AI for Cybersecurity in Distributed Automotive IoT Systems

Vipin Kumar Kukkala, Sooryaa Vignesh Thiruloga, and Sudeep Pasricha

1 Introduction

Modern vehicles consist of several distributed processing elements called electronic control units (ECUs) that communicate using an in-vehicle network. Each ECU runs various mixed-criticality real-time applications that range from advanced vehicle control to entertainment. Each ECU takes input from different sensors or information from other ECUs to control or actuate different components in the vehicle. Additionally, some of the ECUs in the cars connect to various external systems such as OEM servers to receive over-the-air (OTA) updates via the Internet, other vehicles to communicate traffic information, etc. These unique characteristics of automotive systems make them one of the best examples of a complex distributed time-critical cyber-physical IoT system.

The number of ECUs along with the complexity of software running on these ECUs has been steadily increasing in emerging vehicles. This is mainly driven by the need to support state-of-the-art advanced driver assistance system (ADAS) features such as collision warning, lane keep assist, parking assist, blind spot warning, etc. These advancements have resulted in an increase in the complexity of the in-vehicle network, which is the backbone over which huge volumes of heterogeneous sensor data and safety-critical real-time decisions and control commands are communicated. Moreover, the state-of-the-art ADAS solutions are increasingly communicating with various external systems using advanced communication standards such as 5G technology and Vehicle-to-X (V2X) [1]. This increased interaction with external systems makes modern vehicles highly vulnerable to various cybersecurity attacks that can have catastrophic consequences. Several

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

A. Iranmanesh (ed.), *Frontiers of Quality Electronic Design (QED)*,
https://doi.org/10.1007/978-3-031-16344-9_8

cyberattacks on multiple vehicles have been demonstrated in [2–4] showing various approaches to gain access to the in-vehicle network and take control of the vehicle via malicious messages. As connected and autonomous vehicles are becoming increasingly ubiquitous, the problem of security in automotive systems will be further aggravated. Thus, it is highly essential to prevent unauthorized access of vehicular networks from external attackers to ensure the security of automotive systems.

Traditional computer networks use firewalls as a defense mechanism to protect the network from various unauthorized accesses. However, no firewall is flawless and no network can be impenetrable. Therefore, there is a need for an active monitoring system that scans the network to detect cyberattacks manifesting in the system. An intrusion detection system (IDS) actively monitors network traffic and triggers alerts when malicious behavior or known attack signatures are detected. The IDS acts as the last line of defense in distributed automotive IoT systems.

General IDSs can be classified into two categories: (i) signature-based and (ii) anomaly-based. The signature-based IDSs observe for traces of any known attack signatures, while the anomaly-based IDSs observe for any deviation from the known normal system behavior to indicate the presence of an attacker. Signature-based IDS typically have fewer false alarms (false positives) and faster detection times but can only detect pre-modeled attack patterns that were observed previously. On the other hand, anomaly-based IDS can detect both previously observed and novel attack patterns, while they can suffer from high false alarms and relatively longer detection times when designed sub-optimally. An efficient IDS needs to be robust, lightweight, and scalable with diverse system sizes. In addition, a practical IDS needs to be able to detect a large spectrum of attacks with high confidence in detection. A low false-positive rate is also important because in time-critical systems such as automotive systems, recovery from a false positive can be very expensive.

With the increasing adoption of deep learning and artificial intelligence (AI) in emerging vehicles in an attempt to move toward achieving complete autonomy, their power can be leveraged to develop an effective anomaly-based IDS to detect cyberattacks. The large availability of data and the increasing computational power of ECUs further bolsters the case for an AI-based IDS to detect cyberattacks that are active over the in-vehicle networks. The ability of AI to learn the highly complex features in the data that are hard to capture with traditional techniques gives AI-based IDS a unique edge over other techniques. Moreover, the ability of AI to operate on heterogeneous data can provide an AI-based IDS the ability to detect both known and unknown cyberattacks. Thus, AI-based IDS can be a promising solution for the problem of automotive cybersecurity.

In this chapter, we provide an overview of a novel AI-based vehicle IDS cybersecurity framework called *INDRA* [33] that actively monitors messages in the controller area network (CAN) (a popular in-vehicle network protocol) bus to detect cyberattacks. During the *offline* phase, *INDRA* uses advanced deep learning models to learn the normal system behavior in an unsupervised fashion. At *runtime*, the *INDRA* framework leverages the knowledge of previously learned normal system behavior, to monitor and detect various cyberattacks. *INDRA* aims to maximize

detection accuracy and minimize false-positive rate while incurring a very low overhead on the ECUs. The key contributions of the *INDRA* framework are as follows:

- A gated recurrent unit (GRU)-based recurrent autoencoder network to learn the latent representation of normal system behavior during the offline phase.
- A metric called intrusion score (IS), to quantify the deviation from normal system behavior.
- A thorough analysis toward the selection of thresholds for this intrusion score metric.
- A comprehensive analysis that demonstrates the effectiveness of *INDRA* for vehicle cybersecurity, with superior results compared to the best known state-of-the-art prior works in the area.

2 Related Work

Various techniques have been proposed to design IDS for securing time-critical distributed automotive IoT systems. These works attempt to detect various types of cyberattacks by monitoring the network traffic.

Signature-based IDS reckon on detecting known and pre-modeled attack signatures. In [5], the authors used a language theory-based model to derive attack signatures. However, this technique fails to detect intrusions when it misses the packets transmitted during the early stages of an attack. The authors in [6] used transition matrices to detect intrusions in a CAN bus-based system. This technique achieves a low false-positive rate for trivial attacks but failed to detect more realistic attacks such as replay attacks. In [7], the authors identify prominent attack patterns such as a sudden increase in the message frequency and missing messages to detect intrusions. The authors in [8] proposed a specification-based approach that analyzes the behavior of the system and compare it with the predefined attack patterns to detect intrusions. However, their system can only detect predefined attack patterns and fails to detect unknown attacks. The authors in [9] proposed an IDS technique using the Myers algorithm [10] under the map-reduce framework. In [11], the authors use a time-frequency analysis of CAN messages to detect multiple intrusions. A rule-based regular operating mode region is derived in [12] by analyzing the message frequency at design time. This region is observed for deviations at runtime to detect intrusions. The authors in [13] proposed a technique that uses the fingerprints of the sender ECU's clock skew and the messages to detect intrusions by observing for variations in the clock-skew at runtime. A formal analysis for clock-skew-based IDS is presented in [14] and evaluated on a real vehicle. In [15], a memory heat map is used to characterize the memory behavior of the operating system to detect intrusions. An entropy-based IDS is proposed in [16] that observes for change in system entropy to detect intrusions. However, this technique fails to detect small scale attacks where the change in entropy is minimal.

In summary, signature-based techniques offer a quick solution to the intrusion detection problem with low false-positive rates but cannot detect more complex and novel cyberattacks. Moreover, modeling signatures of every possible attack is impractical.

On the other hand, an anomaly-based IDS aims to learn the normal system behavior in an offline phase and observe for any deviation from the learned normal behavior to detect intrusions (as anomalies) at runtime. In [17], a sensor-based IDS was proposed, where the attack detection sensors are used to monitor various system events to observe for any deviations from the normal behavior. This approach is not only expensive but also suffers from poor detection rates. A one-class support vector machine (OCSVM)-based IDS was proposed in [18]. However, this technique suffers from poor detection latency and has high tuning overhead. The authors in [19] used four different nearest neighbor classifiers to distinguish between a normal and an attack-induced payloads in CAN bus. A decision tree-based detection model is proposed in [20] that monitors the physical features of the vehicle to detect intrusions. However, this model is not realistic and suffers from high detection latencies. A hidden Markov model (HMM)-based technique was proposed in [21] that monitors the temporal relationships between messages to detect intrusions. In [22], a deep neural network-based approach was proposed to monitor the message payloads in the in-vehicle network. This approach is tuned for a low priority tire pressure monitoring system (TPMS), which makes it hard to adapt to high priority safety-critical powertrain applications. The authors in [23] proposed a long short-term memory (LSTM)-based IDS for multi-message ID detection. Due to the high complexity of the model architecture, this technique incurs high overhead on the ECUs. The authors in [24] use an LSTM-based IDS to detect insertion and dropping attacks (explained later in Sect. 4.3). An LSTM-based predictor model is proposed in [25] that predicts the next time step message value at a bit level granularity and examines for large variations in loss to detect intrusions. A recurrent neural network (RNN)-based IDS was proposed in [26] that learns the normal patterns in CAN messages in the in-vehicle network. A hybrid IDS to detect anomalies in time-series data was proposed in [27], which utilizes a specification-based system in the first stage and an RNN-based model in the second stage to detect anomalies. *However, none of these techniques provides a holistic system-level cybersecurity solution that is lightweight, scalable, and reliable to detect multiple types of cyberattacks for in-vehicle networks*.

