IDS for SOME/IP attacks using LSTM and Auto-Encoders

Bachelor Thesis Proposal
Bachelor thesis in the department of Law and Economics by Munoz Nico (Student ID: 2707958)
Date of submission: September 2, 2024

1. Review: Prof. Dr. Michael Waidner
2. Review: Christian Plappert, M.Sc.
3. Review: Florian Fenzl, M.Sc. [Fraunhofer SIT]
Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Law and Economics Department
Business Informatics
Fraunhofer SIT - Automotive Security

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB TU Darmstadt

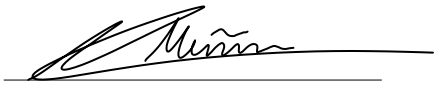
Hiermit erkläre ich, Munoz Nico, dass ich die vorliegende Arbeit gemäß § 22 Abs. 7 APB der TU Darmstadt selbstständig, ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe mit Ausnahme der zitierten Literatur und anderer in der Arbeit genannter Quellen keine fremden Hilfsmittel benutzt. Die von mir bei der Anfertigung dieser wissenschaftlichen Arbeit wörtlich oder inhaltlich benutzte Literatur und alle anderen Quellen habe ich im Text deutlich gekennzeichnet und gesondert aufgeführt. Dies gilt auch für Quellen oder Hilfsmittel aus dem Internet.

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, September 2, 2024



M. Nico

Abstract

Increased connectivity has significantly impacted the automotive industry, transforming vehicles from mere means of transport into intelligent tools that rely on various communication protocols within the In-Vehicle Network (IVN). To handle the growing demand for bandwidth, new communication protocols are emerging, including the standardized Scalable service-Oriented MiddlewarE over IP (SOME/IP) protocol. However, SOME/IP was implemented without basic security features, making it vulnerable to cyber attacks such as tampering or replay. To detect attacks on the protocol's payload, an Intrusion Detection System (IDS) can be utilized. IDS systems can be implemented in various ways. In this thesis, the feasibility of an hybridized Long Short-Term Memory and Auto-Encoder (LSTM-AE) Machine Learning (ML) model architecture was implemented and evaluated. The LSTM-AE model leverages unsupervised learning to effectively compress and reconstruct data, comparing the reconstructed data to the original input. A high reconstruction error could indicate an attack on the system. The Long Short-Term Memory (LSTM) component adds time dependence to the model, enabling it to detect attacks across entire data sequences. Four different models were created and compared using data from an IVN simulation, depicting four service nodes communicating over the SOME/IP protocol and transmitting sensor and processed data over the network. The models performance was evaluated in detecting attacks on both entire sequences and individual packets. The results indicate that LSTM-AE models are not effective in identifying replay attacks and perform poorly in packet-level detections, with an accuracy of 56.82%. However, at the sequence level, the models can detect tamper attacks with an accuracy of 99.90%. Additionally, the LSTM-AE architecture exhibits interesting characteristics in terms of explainability, which require further investigation.

Contents

1. Introduction

The automotive landscape has undergone significant transformations from its earliest stages to the present day. In modern vehicles, most in-vehicle functionalities are controlled by Electronic Control Units (ECU). As technology progresses, features like automotive entertainment systems, Advanced Driver-Assistance Systems (ADAS), and autonomous driving become increasingly complex. As a result, the demand for high bandwidth networks such as Automotive Ethernet (AETH) becomes essential to support software-based features. Vehicles whose features are primarily enabled through software are known as Software-defined Vehicles (SDVs). A key aspect of a SDV is enabling service-oriented communication, a trend supported by the introduction of SOME/IP.

