis for the selection of the IT metric is presented in detail in Sect. 6.2.

A working snapshot of *INDRA* IDS 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. IDS predicted signal value for three signals, and the attack interval is highlighted in red. It can be observed that the reconstruction is close for almost all signals except during the attack interval for the majority of the time. Signal 3 is subjected to a plateau attack in which the attacker maintains a constant value until the end of the attack interval. This is illustrated in the third subplot of Fig. 11a (note the larger difference between the predicted and actual input signal values in that subplot, compared to signals 1 and 2). Figure 11b depicts the signal intrusion scores for all three signals, and the dotted black line represents the intrusion threshold (IT). As stated previously, the maximum of signal intrusion scores is chosen as message intrusion score (MIS), which in this case is the IS of signal 3. As seen in Fig. 11b, the intrusion score of signal 3 is above the IT for the entire duration of the attack interval, which clearly highlights *INDRA's* ability to detect such attacks. The value of IT (equal to 0.002) in Fig. 11b is calculated using the method discussed in Sect. 6.2. However, it is important to note that this value is specific to the example case shown in Fig. 11 and is not the IT value used for the remaining experiments. The details of IT selection is discussed in detail in Sect. 6.2.

6 Experiments

6.1 Experimental Setup

A series of experiments have been conducted to evaluate the real-time performance of our proposed *INDRA* IDS. We begin by presenting an analysis for the selection of intrusion threshold (IT). The derived IT is used to contrast against two variants of the same framework known as *INDRA*-LED and *INDRA*-LD. The *INDRA-LED*