This chapter describes a novel lightweight recurrent autoencoder-based IDS framework called *INDRA* [33] that utilizes gated recurrent units (GRUs) to monitor messages at a signal level granularity to detect various types of attacks more effectively and successfully than the state of the art. Table 1 summarizes some of the state-of-the-art IDS works' performance under different metrics and shows how *INDRA* fills the existing research gap. The *INDRA* framework aims at improving multiple performance metrics compared to the state-of-the art IDS works that target a subset of performance metrics. A detailed analysis of each metric and evaluation results are presented later in Sect. 6.

Table 1 Comparison between *INDRA*[33] framework and state-of-the-art IDS works

	IDS performance			
Technique	Lightweight	Low false -positive rate	High accuracy	Fast inference
PLSTM [25]	X	✓	X	X
RepNet [26]	✓	X	X	✓
CANet [23]	X	✓	✓	X
INDRA	✓	✓	✓	✓

3 Background on Sequence Learning

The availability of increased computing power from GPUs and custom accelerators led to training neural networks with many hidden layers (known as deep neural networks) that resulted in the creation of powerful models for solving difficult problems in many domains. One such problem is detecting intrusions in the distributed automotive IoT systems, specifically in the in-vehicle network that connects them. In an in-vehicle network, the communication between ECUs happens in a timely manner. Hence, there exist temporal relationships between the messages, which is crucial to exploit, in order to detect intrusions. However, this cannot be achieved using traditional feedforward neural networks as the output of any input is independent of the other inputs. One of the solutions is to use sequence models as they are more appropriate and are designed to handle sequences and time-series data.

3.1 Sequence Models

A sequence model can be thought of as a function which ensures that the current output is dependent not only on the current input but also on the previous inputs. Recurrent neural network (RNN) is one of the first sequence models which was introduced in [28]. In recent years, improved sequence models such as long short-term memory (LSTM) and gated recurrent unit (GRU) have also been developed.

3.1.1 Recurrent Neural Network (RNN)

An RNN is a type of artificial neural network that takes sequential data (such as sequence or time-series data) as the input and learns the relationship in the data. RNNs achieve this by using the hidden states, which allows learned information to persist over time steps. Moreover, the hidden states also enable the RNN to connect previous information to current inputs. An RNN cell with feedback is shown in Fig. 1a, and an RNN unrolled in time is shown in Fig. 1b.

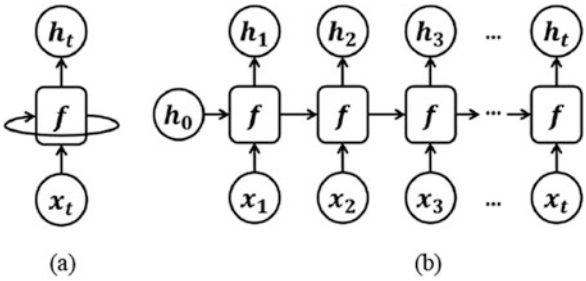

Fig. 1 (**a**) A single RNN cell and (**b**) RNN unit unrolled in time, where f is the RNN cell, x is the input, and h represents hidden states [33]

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

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

where W, U represent the weight matrices, b is a bias term, and f is a nonlinear activation function (such as a sigmoid or tanh). One of the limitations of RNNs is that they are very hard to train. As RNNs and other sequence models deal with sequence or time-series inputs, backpropagation happens through various time samples (known as backpropagation through time (BPTT)). During the BPTT process, the feedback loop in RNNs causes the errors to shrink or grow rapidly (resulting in vanishing or exploding gradients respectively), destroying the information in backpropagation. This problem of vanishing gradients hampers the RNNs from learning *long-term dependencies*. This problem was solved with the introduction of additional states and gates in the RNN cell to remember long-term dependencies, which led to the introduction of long short-term memory networks [29].

3.1.2 Long Short-Term Memory (LSTM) Networks

LSTMs are improved RNNs that use cell state and the hidden state information along with several gates to remember long-term dependencies in the input sequence. The cell state can be visualized as a transport highway that carries relevant information throughout the processing of a sequence. The cell state accommodates the information from earlier time steps, which can be used in the later time steps, thereby reducing the effects of short-term memory. The information in the cell state is modified using various gates, which helps the network decide which information needs to be retained and which information to forget.

An LSTM cell consists of three gates: (i) forget gate (f_t), (ii) input gate (i_t), and (iii) output gate (o_t), as shown in Fig. 2a. The forget gate is a binary gate that controls which information to retain from the previous cell state (c_{t-1}). The input gate is responsible for adding relevant information to the current cell state (c_t). Lastly, the output gate controls the output layer, which uses information from the forget and input gates to produce an appropriate output. An unrolled LSTM unit is shown in Fig. 2b.

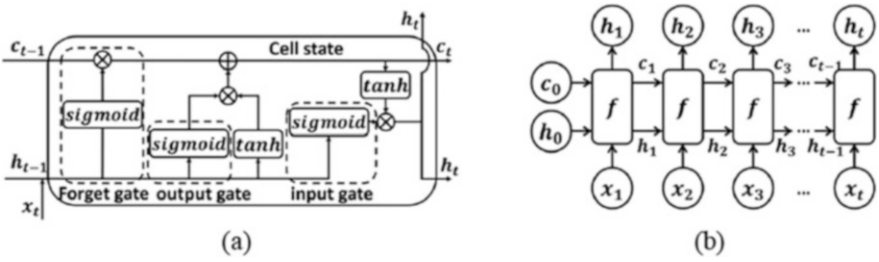

Fig. 2 (**a**) A single LSTM cell with different gates and (**b**) LSTM unit unrolled in time, where *f* is an LSTM cell, *x* is input, *c* is cell state, and *h* is the hidden state [33]

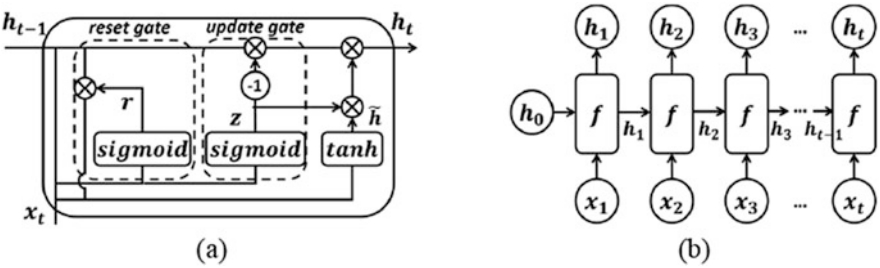

Fig. 3 (**a**) A single GRU cell with different gates and (**b**) GRU unit unrolled in time, where *f* is a GRU cell, *x* is input, and *h* represents hidden states [33]

The combination of the abovementioned different gates, along with the cell and hidden states, enables LSTMs to learn long-term dependencies in sequences. However, they are not computationally efficient as the addition of multiple gates increased the complexity of the sequence path (more than in RNNs) and also require more memory at runtime. Additionally, training LSTMs is compute-intensive even with advanced training methods such as truncated backpropagation. To overcome these limitations, a simpler recurrent neural network called gated recurrent unit (GRU) network was introduced in [30] that can be trained faster than LSTMs and also remembers dependencies in long sequences with relatively low memory overhead while solving the vanishing gradient problem.

3.1.3 Gated Recurrent Unit (GRU)

A GRU cell uses an alternate route for gating information by combining the input and forget gate of the LSTM into a solitary *update* gate. GRUs furthermore combine the hidden and cell states, as shown in Fig. 3a, b.

A GRU cell consists of two gates: (i) reset gate and (ii) update gate. The reset gate combines new input with past memory, while the update gate selects the amount of relevant data that should be held. This enables the GRU cell to control the data stream like an LSTM by uncovering its hidden layer contents. Moreover,

GRUs achieve this using fewer gates and states, which makes them computationally more efficient with low memory overhead compared to the LSTMs. As real-time automotive ECUs are highly resource-constrained distributed embedded systems with stringent energy and power budgets, it is crucial to employ low overhead models for inferencing tasks. This makes the GRU-based networks an ideal fit for inference in automotive systems. Moreover, GRUs are relatively new and less explored and have a lot of potential to offer compared to its predecessors RNNs and LSTMs. Hence, in this chapter, a lightweight GRU-based IDS framework called *INDRA* is presented (explained in detail in Sect. 5).

The sequence models can be trained using both supervised and unsupervised learning approaches. Due to the large volume of automotive network data in a vehicle, labeling the data can become very tedious. Additionally, the variability in the messages between different vehicle models from the same manufacturer and the proprietary nature of this information makes it furthermore challenging to accurately label messages. The accessibility to automotive network data via onboard diagnostics (OBD-II) facilitates the collection of large amounts of unlabeled data. Thus, the IDS in *INDRA* uses GRUs in an unsupervised learning setting.