1.1. Motivation

Marco De Vincenzi et al. [8] conducted a systematic review on security attacks and countermeasures in AETH, which emphasizes that adopting Ethernet for data transfer within the IVN backbone or specific IVN domains could significantly enhance communication within the vehicle network. Among other things, papers focusing on different attack types, and different security approaches were taken into consideration. However, the implementation of security protocols may lead to latency, which is a major concern in the automotive industry. To tackle this concern, IDSs have proven to be effective tools for detecting network attacks [16] and offer the advantage of imposing minimal latency impact. These IDSs, mostly implemented using rule-based or AI-based approaches, provide alerts and contain information such as the attack type. Various studies regarding the mitigation approaches are reviewed by Marco De Vincenzi et al., but IDS solutions focusing on the SOME/IP protocol are scarce. This is problematic because the emerging SOME/IP protocol was designed lacking basic security features, therefore making it vulnerable to various cyberattacks [36, 15]. The few papers which address the security

aspects alongside IDSs mostly focus on specific attack types on the header [10, 1, 9]. Hnamte et al. [16] demonstrated the feasibility and effectiveness of a hybrid solution for a network IDS, producing promising results in detecting modern attack scenarios. This approach can also be transferred to an IDS system for vehicles. Due to the complex communication and attack sequences of the SOME/IP protocol, rule-based approaches are mostly insufficient, to secure the full range of vulnerabilities. The advantage of an Artificial Intelligence (AI)-based anomaly detection approach, is the possibility to detect and flag more complex and perhaps unseen attacks. LSTM-AEs has proven to be a valid concept for understanding complicated, time-dependent communication patterns. When considering that security in safety-critical domains is a major concern, it is only reasonable to further investigate the explainability of AI-based IDS, to understand if and why anomalies in network traffic are tagged as such. However, to my knowledge, there hasn't been a IDS solution utilizing LSTM-AE and incorporating explainability to the detection results. This could be attributed to the scarcity of available real-world network traces that often lack known attack traffic specific to SOME/IP communication which makes it hard to train an AI-based IDS. Therefore, I propose in this thesis the development of an explainable IDS for SOME/IP attacks using the hybridized Deep Learning (DL) techniques LSTM and Auto-Encoders (AEs).

1.2. Research Questions

In this thesis I will focus on answering the following questions:

How feasible is an IDS for SOME/IP attacks using LSTM-AE?

Considering that AETH is primarily intended for the use in safety-critical domains such as IVN, it is crucial to develop security measures that are robust, lightweight, and fast. This ensures that benign network flow remains uninterrupted or delayed by the IDS while still effectively detecting malicious activity.

How accurately can various attacks be identified?

Insufficient security in safety-critical domains can result in death, environmental hazards, and significant damage [9]. Hence, striving to make attacks impossible is imperative. SOME/IP attacks have been discussed in various papers ranging from Man-in-the-middle (MITM) attacks, realized through *De-association*, *Copycat*, *Publish/Subscribe* techniques [36] to *Fuzzy*, *Spoof* and the *Denial of Service* attacks [10, 26]. Furthermore, there

are specific attacks that exploit SOME/IPs implementation, such as the *Wrong Interface attack* and the *Error-On-Event attack*. Consequences of these attacks lead to unpredictable behavior and abnormal communication processes such as error on error, error on event, missing response, and missing request [1, 10, 9, 26]. Additionally, they can result in unauthorized operations including subscription, unsubscription, provision of services, and Remote Procedure Calls (RPC) [26, 36]. Various metrics are available to assess an IDSs performance, aiming to determine its accuracy and precision in detecting attacks, as well as to identify and potentially prevent overfitting. Among these metrics, the most commonly used in the previously examined papers was the F1-Score.

How explainable are the results?

It is important to assess whether the LSTM-AE Model can produce verifiable results. Most decisions made by ML systems, including those based on artificial Neural Networks (NN), rely on probabilities [10]. Understanding why certain network activities are classified as dangerous and vice versa becomes challenging. This inherent uncertainty can lead to false positives and false negatives, which in turn can make the IDS unpredictable and in the worst cases, potentially deadly. To avoid relinquishing responsibility in safety-critical environments, an IDS that returns comprehensible results can be used to strengthen trust in the systems. This also offers the possibility for humans to review incident reports in centralized Security Operation Centers and maintain the system effectively.