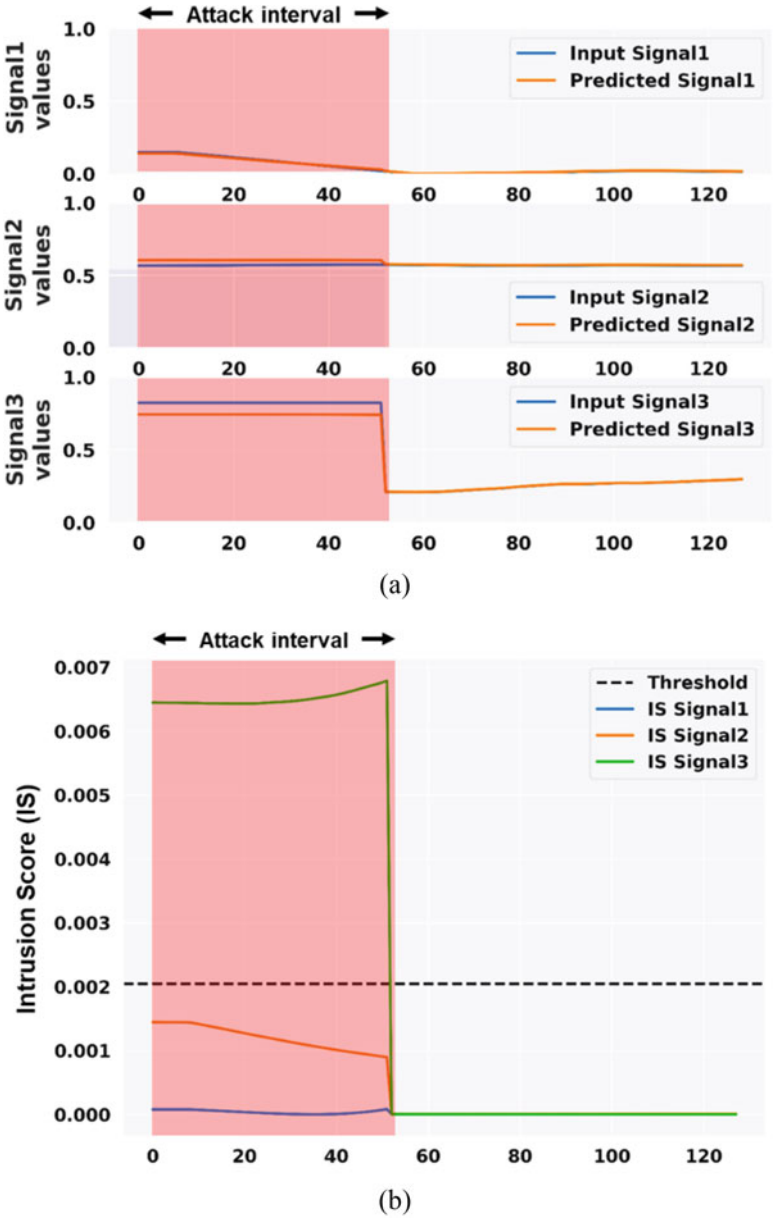


Fig. 11 Internal working of *INDRA* IDS checking a message with three signals under a plateau attack, where (**a**) shows the signal comparisons and (**b**) shows IS of three signals and the IT

removes the linear layer before the output, essentially leaving the task of decoding the message context vector (i.e., reconstructing the input) to GRU based decoder. The abbreviation LED stands for (L)linear layer, (E) encoder GRU, and (D) decoder GRU. The *INDRA-LD* variant replaces the GRU and the linear layer at the decoder with a series of linear layers (LD stands for linear decoder). These experiments were carried out to assess the importance of different layers in the network. However, the encoder part of the network is not changed because it is required to generate an encoding (MCV) of the input time-series data. *INDRA* investigates other variants as well, but they were not included in the discussion as their performance was subpar compared to that of *INDRA-LED* and *INDRA-LD* variants.

Subsequently, the best *INDRA* variant is compared with three state-of-the-art prior works that use different machine learning-based techniques to detect intrusions: *(i)* Predictor LSTM (PLSTM [38]), *(ii)* Replicator Neural Network (RepNet [39]), and *(iii)* 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 actual and the predicted next signal values are computed using a log loss or binary cross-entropy loss function. Additionally, PLSTM uses the prediction loss values at runtime to decide whether a particular message is malicious or not. The second comparison work (RepNet) employs a series of RNN layers to increase the dimensionality of the input data and reconstruct the input signal values by decreasing back to the original dimensionality. RepNet accomplishes this by reducing the mean squared error (MSE) between the input and the reconstructed signal values. At runtime, RepNet uses large deviations between the input received signal and the reconstructed signal values to detect cyberattacks. Lastly, 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 detail in subsequent subsections.

In this work, we use the SynCAN dataset developed by ETAS and Robert Bosch GmbH [36] to evaluate the effectiveness of the *INDRA* framework with its variants and against the above-mentioned prior works. The SynCAN dataset consists of 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 consists of a timestamp, message ID, and individual signal values. Additionally, the test data contains a label column with either 0 or 1 values indicating normal or malicious messages. However, the label information is only available on per message basis and does not specify which signal within the message is under attack. It is important to note that the label information in the training data is not used to train the *INDRA* model, as the *INDRA* model learns the patterns in the input data in an unsupervised manner. This label information in the dataset is only used to evaluate the performance of the proposed IDS using several metrics such as

detection accuracy and false positive rate. 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.

All the machine learning-based frameworks, including the *INDRA* framework and its variants as well as comparison works, are implemented using Pytorch 1.4 with CUDA support. We conducted several 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 a subsequence size of 20 messages and a batch size of 128. Moreover, an early stopping mechanism is employed 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 layer. Furthermore, the ADAM optimizer is used with the mean squared error (MSE) as the loss criterion to compute the reconstruction loss. The trained model is used during testing and subjected to multiple simulated attack scenarios using the test dataset. The intrusion score metric (as stated in Sect. 5.2) was used to calculate the intrusion threshold to flag the message as malicious or normal. Lastly, to evaluate the performance of the IDS, 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.