3.2 Autoencoders

An autoencoder is an unsupervised learning algorithm that tries to reconstruct the input by learning the latent input features. Autoencoders achieve this by encoding the input data (x) toward a hidden layer and finally decoding it to produce a reconstruction $\tilde{x}$ (as shown in Fig. 4). The encoding produced at the hidden layer is called an embedding. The layers that create this embedding are called the encoder, and the layers that utilize the embedding and reconstruct the original input are called the decoder. When training the autoencoders, 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. Both encoder and decoder achieve this with the help of nonlinear activation functions, such as tanh and rectified linear unit (ReLU). Moreover, the autoencoder network tries to recreate the input as accurately as possible by selectively extracting the key features from the inputs with a goal of minimizing reconstruction error. The most commonly used loss functions in autoencoders are mean squared error (MSE) and Kullback-Leibler (KL) divergence.

Since the autoencoders aim to reconstruct the input by learning the underlying distribution of the input data, it makes them an excellent choice to learn and reconstruct highly correlated time-series data efficiently by learning the temporal relations between signals. *Thus, the INDRA framework uses lightweight GRUs in an autoencoder to learn latent representations of CAN messages in an unsupervised learning setting*.

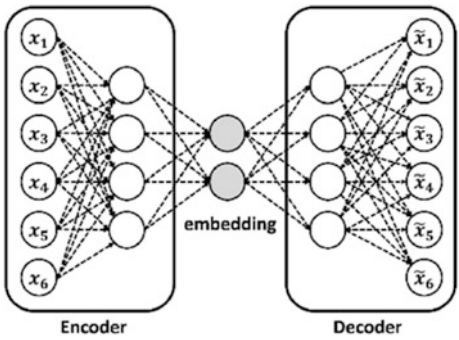

Fig. 4 Autoencoders [33]

4 Definitions and Problem Formulation

4.1 System Model

The *INDRA* framework considers a generic distributed automotive *system* consisting of multiple ECUs connected using a CAN-based in-vehicle network, as shown in Fig. 5. Each ECU runs a set of hard real-time automotive applications that have strict timing and deadline constraints. Additionally, each ECU also executes intrusion detection applications that monitors and detects intrusions in the in-vehicle network. *INDRA* employs a distributed IDS approach, where the intrusion applications are collocated with real-time automotive applications as opposed to a centralized IDS approach where a single central ECU handles all intrusion detection tasks. This design decision is driven by the following reasons:

- A centralized IDS approach is particularly prone to single-point failures, which can fully open up the system to the attacker.
- In some extreme scenarios such as during a distributed denial-of-service (DDoS) or flooding attack (explained in Sect. 4.3), the in-vehicle network can get highly congested and the centralized IDS might not be able to communicate with the victim ECUs.
- If an attacker succeeds in fooling the centralized IDS ECU, attacks can go undetected by the other ECUs, compromising the entire system. However, with a distributed IDS approach, fooling multiple ECUs is required which is much harder. Even if one of the ECUs is compromised, the attack can still be detected by the decentralized intelligence.
- In a distributed IDS approach, ECUs can stop accepting suspicious messages as soon as an intrusion is detected without waiting for a centralized system to notify them, resulting in faster detection times.
- In a distributed IDS approach, the computation load of intrusion detection is split among the ECUs, and the monitoring can be limited to only the required messages. This facilitates multiple ECUs to monitor a subset of messages independently, with very lower overhead.

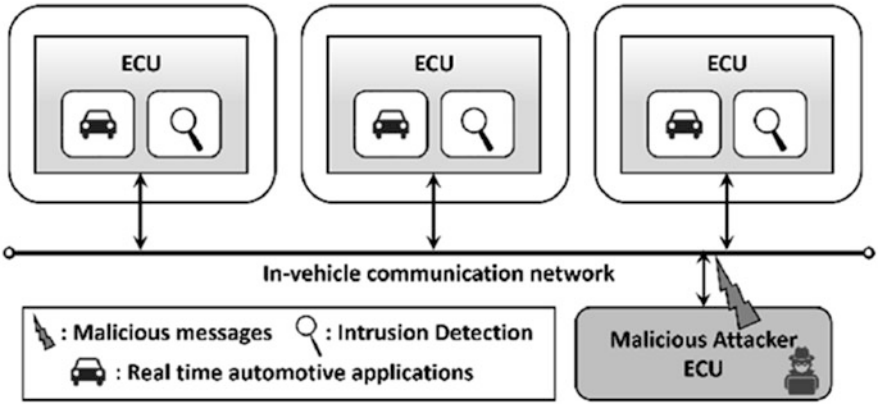

Fig. 5 Overview of the system model considered in *INDRA* [33]

Many prior works, such as in [5, 12], consider a distributed IDS approach for these reasons. Moreover, with automotive ECUs becoming increasingly powerful, the collocation of IDS applications with real-time automotive applications in a distributed manner is feasible, provided the overhead from the IDS is minimal. The *INDRA* framework is not only lightweight but also highly scalable and achieves superior intrusion detection performance, as discussed in Sect. 6.

An efficient IDS design should have low susceptibility to noise, low cost, and a low power/energy footprint. Additionally, *INDRA* considers the following design objectives in the development process of the IDS:

- *Lightweight*: Intrusion detection tasks can incur additional overhead on the ECUs, which could result in poor application performance or missed deadlines for real-time applications. This can have catastrophic consequences in some cases. Thus, it is important to have a lightweight IDS that incurs very minimal overhead on the system.
- *Few false positives*: This is a highly desired quality in any kind of IDS (even outside of the automotive domain), as handling false positives can become expensive very quickly. An efficient IDS needs to have few false positives or false alarms.
- *High attack coverage*: Attack coverage is the range of attacks an IDS can detect. A good IDS needs to be able to detect more than one type of attack. A high attack coverage for IDS will make the system resilient to multiple attack surfaces.
- *Scalability*: This is a crucial requirement as the numbers of ECUs, software, and network complexity have been increasing in the emerging vehicles. A practical IDS should be highly scalable and be able to support various system sizes.

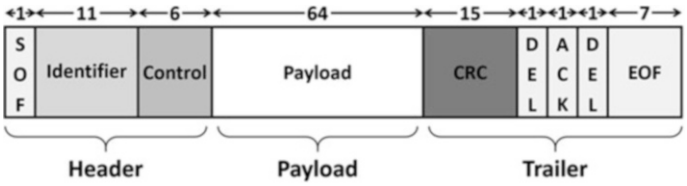

Fig. 6 Standard CAN frame format [33]

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 Real-world CAN message with signal information [33]

4.2 Communication Model

A brief overview of the vehicle communication model that was considered in *INDRA* is presented in this subsection. The *INDRA* framework mainly focuses on detecting intrusions in a CAN bus-based automotive system. Controller area network (CAN) is the de facto industry standard in-vehicle network protocol for modern-day automotive systems. CAN is a low-cost, lightweight event-triggered communication protocol that transmits messages in the form of CAN frames. The structure of a standard CAN frame is shown in Fig. 6, and the length of each field (in bits) is shown on the top. The standard CAN frame consists of header, payload, and trailer segments. The header segment consists of information such as the message identifier (ID) and the length of the message. The actual data that needs to be transmitted is in the payload segment. Lastly, the information in the trailer segment is mainly used for error checking at the receiver. A variation of the standard CAN, called CAN-extended or CAN 2.0B, is also becoming increasingly common in modern vehicles. CAN extended consists of a 29-bit identifier compared to 11-bit identifier in the CAN standard, allowing for more number of unique message IDs.

The IDS design in *INDRA* focuses on monitoring the payload segment of the message and observe for anomalies to detect intrusions. This is mainly because an attacker needs to modify the message payload to accomplish a malicious activity. An attacker could also target the header or trailer segments, but it would result in the message getting rejected at the receiver. The payload segment consists of multiple data entities called signals. An example real-world CAN message with multiple signals is shown in Fig. 7 [31]. Each signal has a fixed length (in bits), an associated data type, and a start bit that specifies its starting location in the 64-bit payload segment of the CAN message.

The *INDRA* framework focuses on monitoring individual signals within message payloads to observe for anomalies and detect intrusions. The neural network model in the *INDRA* framework is trained to learn the temporal dependencies between the messages at a signal level during training and observes for deviations during the deployment (at runtime) to detect intrusions in the in-vehicle network. This signal level monitoring would not only give the capability to detect the presence of an intruder but also helps in identifying the signal within the message that is being targeted during an attack. This information can be crucial in understanding the intentions of the attacker, which can be used for developing countermeasures. The signal level monitoring mechanism in *INDRA* is discussed in detail in Sect. 5.2.
Note: Even though the *INDRA* framework mainly focuses on detecting intrusions by monitoring CAN messages, this approach can be extended to be used with other in-vehicle network protocols as the framework is agnostic to the underlying communication protocol.