1.3. Thesis Structure

Following the motivation for this thesis and the discussion of key problems, the remainder of this work is structured as follows:

Chapter 2 eprovides the necessary background knowledge to understand this paper. It covers topics such as SDV, IVNs, the SOME/IP protocol, and IDS. Additionally, the LSTM-AE architecture is explained in the context of AI.

Chapter 3 reviews previous research and places it in the context of this thesis.

Chapter 4 details the proposed LSTM-AE system, including its implementation and the evaluation approach, as well as the tuning of the model.

Chapter 5 evaluates the different models implemented in terms of attack detection. Additionally, the explainability of the LSTM-AE model architecture is explored.

Chapter 6 compares the different models and discusses their performances. Future work is also proposed at the end of this chapter.

In Chapter 7 the different models are compared and the performances are discussed. Moreover, future work is proposed at the end of this chapter.

The thesis concludes in Chapter 8.

2. Background

This chapter provides the essential background knowledge necessary to understand the following chapters. At first, an introduction to SDVs is given, transitioning to the topic of IVNs, and AETH, followed by a deeper insight into the SOME/IP protocol. Afterward, information about IDS focusing on the automotive domain is provided. The second part conveys general information about ML, NN, and AI. The use of IDS is particularly emphasized in this context, laying the foundation for understanding the DL LSTM and AE approach used in this work and looked at in more detail in this chapter. The chapter concludes with a discussion on eXplainable Artificial Intelligence (XAI) and its significance in this specific use case.

2.1. Software-defined Vehicle (SDV)

Vehicle manufacturers used to differentiate themselves with features like horsepower and maximum speed. This longstanding status quo, which primarily emphasized mechanical hardware components in vehicle development, is now transitioning towards a model driven by software [33]. Consequently, the quantity and value of software, including associated electronic hardware, surpasses that of increasingly standardized mechanical hardware. This opens the door for more intelligent and upgradable systems with features and functions primarily enabled through software [2].

This transformation can be compared to that of cell phones. Initially, hardware and software were tightly integrated. However, with the introduction of smartphones, the device became a platform capable of hosting various applications. Today, consumers seek more sophisticated software-enabled features, like autonomous driving, infotainment, and connectivity [14].

Traditionally, upgrades required a trip to a dealership or were not feasible at all. However, with the introduction of SDV, customers will be able to receive new features through

over-the-air updates, whether at home or on the road. Information will travel both ways, allowing manufacturers to gain insights into various aspects of the vehicle's performance. [13].

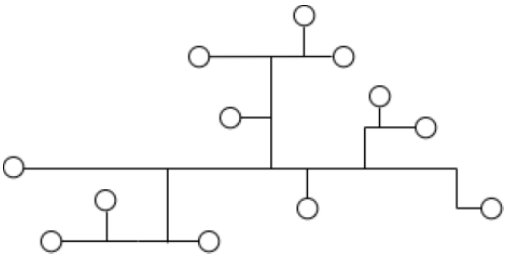
2.2. In-Vehicle Network (IVN)

> **Disclaimer:** *Given that the automotive industry comprises numerous car manufacturers and multiple additional stakeholders, it is reasonable to assume that a lot of changes happened non-dependent with standards derived from multiple sources based on different innovations and research. Hence, this chapter mostly draws upon the research conducted by Kirsten Matheus and Thomas Königseder for their book "Automotive Ethernet", published by Cambridge University Press. The focus here is the development process of AETH by Bayerische Motoren Werke (BMW), an innovation leader in the IVN domain.*