Additionally, we present the following definitions in the context of IDS before discussing the experimental results:

- True Positive (TP)- when the IDS detects an actual malicious message as malicious;
- False Negative (FN)- when the IDS detects an actual malicious message as normal;
- False Positive (FP)- when the IDS detects a normal message as malicious (aka false alarm);
- True Negative (TN)- when the IDS detects an actual normal message as normal.

INDRA framework primarily focuses on two key performance metrics: *(i) Detection accuracy*- a measure of IDS's ability to detect malicious messages correctly, and *(ii) False positive rate*: also known as false alarm rate. These metrics are computed using (4) and (5), respectively.

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

$$False\ Positive\ Rate = \frac{FP}{FP + TN} \tag{5}$$

6.2 Intrusion Threshold Selection

In this subsection, we present a detailed analysis on the selection of intrusion threshold (IT) by investigating various options such as maximum (max), median, mean, and different quantile bins of validation loss of the final model. As the model is trained only on attack-free data, the reconstruction error for the malicious message will be much larger than the error for normal messages. Hence, *INDRA* explores several candidate options for IT 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 positives, which is highly undesirable. Thus, it is crucial to select an appropriate intrusion threshold value to achieve optimal model performance.

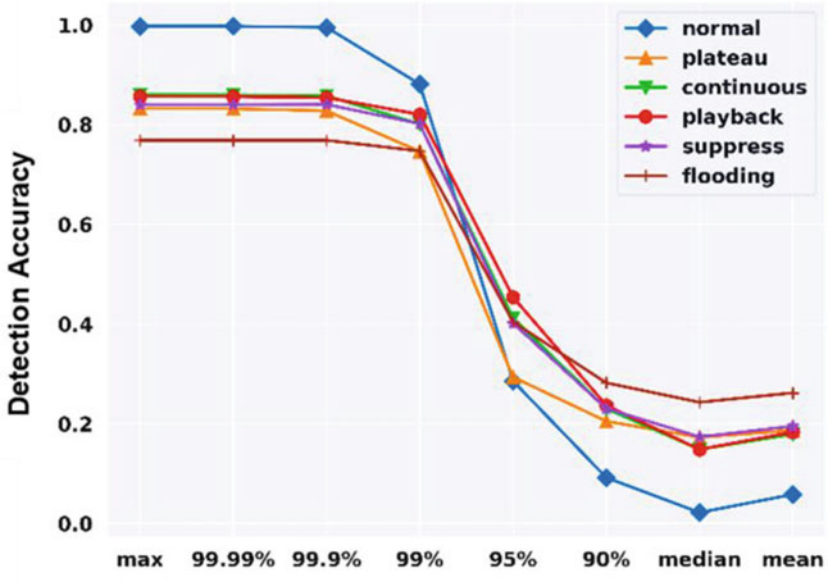
The detection accuracy and false positive rate for various candidate options used to calculate IT is shown in Fig. 12a, b, respectively, under different attack scenarios. The results from the Fig. 12 indicates that selecting a higher validation loss as the IT can lead to high accuracy and a low false alarm rate. However, selecting a very high value (such as '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 are very close. Moreover, to capture attacks that produce small deviations, a slightly smaller threshold value is selected that would still perform similar to the max and 99.99 percentile thresholds under all attack scenarios. Thus, the 99.9th percentile value of the validation loss is chosen as the intrusion threshold (IT) value, and the same IT value is used for the remainder of the experiments (discussed in the following subsections).