4.3 Attack Model

The *INDRA* framework aims to protect the vehicle from various types of cyberattacks that are listed below. These are some of the most commonly seen and hard to detect automotive attack patterns that have been widely considered in literature to evaluate IDS models.

1. *Flooding attack*: This is the most common and easy to launch attack and requires little to no knowledge about the system. In this attack, the attacker floods the in-vehicle network with a random or specific message and prevents the other ECUs from communicating. This is also known as the denial-of-service (DoS) attack. These attacks are generally detected and prevented by the bridges and gateways in the in-vehicle network and often do not reach the last line of defense (the IDS). However, it is important to consider these attacks in the IDS evaluation as they can have a severe impact on the system when handled incorrectly.
2. *Plateau attack*: In this attack, an attacker overwrites a signal value with a constant value over a period of time. The attack severity depends on the magnitude of the change in signal value and the attack duration. Large changes in magnitude of the signal values are easier to detect compared to shorter changes.
3. *Continuous attack*: In this attack, an attacker slowly overwrites the signal value until some target value is achieved and tries to avoid the triggering of IDS. This attack is hard to detect and can be sensitive to the IDS parameters (discussed in Sect. 5.2).
4. *Suppress attack*: In this attack, the attacker suppresses the signal value(s) by either disabling the communication controller or by powering off the target ECU. These attacks can be easily detected, when the message transmissions are shut down for long durations, but are harder to detect for shorter durations.

5. *Playback attack*: In this attack, the attacker replays a valid series of message transmissions from the past trying to trick the IDS. This attack is hard to detect if the IDS does not have the ability to capture the temporal relationships between messages.

Moreover, the *INDRA* framework assumes that the attacker can gain access to the vehicle using the most common attack vectors, which include connecting to V2X systems that communicate with the outside world (e.g., infotainment and connected ADAS systems), connecting to the OBD-II port, probing into the in-vehicle bus, and replacing an existing ECU. Furthermore, the *INDRA* framework assumes that the attacker has access to the bus parameters (such as BAUD rate, parity, flow control, etc.) that can help in gaining access to the in-vehicle network.

Problem objective The goal of *INDRA* is to implement a lightweight IDS that can detect various types of cyberattacks (mentioned earlier) in a CAN bus-based distributed automotive system, with a high detection accuracy and low false-positive rate, and while maintaining a large attack coverage.

5 *INDRA* Framework Overview

The *INDRA* framework aims to achieve a signal level anomaly-based IDS for monitoring CAN messages in automotive embedded systems. An overview of the *INDRA* framework is shown in Fig. 8. At a high level, the *INDRA* framework consists of design-time and runtime phases. At design time, *INDRA* uses trusted CAN message data to train a recurrent autoencoder-based model to learn the normal operating behavior of the system. At runtime, the trained recurrent autoencoder model is used for observing deviations from normal behavior (inference) and detect intrusions based on the deviation computed using the proposed intrusion score metric (detection). The following subsections describe these steps in more detail.

5.1 Recurrent Autoencoder

Recurrent autoencoders are powerful neural networks that are designed to behave like an autoencoder network but handle time-series or sequence data inputs. They can be visualized similar to the regular feed-forward neural network-based autoencoders, except with the neurons being either RNN, LSTM, or GRU cells (discussed in Sect. 3). Similar to regular autoencoders, the recurrent autoencoders consist of an encoder and a decoder stage. The encoder is responsible for generating a latent representation of the input sequence data in an n-dimensional space. The decoder uses the generated latent representation from the encoder and tries to reconstruct the input with minimal reconstruction error. In this section, a lightweight recurrent autoencoder model that is customized for the design of IDS to detect

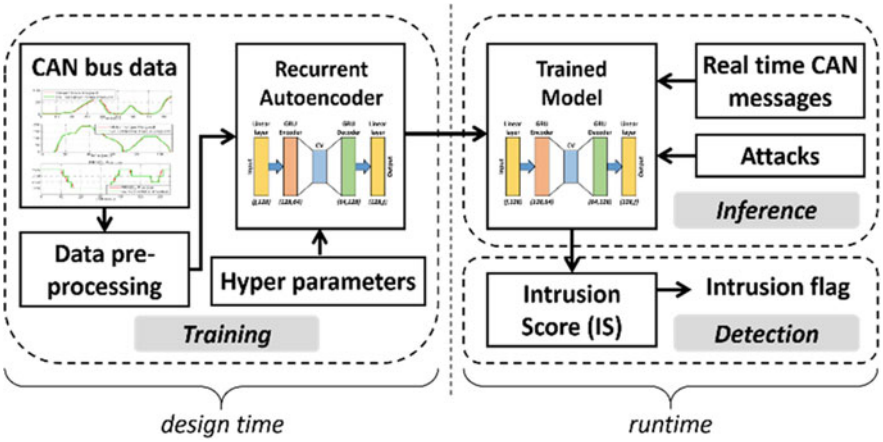

Fig. 8 Overview of *INDRA* framework [33]

intrusions in the in-vehicle network data is presented. The details related to the recurrent autoencoder model architecture and the different stages involved in its training are discussed in the subsequent subsections.

5.1.1 Model Architecture

The proposed recurrent autoencoder model architecture in *INDRA* with the dimensions (input, output) of each layer is illustrated in Fig. 9. The model consists of a linear layer at the input, GRU-based encoder, GRU-based decoder, and a final linear layer before the output. The time-series CAN message data with signal level values with f features (where f is the number of signals in that particular message) is given as the input to the first linear layer. The output of the first linear layer is passed to the GRU-based encoder to generate 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 different signals in the input message data in the form of a vector, hence the name. Each MCV can be thought of as a point in an n-dimensional space that contains the context of the series of signal values given as input. The MCV is given as the input to a GRU-based decoder, which feeds its output as an input to the final linear layer. The linear layer at the end produces the reconstructed input time-series that represents the CAN message data with individual signal level values. Mean square error (MSE) loss function is used to compute the loss between the input and the reconstructed input. The model weights are updated using backpropagation through time (BPTT). The *INDRA* framework builds a recurrent autoencoder model for each message ID.

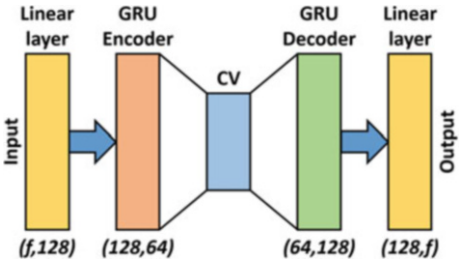

Fig. 9 Recurrent autoencoder network architecture in *INDRA* (f is number of features, i.e., number of signals in the input CAN message, MCV is message context vector) [33]

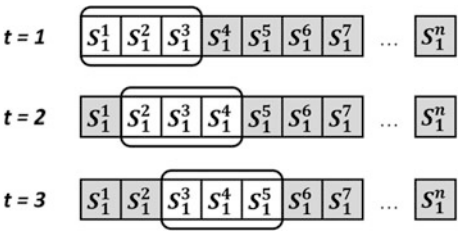

Fig. 10 Illustration of rolling window-based approach [33]

5.1.2 Training Procedure

The first step of the training process is preprocessing the input CAN message data. Each sample in the dataset consists of a message ID and corresponding values of the signals within that message ID. The signal values are scaled between 0 and 1 for each signal type, as the range of signal values can be very large in some cases. Using unscaled signal values as inputs can result in an extremely slow or very unstable training process. Moreover, scaling the signal values also helps in avoiding the problem of exploding gradients.

The final preprocessed data is split into training data (85%) and validation data (15%) and is prepared for training using a rolling window-based approach. This involves selecting a window of fixed size and rolling it to the right by one-time sample every time step. A rolling window size of three samples for three time steps ($t = 1, 2, 3$) is illustrated in Fig. 10, where the term S_i^j represents the i^{th} signal value at j^{th} sample. The elements in the rolling window are collectively called as a subsequence and the subsequence length is equal to the size of the rolling window. As each subsequence consists of a set of signal values over time, the recurrent autoencoder model in *INDRA* tries to learn the existing temporal relationships between the series of signal values. These signal level temporal relationships play a crucial role in identifying more complex cyberattacks such as *continuous* and *playback* (as discussed in Sect. 4.3). The process of training using subsequences is carried out iteratively until the end of the sequence in training data.

Each iteration in the training step consists of a forward pass and a backward pass using BPTT to update the weights and biases of the neurons (discussed in Sect. 3) based on the error value. At the end of the training, the model's learning is evaluated (forward pass only) using the validation data, which was not seen by the model during the training. By the end of validation, the model has seen the

complete dataset once and this is known as an *epoch*. The model is trained for multiple epochs until the model reaches convergence. Moreover, the process of training and validation using subsequences is sped up by training the input data in groups of subsequences known as mini-batches. Each mini-batch consists of multiple consecutive subsequences that are given as input to the model in parallel. The size of each mini-batch is commonly referred to as *batch size*, and it is a common practice to choose the batch size as a power of two. Lastly, to control the rate of update of parameters during backpropagation, a learning rate needs to be specified to the model. The hyperparameters such as subsequence size, batch size, learning rate, etc., that are chosen in the *INDRA* are presented later in Sect. 6.1.