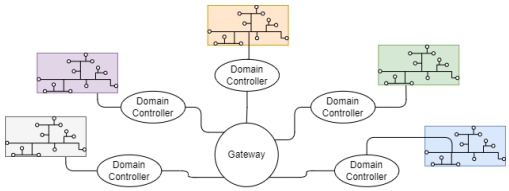
One major domain affected by the transition described in Chapter 2.1 is the IVN domain. IVN became relevant with new electrical-powered comfort features such as electrically adjustable windows, central door locking, and electronic side mirror adjustment. Originally every feature had its own wire, which led to a lot of wires going through the same locations. The wiring became so heavy, costly, hard to install, and error-prone that a new approach was sought. The solution were bus systems in which the intelligence lay in ECUs. This not only solved a lot of the aforementioned problems but also brought new possibilities, such as activating the light when the car is being unlocked. Over the years and through numerous innovation cycles, the vehicle's inner workings became increasingly more complex with enhanced comfort features and driving functionality being substituted by electronics [29].

The IVN of vehicles has evolved from the traditional flat architecture to a domain-specific approach, and most recently, to a zonal architecture. The flat architecture (Figure 2.1a) is characterized by its distinct application-oriented design. Each ECU mostly communicates with another specific ECU leading to a 1-to-1 communication scheme. The domain-specific architecture (Figure 2.1b) was created to adapt to the increasingly complex in-vehicle functionalities. In this approach, the logic of multiple functions and ECUs are consolidated into Domain Controller Units (DCUs) that are connected to a central gateway. Each domain manages specific functions such as safety, energy management, perception, connectivity, body/comfort, and infotainment. The most recent trend in vehicle architecture is the zonal approach (Figure 2.2), which further simplifies and enhances the management of vehicle functions. To support the highly computational needs of modern vehicles, the car is divided

into physical zones (e.g. rear left of the car) and centralized around a Vehicle Processing Unit (VCU) [35]. This progression reflects the automotive industry's response to the increasing complexity, and rapidly changing demands of modern vehicle systems. To meet these demands and tackle resource-intensive advancements, the development of a new IVN technology that is flexible, scalable, and capable of evolving with emerging requirements was essential. In many cases, when facing new complex challenges, it helps to look at existing solutions and repurpose them into something else. The automotive industry embraced that approach and transformed traditional Ethernet, a proven technology, to fit their unique requirements. AETH differs from traditional IVN technologies by being inherently packet-based, high-speed, and switched, making it well-suited for modern automotive applications [29].



(a) Flat Architecture, the vehicle components are interconnected without hierarchy.



(b) Domain Architecture, the network is organized into domains controlled by domain controllers.

Figure 2.1.: Flat-based and Domain-based in-vehicle architecture based on [35].

2.2.1. Automotive Ethernet (AETH)

Even though implementing Ethernet in the automotive domain was uncharted territory and raised questions of feasibility, initial testing, and the following results were very promising. For the first time, free software stacks (e.g., lightweight IP stacks) could be utilized in the development process. This enabled vehicles to be used as nodes within a network, allowing them to connect to multiple external nodes, such as testers and servers. Consequently, this provided a novel opportunity to locate and access a car in any workshop worldwide and to use ordinary PCs to install test software. Originally, the new AETH technology was only used for programming and diagnostic purposes and not during runtime [30]. This was due to the use of 100BASE-TX unshielded twisted pair cables [19] which would interfere, through electromagnetic emissions, with the vehicle