6.3 Comparison of INDRA Variants

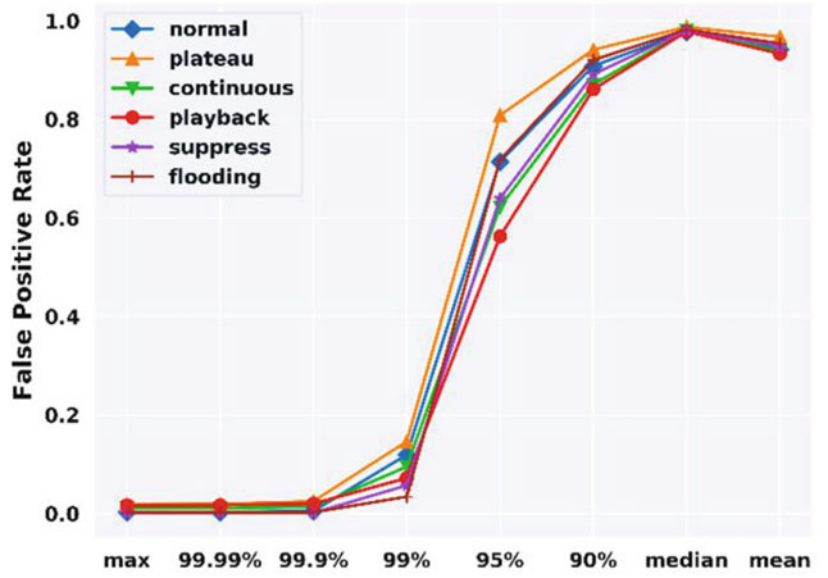
After selecting the appropriate intrusion threshold from the previous subsection, we use that same criterion for evaluating against two other variants: *INDRA*-LED and *INDRA*-LD. The main intuition behind evaluating different variants of *INDRA* is to study the impact of different layers in the model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

Figure 13a illustrates the detection accuracy for the *INDRA* framework and its variants on the y-axis with multiple types of attacks and 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 attack scenarios.

The false positive rate or false alarm rate of *INDRA* and other variants under different attack scenarios is illustrated in Fig. 13b. When compared to other variants, *INDRA* has the lowest false positive rate and highest detection accuracy. Moreover, *INDRA*-LED, which is just short of a linear layer on the decoder side, is the second-best performing model after *INDRA*. The ability of *INDRA*-LED to use



(a)



(b)