5.2 Inference and Detection

At runtime, the trained model is set to evaluation mode, where only the forward passes occur and the weights and biases are not updated. In this phase, the trained model is tested under multiple attack scenarios (mentioned in Sect. 4.3), by simulating appropriate attack conditions in the CAN message dataset.

Every data sample that is given as the input to the model gets reconstructed at the output, and the reconstruction loss is fed to the detection module to compute a metric called *intrusion score* (IS). The IS helps in identifying whether a signal is normal or malicious. The IS metric is computed at a signal level to predict the signal that is under attack. The IS metric is computed at every iteration during inference, as a *squared error* to estimate the prediction deviation from the input signal value, as shown below:

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

where, S_i^j represents i^{th} signal value of the j^{th} sample, $\hat{S}_i^j$ denotes its reconstruction, and m is the number of signals in the message. The predicted value would have a large deviation from the input signal value (i.e., large IS value), when the input signal pattern is not seen during the training phase, and a minimal IS value otherwise. This is the basis for the detection step in *INDRA*.

Additionally, the *INDRA* framework combines the signal level IS information into a message-level IS, by taking the maximum IS of the signals in that message as shown in Eq. (3). This is mainly to facilitate the lack of signal level intrusion label information in the dataset.

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

To get adequate detection accuracy, an *intrusion threshold* (IT) needs to be selected for flagging messages appropriately. *INDRA* explores multiple choices for IT, using the best model from the training process. The best model is the model with

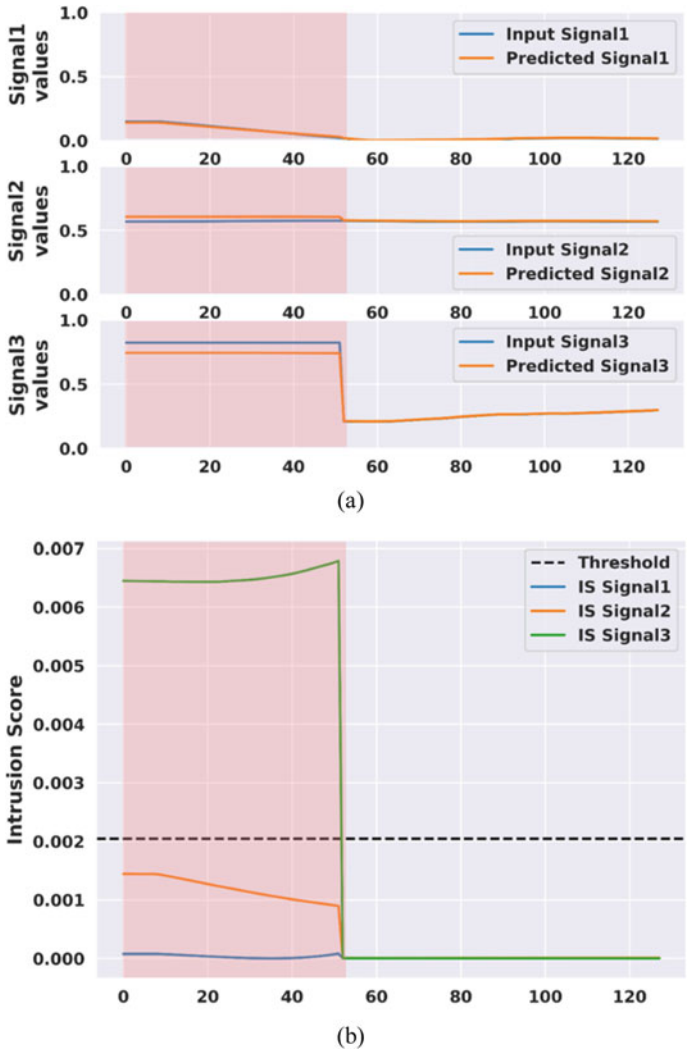

Fig. 11 Snapshot of *INDRA* IDS checking a message with three signals under a plateau attack, where (**a**) shows the signal comparisons and (**b**) shows IS for signals and IS for the message and intrusion flag [33]

the lowest validation running loss during the training process. Using the best model, multiple metrics such as maximum, mean, median, and 99.99%, 99.9%, 99%, and 90% validation loss across all iterations are logged as the choices for the IT. The detailed analysis on selection of IT is presented in Sect. 6.2.

A snapshot of *INDRA* IDS working in an environment with cyberattacks is illustrated in Fig. 11a, b, with a plateau attack on a message with three signals, between time 0 and 50. The highlighted area in red represents the attack interval.

Figure 11a shows the input (true) vs *INDRA* IDS predicted signal value comparisons for three signals. It can be observed that for most of the time, the reconstruction is close for almost all signals except during the attack interval. Signal 3 is subjected to a plateau attack where the attacker held a constant value until the end of attack interval as shown in the third subplot of Fig. 11a (note that this resulted in a larger difference between the IDS predicted and actual input signal values in the third subplot, compared to signals 1 and 2). Figure 11b shows the different signal intrusion scores for the three signals in the message. The dotted black line represents the intrusion threshold (IT). As mentioned earlier, the maximum of signal intrusion scores is chosen as message intrusion score (MIS), which in this case is the IS of signal 3. It can be clearly seen in Fig. 11b that the intrusion score of signal 3 is above the IT, for the entire duration of the attack interval, highlighting the ability of *INDRA* to detect such attacks. The value of IT (equal to 0.002) in Fig. 11b is computed using the method discussed in Sect. 6.2. *Note*: This IT value is specific to the example case shown in Fig. 11 and is not a representation of the IT value used for the remaining experiments. Section 6.2 describes the selection of IT value in the *INDRA* framework.

6 Experiments

6.1 Experimental Setup

To evaluate the performance of the *INDRA* framework, an analysis for the selection of intrusion threshold (IT) is presented. Using the selected IT, two variants of the *INDRA* framework (*INDRA*-LED and *INDRA*-LD) are compared against the baseline *INDRA* framework. The former variant removes the end linear layer before the output and essentially has only the GRU to decode the context vector. The term LED implies the (L) linear layer, (E) encoder GRU, and (D) decoder GRU. The latter variation replaces the GRU and the linear layer at the decoder with a series of linear layers (LD implies linear decoder). These variants were studied mainly to understand the importance of different layers in the network. However, the encoder portion of the network remained unchanged in the variants as a sequence model is needed to generate an encoding of the time-series data. The study in *INDRA* explored other variants, but they are not included in the discussion as their performance was inferior compared to the LED and LD variants.

Subsequently, the best variant of the *INDRA* framework is compared with three prior works: predictor LSTM (PLSTM [25]), replicator neural network (RepNet [26]), and CANet [23]. The first comparison work (PLSTM) employs an LSTM-based network that is trained to predict the signal values in the next message transmission. PLSTM achieves this by taking the 64-bit CAN message payload as the input and learns to predict the next signal values in the message at a bit-level granularity by minimizing the prediction error. A log loss or binary cross-entropy

loss function is used to measure the bit level deviations between the real next signal values and the predicted next signal values. At runtime, PLSTM uses the calculated prediction loss value to decide whether a particular message is malicious or not. The second comparison work (RepNet) uses a series of RNN layers to increase the dimensionality of the input data and reconstruct the signal values by reducing back to the original dimensionality. RepNet achieves this by minimizing the mean squared error (MSE) between the input and the reconstructed signal values. During runtime, large deviations between the input signal and the reconstructed signal values are used to detect intrusions. Lastly, the third comparison work (CANet) unifies multiple LSTMs and linear layers in an autoencoder architecture and adapts a quadratic loss function to minimize the signal reconstruction error. Details related to all experiments conducted with the *INDRA* variants and comparison works are discussed in further subsections.

To evaluate the *INDRA* framework with its variants and against prior works, an open-source dataset called SynCAN, developed by ETAS and Robert Bosch GmbH [23] is used. The dataset consists of CAN message data for ten different message IDs that were modeled based on the real-world CAN message data. The dataset comes with both training and test data with multiple attacks, as discussed in Sect. 4.3. Each row in the dataset consists of a timestamp, message ID, and individual signal values. Additionally, the test data consists of a label column with either 0 or 1 values indicating normal or malicious messages, respectively. The label information is available on a per message basis and does not indicate which signal within the message is subjected to the cyberattack. This label information is used to evaluate the *INDRA* IDS over several metrics such as detection accuracy and false-positive rate (discussed in detail in the next subsections). Moreover, to simulate a more realistic attack scenario in the in-vehicle networks, the test data has normal CAN traffic between the attack injections. *Note*: The training phase does not use any label information, as *INDRA* learns the patterns in the input data in an unsupervised manner.