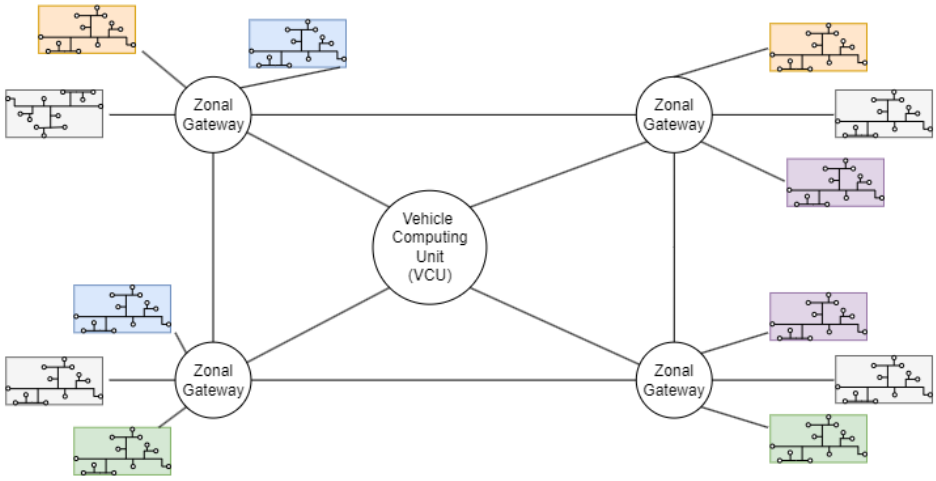


Figure 2.2.: Zone-based in-vehicle architecture based on [35], illustrating how zonal gateways connect physical sections of the vehicle network to a central VCU

during runtime. At the time, utilizing shielded Ethernet [20] during runtime to reduce the interference was considered too expensive [30].

The breakthrough for Ethernet communication was made by *Broadcom Inc.*, who developed a technology called BroadR-Reach [7] on behalf of BMW. This technology made transmitting 100 Mbps over a single pair of unshielded cables possible without interfering with the vehicle during runtime, whereas the earlier 100BASE-TX solution required two cable pairs. One of the first pilot applications for AETH was the Surround View System (SVS), which proved to be an optimal initial use case with encouraging results. To standardize AETH, the One Pair EtherNET (OPEN) Alliance was established. This provided the opportunity to align the market to a single solution and consolidate Ethernet-based communication as an IVN technology. AETH was highly attractive because of its scalability potential and flexibility. While *Broadcom Inc.'s* 100 Mbps solution was a good first step, to provide a future-proof networking technology an automotive-suitable Gbps Ethernet solution was desirable. Consequently, the 1000BASE-T1 standard was developed by the IEEE 802.3bp task force, introducing the automotive industry as a new application field for the IEEE 802.3 working group [30]. This group develops a collection of standards that define the physical layer and the Media Access Control (MAC) of the data link layer for wired Ethernet. The IEEE, short for the Institute of Electrical and Electronics Engineers, is the

world's largest technical professional organization dedicated to advancing technology for the benefit of humanity, according to their website [17].

Since AETH encompasses all layers of the Open Systems Interconnection (OSI) layering model and not only the physical layers described in this chapter, many different organizations have worked on its development [30]. One of these organizations is the AUTomotive Open System ARchitecture (AUTOSAR) development partnership. A major objective of AUTOSAR is to establish a standard for software for future intelligent mobility. A more recent development is the communication protocol named SOME/IP [3].

2.3. Scalable service-Oriented MiddlewarE over IP (SOME/IP)

The SOME/IP protocol was initially developed exclusively for automotive applications, it was designed to fit a variety of domains within the vehicle. Since its inception, it has undergone continuous development and is increasingly recognized as the predominant future protocol for IVN. In this section, the general specifications and common requirements outlined in the AUTOSAR Foundation standards for SOME/IP [4, 5], are used to explain the basic concepts and functionalities of SOME/IP.

2.3.1. SOME/IP Protocol Specification

The SOME/IP protocol is located between the transport and application layers of the OSI communication layer model. It standardizes communication between ECUs, using services that provide different functionalities to which clients can subscribe to or interact with. These functionalities can consist of a combination of zero or multiple events, methods, and fields. Events provide data periodically or on change to the subscriber. Whereas RPCs are realized through methods, executed on the provider side. Fields consist of one or more notifiers, getters, or setters. Notifiers, akin to events dispatch data based on changes, epsilon changes, or cyclic configurations but with the minor difference that data is sent immediately after subscribing to the service and not just on changes. Getters provide subscribers the functionality to explicitly request data, while setters allow the data values to be modified on the provider side.