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

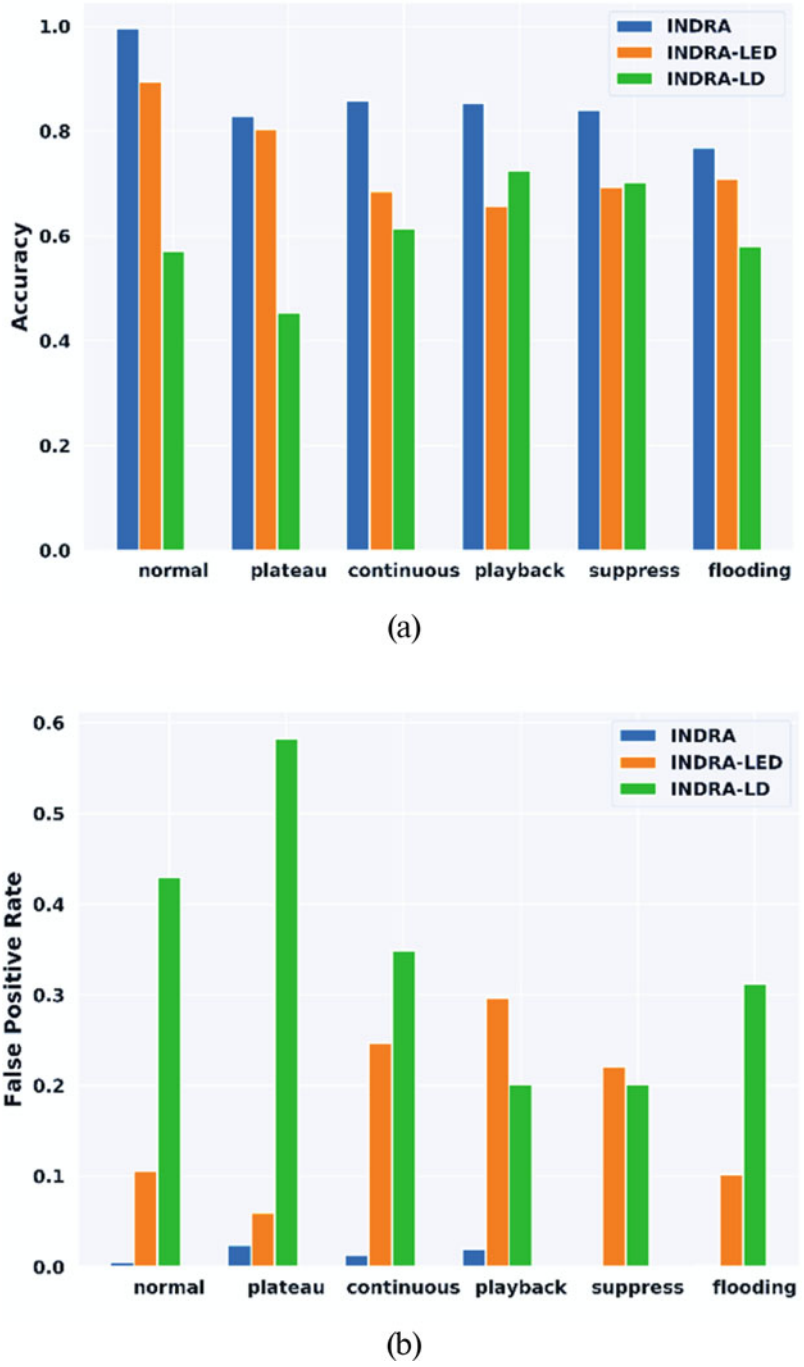


Fig. 13 Comparison of (**a**) detection accuracy and (**b**) false positive rate under different attack scenarios for *INDRA* and its variants (*INDRA*-LED and *INDRA*-LD)

a GRU-based decoder helps in efficiently reconstructing the MCV back to the original input signals. Moreover, it can be clearly seen in both Fig. 13a, b that the absence of GRU layers on the decoder end of *INDRA*-LD resulted in significant performance degradation. Thus, *INDRA* is chosen as the candidate model for subsequent experiments.