All the machine learning-based frameworks including *INDRA* and its variants, and comparison works are implemented using PyTorch 1.4. Additionally, several experiments were conducted to select the best performing model hyperparameters (including number of layers, hidden unit sizes, and activation functions). The final recurrent autoencoder model presented in Sect. 5.1 was trained using the SynCAN dataset by splitting 85% of train data for training and the remaining for validation. The validation data is primarily used to evaluate the performance of the trained model at the end of every epoch. The model was 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. An early stopping mechanism was employed during the training phase that monitors the validation loss across epochs and stops the training process if there is no improvement after ten (patience) epochs. A learning rate of 1e-4 is chosen, and tanh activations are applied after each linear and GRU layers. Lastly, an ADAM optimizer with the mean squared error (MSE) loss criterion is used for back propagation. During testing, the trained model is evaluated using multiple test data inputs to simulate various attack scenarios. The

intrusion threshold is computed based on the intrusion score metric (as described in Sect. 5.2), which was used in determining a message as malicious or normal. Various performance metrics such as detection accuracy, false positives, etc. are computed to quantify the performance of *INDRA*. All the simulations are run on an AMD Ryzen 9 3900X server with an Nvidia GeForce RTX 2080Ti GPU.

Before looking at the experimental results, the following terminologies are defined in the context of IDS:

- True positive (TP): when an IDS detects an actual malicious message as malicious.
- False negative (FN): when an IDS detects an actual malicious message as normal.
- False positive (FP): when an IDS detects a normal message as malicious (aka false alarm).
- True negative (TN): when an IDS detects an actual normal message as normal.

The INDRA framework focuses on two key performance metrics: (i) *detection accuracy*, which is the measure of an IDS ability to detect intrusions correctly, and (ii) *false-positive rate*, also known as false alarm rate. These metrics are calculated using Eqs. (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 Intrusion Threshold Selection

A comprehensive analysis for the selection of intrusion threshold (IT) by considering various options such as max, median, mean, and different quantile bins of validation loss of the final model is presented in this subsection. The reconstruction error of the model for the normal messages should be much smaller than the error for malicious messages. Hence, several candidate options for the IT are explored to achieve this goal that would work across multiple attack and no-attack scenarios. In some scenarios, having a large IT value can make it harder for the model to detect the attacks that change the input pattern minimally (e.g., continuous attack). In contrast, having a small threshold value can potentially trigger multiple false alarms, which is highly undesirable in time-critical systems. Thus, it is crucial to select an appropriate IT value to optimize the performance of the model.

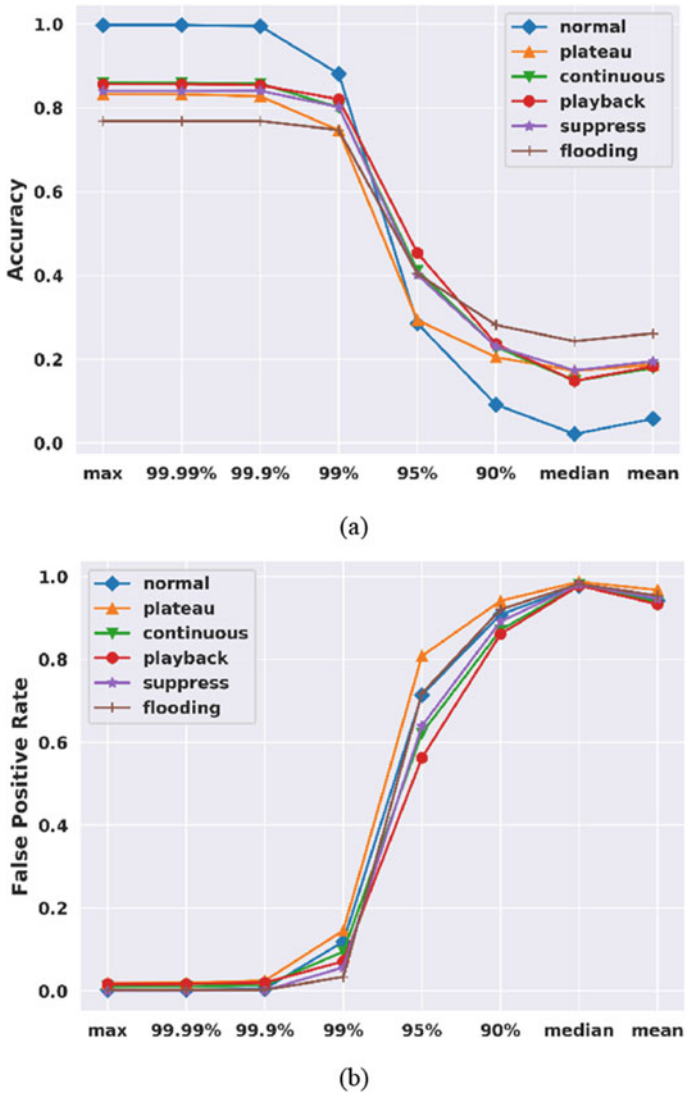

Fig. 12 Comparison of (**a**) detection accuracy and (**b**) false-positive rate for various candidate options of intrusion threshold (IT) as a function of validation loss under different attack scenarios. (% refers to percentile not percentage) [33]

Figure 12a, b illustrates the detection accuracy and false-positive rate, respectively, for various candidate options to calculate IT, under different attack scenarios. From the results in Fig. 12a, b, it can be seen that selecting higher validation loss as the IT can result in a high detection accuracy and low false alarm rate. However, choosing a very high value (such as "max" or "99.99 percentile") can sometimes

result in missing small variations in the input patterns that are caused by more sophisticated attacks. Moreover, the *INDRA* IDS performance is very similar when maximum or 99.99 percentile of validation loss of the final model is selected as the IT. But, in order to capture the attacks that produce small deviations, a slightly smaller IT is selected that would still perform similar to max and 99.99 percentile thresholds under various cyberattack scenarios. Hence, *INDRA* chooses the 99.9th percentile value of the validation loss as the value of the intrusion threshold (IT). The same IT value is used for the remainder of the experiments discussed in the next subsections.

6.3 Comparison of INDRA Variants

After selecting the intrusion threshold using the methodology presented in previous subsection, the performance of *INDRA* framework is evaluated with two other variants: *INDRA*-LED and *INDRA*-LD. The motivation behind evaluating different variants of *INDRA* is to analyze the impact of different layer types in the recurrent autoencoder model on the performance metrics discussed in Sect. 6.1.

The detection accuracy of *INDRA* and its variants is illustrated in Fig. 13a under different attacks and for a no-attack scenario (normal). It can be observed that *INDRA* outperforms the two variants and has high detection accuracy in normal and every attack scenario. The high detection accuracy of *INDRA* is achieved due to its monitoring capability at a signal level unlike the prior works that monitor at the message level.

Figure 13b shows the false-positive rate or false alarm rate of *INDRA* and other variants under different attack scenarios. It is evident that *INDRA* has the lowest false-positive rate and highest detection accuracy compared to the other variants. Moreover, *INDRA*-LED is the second best-performing model after *INDRA*, which leverages the power of GRU-based decoder to reconstruct the original signal values from the MCV. Figure 13a, b clearly shows that the lack of GRU layers in the decoder of *INDRA*-LD resulted in a significant performance degradation. Thus, *INDRA* is chosen as the candidate model for subsequent experiments.

6.4 Comparison with Prior Works

In this subsection, a comparison of the *INDRA* framework with PLSTM [25], RepNet [26], and CANet [23], which are some of the best known prior works in the IDS area, is presented. Figure 14a, b shows the detection accuracy and false-positive rate, respectively, for the various techniques under different attack scenarios.