Figure 2.3 outlines the header format and the attached payload of SOME/IP packets.

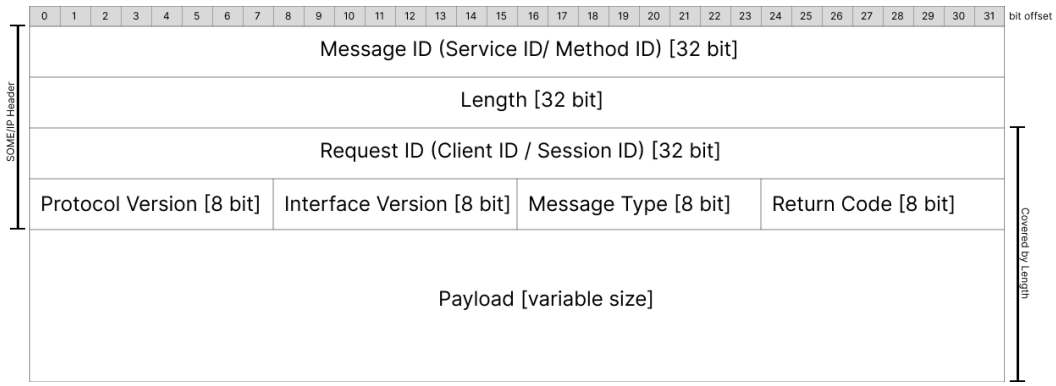


Figure 2.3.: SOME/IP Packet Format based on [4]

Message ID

This field is represented with more detail in Figure 2.4 and consists of unique 16-bit identifiers for the Service ID and Method/Event ID predetermined by the developer. It is common practice that the ID space is divided between Methods and Events. In particular, methods are allocated the range from 0x0000 to 0x7FFFF. Conversely, Events occupy the range from 0x8000 to 0x8FFF. This ensures that Methods and Events can be distinguished based on the first bit.

Figure 2.4.: Structure of the Message ID based on [4]

Length

Represents the total number of bytes from the Request ID/Client ID field to the end of the SOME/IP packet.

Request ID

The Request ID, as shown in Figure 2.5, is split into two parts. These are the Client ID and the Session ID, both of which should be unique for each request-response pair. The Client ID identifies the client making the request, whereas the Session ID serves to differentiate among various sessions or requests originating from the same client. This is crucial to distinguish between multiple concurrent uses of the same method, getter, or setter.

Figure 2.5.: Structure of the Request ID based on [4]

Protocol Version

Represents the used version of the SOME/IP protocol

Interface Version

Contains the Major Version of the Service

Message Type

Several important Message Types are depicted in the Table 2.1.

ID	Name	Description
0x00	REQUEST	A request expecting a response (even void)
0x01	REQUEST_NO_RETURN	A fire and forget request
0x02	NOTIFICATION	A request for a notification/event callback expecting no response
0x80	RESPONSE	The response message
0x81	ERROR	The response containing an error

Table 2.1.: Possible Message Types based on [4]

Return Code

Signals if a request was processed successfully. Several important Return Codes are depicted in Table 2.2.

ID	Name	Description
0x00	E_OK	No error occurred
0x01	E_NOT_OK	An unspecified error occurred
0x02	E_UNKNOWN_SERVICE	Requested Service ID unknown.
0x03	E_UNKNOWN_METHOD	Requested Method ID unknown but Service ID is known.
0x04	E_NOT_READY	Service ID, Method ID known but the application is not running.
0x05	E_NOT_REACHABLE	System running the service is unreachable (internal error code only).
0x07	E_WRONG_PROTOCOL_VERSION	Version of SOME/IP protocol not supported
0x08	E_WRONG_INTERFACE_VERSION	Interface version mismatch
0x0a	E_WRONG_MESSAGE_TYPE	An unexpected message type was received (e.g. REQUEST_NO_RETURN for a method defined as REQUEST).