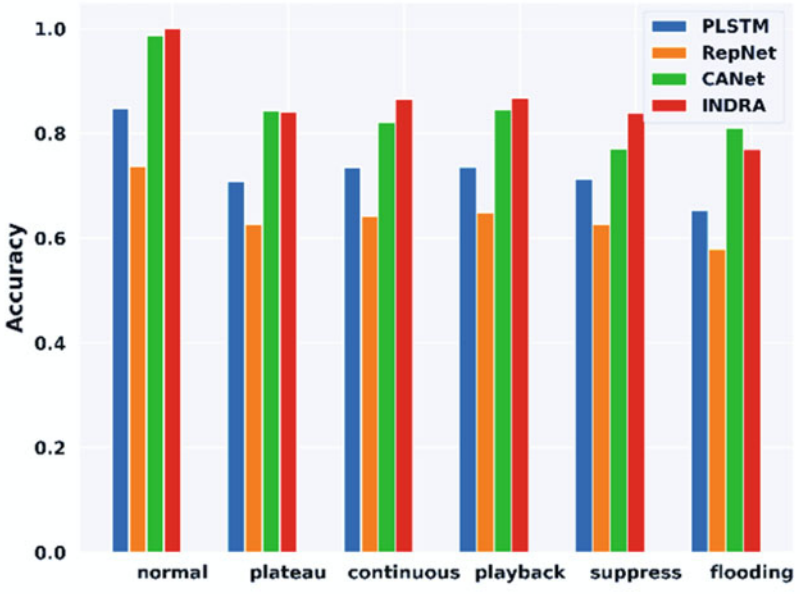
6.4 Comparison with Prior Works

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

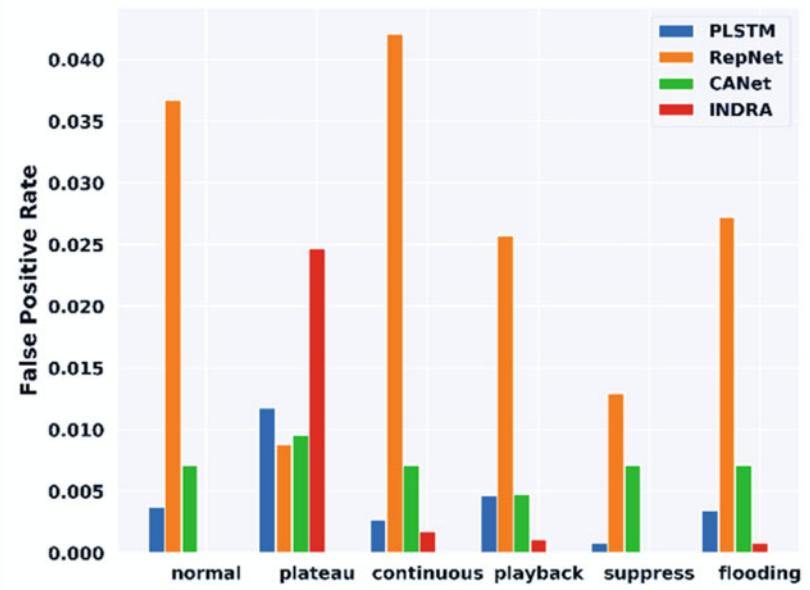
From Fig. 14a, b, it is evident that *INDRA* achieves high detection accuracy under each attack scenario while achieving lower false positive rates. The ability to monitor signal level variations combined with a more cautious selection of intrusion threshold gives *INDRA* an advantage over comparison works. PLSTM and RepNet use the maximum validation loss in the final model as the threshold, whereas CANet uses interval-based monitoring to detect malicious 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, the interval-based monitoring approach employed in CANet struggles to find an optimal threshold resulting in subpar performance. It is essential to highlight that *INDRA* achieves this superior performance by monitoring at a signal level as opposed to prior works that monitor at the message level. Lastly, the false positive rates of *INDRA* remain significantly low, with a maximum of 2.5% for plateau attacks.

6.5 IDS Overhead Analysis

In this section, we present a detailed analysis of the overhead incurred by our proposed *INDRA* IDS. We quantify the IDS overhead in terms of memory footprint and time taken to process an incoming message, i.e., inference time. The former metric is important as the automotive ECUs are highly resource-constrained and have limited memory and compute capacities. Therefore, it is critical to have a low memory overhead to avoid interference with real-time automotive applications. The inference time metric provides important information about the time it takes to detect the attacks and can also be used to compute the utilization overhead on the ECU. Hence, the above-mentioned two metrics are analyzed to study the overhead and quantify the lightweight nature of *INDRA* IDS.



(a)



(b)

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]

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

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)	
IDS 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

To quantify the overhead of our proposed *INDRA* framework and the prior works, we implemented the IDSs on the NVIDIA Jetson Tx2 board, consisting of an ARM Cortex- A57 CPU, which has similar specifications to the state-of-the-art multi-core ECUs. The memory footprint of the *INDRA* framework and the prior works mentioned in the previous subsections is shown in Table 2. It is evident that the *INDRA* framework has a low memory footprint compared to the prior works, except for the RepNet [39]. However, it is important to observe that even though the *INDRA* framework has a slightly higher memory footprint compared to the RepNet [39], *INDRA* outperforms all the prior works, including RepNet [39], in all performance metrics under various 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 other hand, a much lighter model (such as RepNet) fails to capture necessary details about the system behavior due to its limited model parameters, which in turn suffers from performance issues.

We benchmarked different IDS frameworks on an ARM Cortex- A57 CPU to study the inference overhead. In this study, we considered different system configurations to explore a wide variety of ECU hardware that is available in state-of-the-art vehicles. Based on the available hardware resources, single-core (uses only one CPU core) and dual-core (uses two CPU cores) system configurations were selected on the Jetson TX2. The IDS frameworks are executed 10 times for each CPU configuration, and the average inference times (in μs) are recorded in Table 3. From the results in Table 3, it is clear that the *INDRA* framework has significantly faster inference times compared to the prior works (excluding RepNet) under all system configurations. From the results in Fig. 14, it can be seen that RepNet has the worst performance of any comparison framework, despite having a lower inference time. The large inference times for the better-performing frameworks can have a significant impact on the real-time performance of the vehicle and can be catastrophic in the event of deadline misses. We also believe that using a

dedicated deep learning accelerator (DLA) further enhances the performance of the IDS models.