From Fig. 14a, b, it is evident that *INDRA* achieves a high detection accuracy for each attack scenario and also has low positive rates for most scenarios. The ability to monitor signal level variations along with the more cautious selection of intrusion

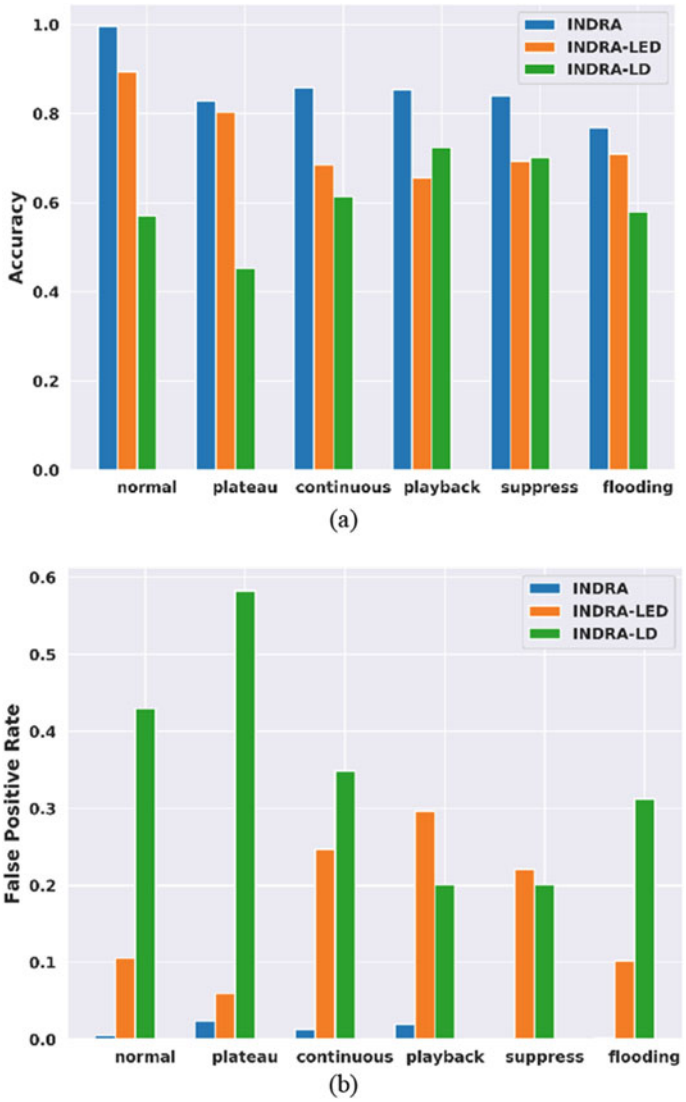

Fig. 13 Comparison of (**a**) detection accuracy and (**b**) false-positive rate for *INDRA* and its variants *INDRA*-LED and *INDRA*-LD under different attack scenarios [33]

threshold gives *INDRA* an advantage over comparison works. Both PLSTM and RepNet use the maximum validation loss in the final model as the threshold to detect intrusions in the system, while CANet uses an interval-based monitoring to detect cyberattacks. The larger threshold value helped PLSTM to achieve slightly lower false-positive rates for few scenarios, but it hurt the ability of both PLSTM and

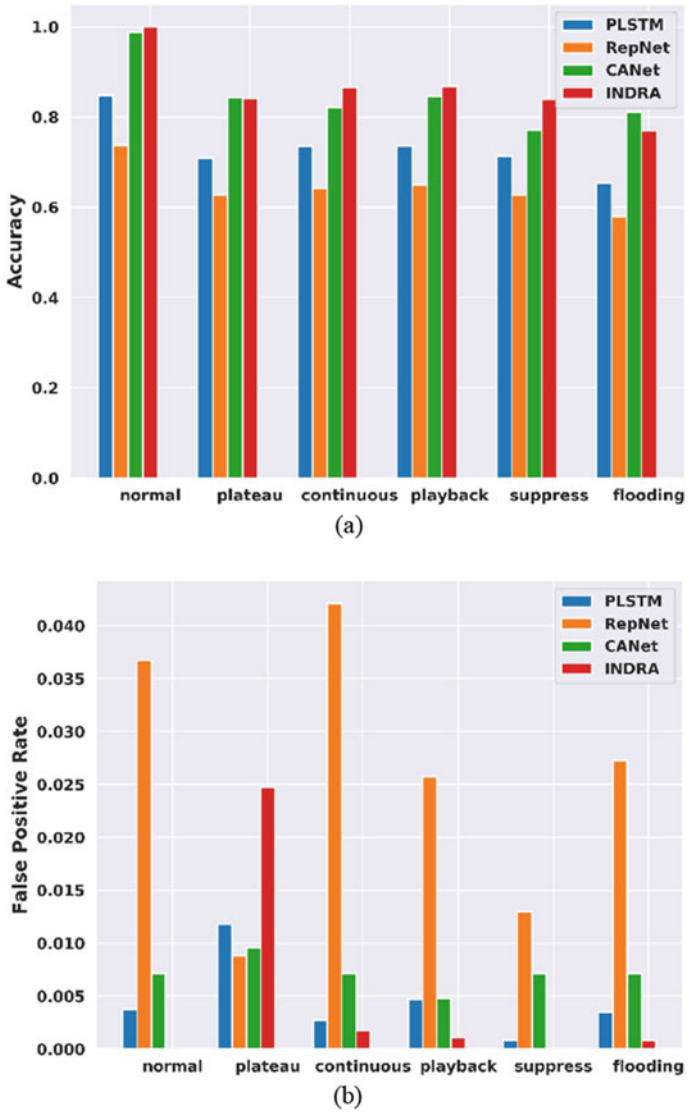

Fig. 14 Comparison of (**a**) detection accuracy and (**b**) false-positive rate of *INDRA* [33] and the prior works PLSTM [25], RepNet [26], and CANet [23]

RepNet to detect cyberattacks that produce small 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 in CANet struggles with finding an optimal threshold value. Lastly, the false-positive rates of *INDRA* are still significantly low with the maximum of 2.5%

Table 2 Memory footprint comparison between *INDRA* framework and the prior works PLSTM [25], REPNET [26], and CANET [23]

Framework	Memory footprint (KB)
PLSTM [25]	13,417
RepNet [26]	55
CANet [23]	8718
INDRA	443

Note: Data in this table is adapted from [33]

for plateau attacks. It is important to note 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 IDS Overhead Analysis

In this subsection, a detailed analysis of the *INDRA* IDS overhead is presented. 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 as the resource-constrained automotive ECUs have limited available memory, and it is crucial to have a low memory overhead to avoid interference with real-time automotive applications. The inference time not only provides important information about the time taken 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* IDS.

To accurately capture the overhead of the *INDRA* framework and the prior works, they are implemented on an ARM Cortex-A57 CPU on a Jetson TX2 board, which has similar specifications to the state-of-the-art multi-core ECUs. The memory footprint of the *INDRA* framework and the comparison works mentioned in the previous subsections are shown in Table 2. It is clear that the *INDRA* framework has a low memory footprint compared to the comparison works, except for the RepNet [26]. However, it is important to observe that even though the *INDRA* framework has slightly higher memory footprint compared to RepNet [26], *INDRA* outperforms all prior works including RepNet [26] in every performance metric under different cyberattack scenarios, as shown in Fig. 14. Even though the heavier (high memory footprint) models can provide the ability to capture a large variety of details about the system behavior, they are not an ideal choice for resource-constrained automotive embedded systems. On the other hand, a much lighter model such as RepNet cannot capture crucial details about the system behavior due to limited parameters and therefore suffers from performance issues.

In order to understand the inference overhead, different IDS frameworks are benchmarked on an ARM Cortex-A57 CPU. In this experiment, different system configurations are considered to encompass a wide variety of state-of-the-art ECU

Table 3 Inference time comparisons between *INDRA* framework and the prior works PLSTM [25], REPNET [26], and CANET [23] using single-core and dual-core configurations

	Average inference time (μs)	
Framework	Single-core ARM Cortex A57 CPU	Dual-core ARM Cortex A57 CPU
PLSTM [25]	681.18	644.76
RepNet [26]	19.46	21.46
CANet [23]	395.63	378.72
INDRA	80.35	72.91

Note: Data in this table is adapted from [33]

hardware in vehicles. Based on the available hardware resources on the Jetson TX2, two different system configurations are selected. The first configuration utilizes only one CPU core (single core), while the second configuration uses two CPU cores (dual core).

Each framework is run ten times for two different CPU configurations, and the average inference time (in μs) is computed, as shown in Table 3. From the results in Table 3, it can be seen that *INDRA* has significantly faster inference times compared to the prior works (excluding RepNet) under all configurations. This is partly associated with the lower memory footprint of the *INDRA* IDS. As mentioned earlier, even though RepNet has a lower inference time, it has the worst performance out of all frameworks, as shown in Fig. 14. The large inference times for the better performing frameworks can impact the real-time performance of the control systems in the vehicle and can result in missing of critical deadlines, which can be catastrophic. These inference times can be further improved by employing a dedicated deep learning accelerator (DLA) compared to the above presented configurations.

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

6.6 Scalability Results

In this subsection, a scalability analysis of the *INDRA* IDS 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 has a real-time utilization (U_{RT}) and an IDS utilization (U_{IDS}) from running real-time and IDS applications, respectively. The IDS overhead (U_{IDS}) is analyzed as a measure of the compute efficiency of the IDS. Since the safety-critical messages monitored by the IDS are periodic in nature, the intrusion detection task can be modeled as a periodic application with a period that is same as the message period [32]. Thus, monitoring an i^{th} message m_i results in an induced IDS utilization

(U_{IDS,m_i}) at an ECU and can be computed as:

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

where T_{IDS} and P_{mi} indicate 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 as a result of monitoring different messages is the overall IDS utilization at that ECU (U_{IDS}) 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 are studied. Moreover, a pool of commonly used message periods {1, 5, 10, 15, 20, 25, 30, 45, 50, 100} (all periods in ms) in automotive systems are uniformly assigned to various messages in the system. These messages are evenly distributed among different ECUs in each system configuration and the IDS utilization is computed using Eqs. (6) and (7). To analyze the worst case overhead, a pessimistic scenario consisting of only a single core per each ECU in the system is considered in this experiment.