Table 2.2.: Possible Return Codes based on [4]

2.3.2. SOME/IP Service Discovery (SOME/IP-SD)

Service Discovery (SD) is an essential part of the SOME/IP protocol, enabling communication by dynamically offering and discovering services within the network, as well as managing the transmission behavior of event messages. The SOME/IP Service Discovery (SOME/IP-SD) header is based on the SOME/IP packet visualized in Figure 2.3 with an additional section field containing the SD information (Figure 2.6). The main objective is to communicate available functional entities called services to the IVN, essentially making it possible to establish and maintain connections between multiple different ECUs.

SOME/IP-SD data is divided into Flags, Reserved, Entry, and Options fields. More infor-

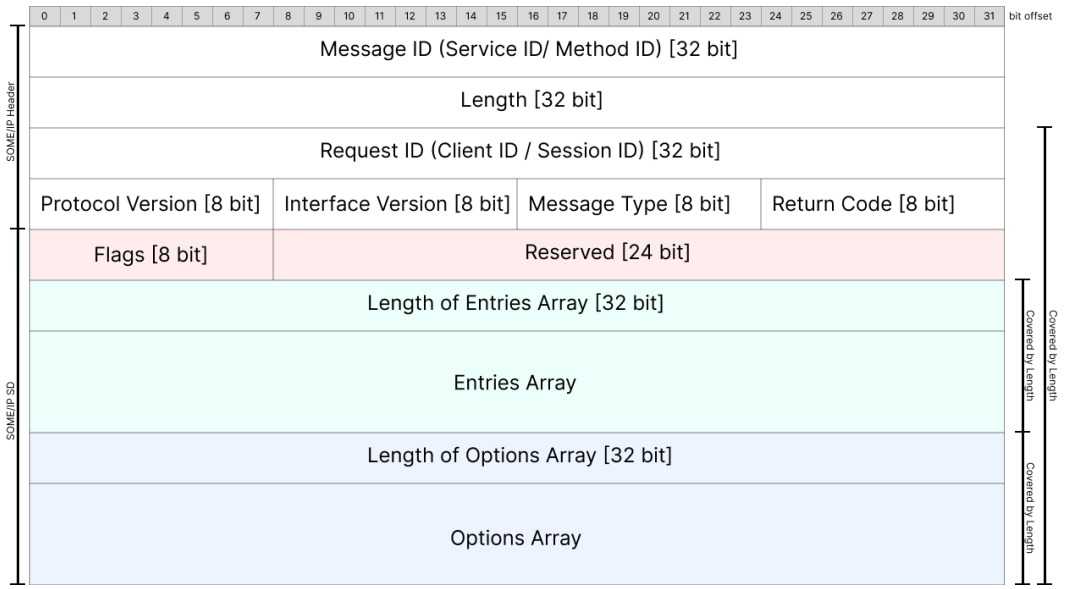


Figure 2.6.: SOME/IP-SD Header Format based on [5]

mation about the Entries and Options field is provided in the following paragraphs.

Entries

SOME/IP-SD supports multiple entries within a single discovery message. These entries are utilized to synchronize the state of service instances and manage the basic Publish/-Subscribe communication pattern. A distinction is made between two groups: The Service Entry, as represented in Figure 2.7, and the Eventgroup Entry Type, shown in Figure 2.8.

The **Type** field of the Service Entries is needed to find a service (*FindService* (0x00)), offer a service(*OfferService* (0x01)), and stop offering a service (*StopOfferService* (0x01)). This is set in the first 8 bits of the entry. The following 24 bits determine which option arrays are executed and the number of options used. The next 32 bits are reserved for the **Service ID** and **Instance ID**, which are used to identify a service and its instance. This is followed by the **Major Version**, which encodes the version of the service instance to which this eventgroup belongs. Another important field is the **Time-to-Live (TTL)** field, which specifies the entry's lifetime in seconds. Setting the TTL to zero results in the termination of a service offering.