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

6.6 Scalability Results

In this subsection, we present a detailed analysis on the scalability of the *INDRA* framework by studying the system performance using the ECU utilization metric as a function of increasing system complexity (i.e., number of ECUs and messages). Each ECU in the system has a real-time utilization (U_{RT}) and an IDS utilization (U_{IDS}) from running real-time and IDS applications, respectively. We focus on analyzing the IDS overhead (U_{IDS}), as it is a direct measure of the compute efficiency of the IDS. Moreover, as the safety-critical messages monitored by the IDS are periodic, the IDS can be modeled as a periodic application with a period that is the same as the message period [5]. As a result, monitoring an *ith* message (m_i) results in an induced IDS utilization ($U_{IDS,\,mi}$) at an ECU, which can be calculated using (6).

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

where, T_{IDS} and P_{mi} represent the time taken by the IDS to process one message (inference time) and the period of the monitored message, respectively. Moreover, the sum of all IDS utilizations because of monitoring different messages is the overall IDS utilization at that ECU (U_{IDS}) and is computed using (7).

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

To evaluate the scalability of the *INDRA* IDS, six different system sizes were considered. Moreover, a set 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 IDS utilization is calculated using (6) and (7). *INDRA* assumes a pessimistic scenario where all the ECUs in the system have only a single core, which would allow us to analyze the worst-case overhead of the IDS.

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, is illustrated in Fig. 15. In this study, a very pessimistic estimate of 50% real-time ECU utilization

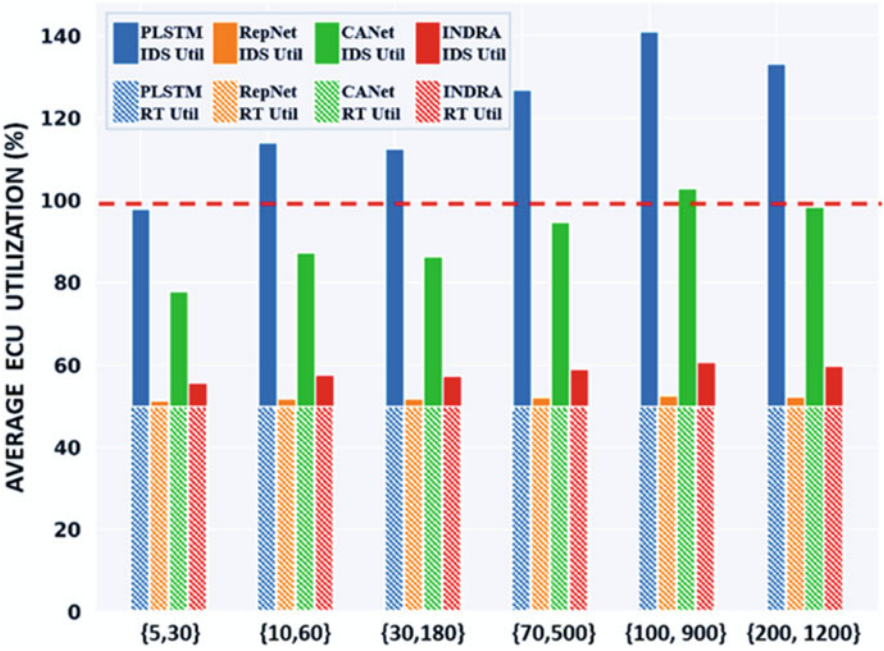


Fig. 15 Scalability analysis of our proposed *INDRA* IDS for different system sizes and the prior works PLSTM [38], RepNet [39], and CANet [36]