The average ECU utilization under various system sizes is illustrated in Fig. 15. The system size is denoted by {p, q}, where p is the number of ECUs and q is the number of messages in the system. Additionally, a very pessimistic estimate of 50% real-time ECU utilization for real-time automotive applications is assumed ("RT Util," as shown in the dotted bars) for all system configurations. The overhead incurred by the IDS executing on the ECUs is represented by the solid bars on top of the dotted bars, and the red horizontal dotted line represents the 100% ECU utilization mark. It is important to avoid exceeding the 100% ECU utilization under any scenario, as it could induce undesired latencies that could result in missing deadlines for time-critical automotive applications, which can be catastrophic. From the results in Fig. 15, it is evident that the prior works PLSTM and CANet incur heavy overhead on the ECUs while RepNet and *INDRA* have a very minimal overhead that scales favorably with increasing system sizes. Thus, from the results in this section (Figs. 14 and 15; Tables 2 and 3), it is apparent that not only does *INDRA* achieve superior performance in terms of both detection accuracy and low false-positive rate for intrusion detection than state-of-the-art prior works, but it is also lightweight and scalable.

7 Conclusion

In this chapter, a novel recurrent autoencoder-based lightweight intrusion detection system framework called *INDRA* for distributed automotive embedded systems was presented. The *INDRA* framework uses the proposed metric called the intrusion score (IS) to quantify the deviation of the prediction signal from the actual input signal. Moreover, a thorough analysis of the intrusion threshold selection process

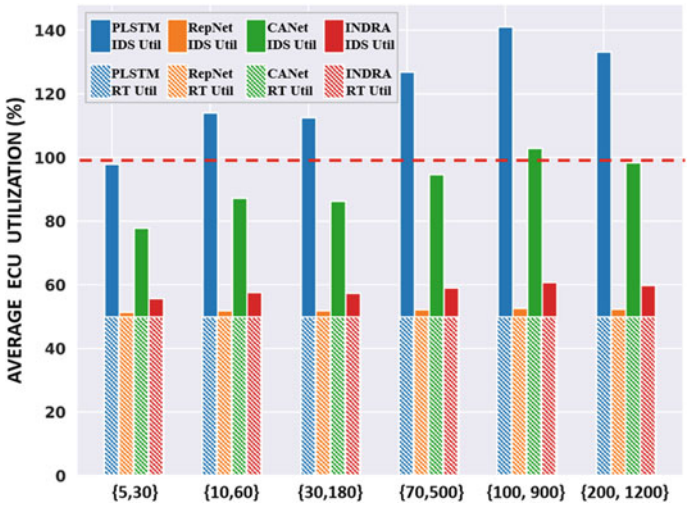

Fig. 15 Scalability results of the *INDRA* [33] IDS for different system sizes compared to the prior works PLSTM [25], RepNet [26], and CANet [23]

and the comparison of *INDRA* with the best known prior works in this area is presented in this chapter. The promising results of *INDRA* indicate a compelling potential for being adapted to enhance cybersecurity in emerging automotive platforms. Our ongoing work is exploring newer and more powerful algorithms [34–36] for intrusion detection in automotive embedded systems.

Acknowledgments This research is supported by a grant from NSF (CNS-2132385).

References

1. Kukkala, V., Tunnell, J., Pasricha, S.: Advanced driver assistance systems: a path toward autonomous vehicles. IEEE Consum. Electron. **7**(5), 18–25 (2018)
2. Koscher, K., et al.: Experimental security analysis of a modern automobile. In: IEEE SP (2010)
3. Valasek, C., et al.: Remote exploitation of an unaltered passenger vehicle. https://ioactive.com/pdfs/IOActive_Remote_Car_Hacking.pdf (2015)
4. Izosimov, V., et al.: Security-aware development of cyber-physical systems illustrated with automotive case study. In: IEEE/ACM DATE (2016)
5. Studnia, I., et al.: A language-based intrusion detection approach for automotive embedded network. In: IEEE PRISDC (2015)
6. Marchetti, M., et al.: Anomaly detection of CAN bus messages through analysis of ID sequences. In: IEEE IV (2017)
7. Hoppe, T., et al.: Security threats to automotive CAN networks- practical examples and selected short-term countermeasures. In: RESS (2011)
8. Larson, U.E., et al.: An approach to specification-based attack detection for in-vehicle networks. In: IEEE IV (2008)

9. Aldwairi, M., et al.: Pattern matching of signature-based IDS using Myers algorithm under MapReduce framework. In: EURASIP (2017)
10. Myers, E.W.: An O(ND) difference algorithm and its variations. In: Algorithmica (1986)
11. Hoppe, T., et al.: Applying intrusion detection to automotive IT-early insights and remaining challenges. In: JIAS (2009)
12. Waszecki, P. et al.: Automotive electrical and electronic architecture security via distributed in-vehicle traffic monitoring. In: IEEE TCAD (2017)
13. Cho, K.T., et al.: Fingerprinting electronic control units for vehicle intrusion detection. In: USENIX (2016)
14. Ying, X., et al.: Shape of the Cloak: formal analysis of clock skew-based intrusion detection system in controller area networks. In: IEEE TIFS (2019)
15. Yoon, M.K., et al.: Memory heat map: anomaly detection in real-time embedded systems using memory behavior. In: IEEE/ACM/EDAC DAC (2015)
16. Müter, M., et al.: Entropy-based anomaly detection for in-vehicle networks. In: IEEE IV (2011)
17. Müter, M., et al.: A structured approach to anomaly detection for in-vehicle networks. In: ICIAS (2010)
18. Taylor, A., Japkowicz, N., Leblanc, S.: Frequency-based anomaly detection for the automotive CAN bus. In: Proceedings of WCICSS (2015)
19. Martinelli, F., et al.: Car hacking identification through fuzzy logic algorithms. In: FUZZ-IEEE (2017)
20. Vuong, T.P., et al.: Performance evaluation of cyber-physical intrusion detection on a robotic vehicle. In: IEEE CIT/IUCC/DASC/PICOM (2015)
21. Levi, M., Allouche, Y., et al.: Advanced analytics for connected cars cyber security [Online]. Available: https://arxiv.org/abs/1711.01939 (2017)
22. Kang, M.-J., et al.: A novel intrusion detection method using deep neural network for in-vehicle network security. In: IEEE, VTC Spring (2016)
23. Hanselmann, M., et al.: CANet: an unsupervised intrusion detection system for high dimensional CAN bus data: In: IEEE Access (2020)
24. Loukas, G., et al.: Cloud-based cyber-physical intrusion detection for vehicles using deep learning. In: IEEE Access (2018)
25. Taylor, A., et al.: Anomaly detection in automobile control network data with long short-term memory networks. In: IEEE DSAA (2016)
26. Weber, M., et al.: Online detection of anomalies in vehicle signals using replicator neural networks. In: ESCAR (2018)
27. Weber, M., et al.: Embedded hybrid anomaly detection for automotive can communication. In: Embedded Real Time Software and Systems (2018) [Online]. Available: https://hal.archives-ouvertes.fr/hal-01716805
28. Schmidhuber, J.: Habilitation thesis: system modeling and optimization (1993)
29. Hochreiter, S., et al.: Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In: IEEE Press (2001)
30. Cho, K., et al.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: EMNLP (2014)
31. DiDomenico, G., et al.: Colorado State University EcoCAR 3 final technical report. In: SAE, WCX (2019)
32. Kukkala, V., Pasricha, S., Bradley, T.H.: SEDAN: Security-aware design of time-critical automotive networks. IEEE Trans. Veh. Technol. (TVT) **69**(8), 9017–9030 (2020, August)
33. Kukkala, V.K., Thiruloga, S.V., Pasricha, S.: INDRA: intrusion detection using recurrent autoencoders in automotive embedded systems. IEEE Trans. Comput-Aid. Des. Integr. Circuits Syst. (TCAD) **39**(11) (2020, November)
34. Thiruloga, S.V., Kukkala, V.K., Pasricha, S.: TENET: temporal CNN with attention for anomaly detection in automotive cyber-physical systems. In: IEEE/ACM Asia & South Pacific Design Automation Conference (ASPDAC) (2022, January)

35. Kukkala, V.K, Thiruloga, S.V., Pasricha, S.: LATTE: LSTM self-attention based anomaly detection in embedded automotive platforms. In: ACM Transactions on Embedded Computing Systems (TECS) (2021)
36. Kukkala, V.K., Thiruloga, S.V., Pasricha, S.: Roadmap for cybersecurity in autonomous vehicles. In: IEEE Consumer Electronics (2022)