for real-time automotive applications ("RT Util", as shown in the dotted bars) is assumed. The solid bars on top of the dotted bars represent the IDS overhead on the ECUs, and the horizontal dotted line in red represents the 100% ECU utilization mark. It is critical to ensure that the ECU utilization does not exceed 100% under any scenario, as it could introduce undesired latencies resulting in missing deadlines for time-critical automotive applications, which 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 a very minimal overhead that is favorable to increasing system sizes. Thus, from the results in this section (Figs. 14 and 15; Tables 2 and 3), it is apparent that *INDRA* not only achieves better performance in terms of detection accuracy and false positive rate for intrusion detection than state-of-the-art prior works, but it is also lightweight and highly scalable.

7 Conclusion

In this chapter, we presented a novel recurrent autoencoder-based lightweight real-time intrusion detection system called *INDRA* for automotive CPS. *INDRA* framework uses the intrusion score (IS) metric to measure the deviation from the

learned system behavior to detect intrusions. Moreover, we presented a thorough analysis on the intrusion threshold selection process and compared the *INDRA* IDS with the best-known prior works in this area. The promising results indicate a compelling potential for utilizing our proposed *INDRA* IDS in emerging automotive platforms.

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 (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) (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) (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. Design Autom. Electron. Syst. (TODAES). **24**(6) (2019)
5. Kukkala, V., Pasricha, S., Bradley, T.: SEDAN: Security-aware design of time-critical automotive networks. IEEE Trans. Vehic. Technol. (TVT). **69**(8) (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. (TCAD). **39**(11) (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. (TECS). **20**(5s), Article 67 (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) (2022)
9. Kukkala, V.K., Thiruloga, S.V., Pasricha, S.: Roadmap for cybersecurity in autonomous vehicles. In: IEEE Consum. Electron. Magaz. (CEM) (2022)
10. Tunnell, J., Asher, Z., Pasricha, S., Bradley, T.H.: Towards improving vehicle fuel economy with ADAS. SAE Int. J. Connect. Autom. Veh. **1**(2) (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) (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) (2018)
13. Dey, J., Taylor, W., Pasricha, S.: VESPA: A framework for optimizing heterogeneous sensor placement and orientation for autonomous vehicles. IEEE Consum. Electron. Magaz. (CEM). **10**(2) (2021)
14. Kukkala, V.K., Pasricha, S., Bradley, T.: Advanced driver-assistance systems: A path toward autonomous vehicles. IEEE Consum. Electron. Magaz. **7**(5) (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) (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) (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. (IJES). **10**(8) (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) (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) (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) (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. **1** (2017)
23. Myers, E.W.: An O(ND) difference algorithm and its variations. Algorithmica (1986)
24. Hoppe, T., Kiltz, S., Dittmann, J.: Applying intrusion detection to automotive IT-early insights and remaining challenges. J. Inf. Assur. Secur. (JIAS). **4**(6) (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. (TCAD). **36**(11) (2017)
26. Cho, K.T., Shin, K.G.: Fingerprinting electronic control units for vehicle intrusion detection. In: Proceedings of USENIX (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. (TIFS). **14**(9) (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) (2015)
29. Müter, M., Asaj, N.: Entropy-based anomaly detection for in-vehicle networks. In: Proceedings of IEEE Intelligent Vehicles Symposium (IV) (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) (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) (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) (2017)
33. Vuong, T.P., Loukas, G., Gan, D.: Performance evaluation of cyber-physical intrusion detection on a robotic vehicle. In: Proc. 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) (2015)
34. Levi, M., Allouche, Y., Kontorovich, A.: Advanced analytics for connected car cybersecurity. In: Proceedings of IEEE Vehicular Technology Conference (VTC) (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) (2016)
36. Hanselmann, M., Strauss, T., Dormann, K., Ulmer, H.: CANet: An unsupervised intrusion detection system for high dimensional CAN bus data. IEEE Access. (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. (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) (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. 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) (2019)