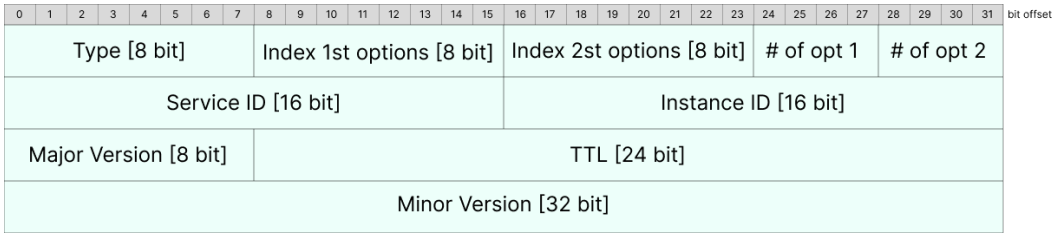


Figure 2.7.: SOME/IP-SD Service Entry Type based on [5]

The **Type** field of the Eventgroup Entries manages subscribing (*SubscribeEventgroup* (0x06)), unsubscribing (*StopSubscribeEventgroup* (0x06)), acknowledging a subscription (*SubscribeEventgroupAck* (0x07)), and rejecting a subscription (*SubscribeEventgroupNack* (0x07)). The subsequent fields are identical to those in the Service Entry, except for the last 32 bits. These bits are divided into a **Reserved** field, a **Counter** field to differentiate identical *SubscribeEventgroup* messages from the same subscriber, and an identification field for the Eventgroup (**Eventgroup ID**).

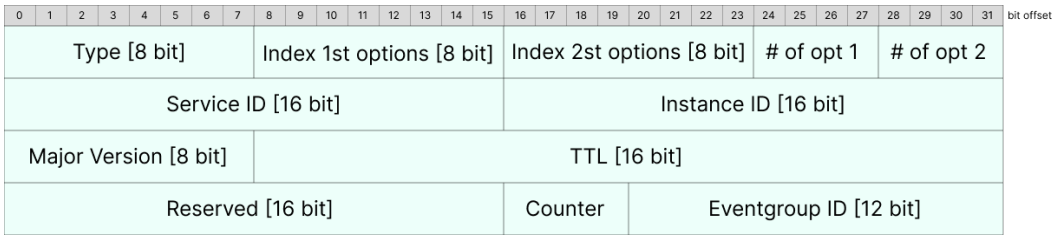


Figure 2.8.: SOME/IP-SD Eventgroup Entry Type based on [5]

Options

Options are referenced by the Entries and are used to transport additional information. This includes details on how a service instance is reachable, such as IP address, transport protocol, and port number. Each option begins with a **Length** field, which represents how many bits the option encompasses. This is followed by a **Type** field that determines the option type and the subsequent structure of the Options. The currently supported option **Types** include: Configuration Option (0x01), Load Balancing Option (0x02), IPv4/IPv6 Endpoint Option (0x04, 0x06), IPv4/IPv6 Multicast Option (0x14, 0x16), and IPv4/IPv6 SD Endpoint Option (0x24, 0x26). The subsequent fields are the **Discardable Flag** indicating whether the option can be discarded, and a **Reserved** field.

Figure 2.9 illustrates the IPv4-SD Endpoint Option, utilized to transport the endpoint details (IP address, protocol, and port) from the Service to the Client. This option is employed with an *OfferService* message, which specifies how clients should connect to the service, and *SubscribeEventgroupAck* message to confirm the connection. The IPv4-Address field contains the unicast IP address of SOME/IP-SD which typically is configured to the transport layer protocol of SOME/IP-SD (default: 0x11 UDP). The Port Number is typically set to the transport layer port of SOME/IP-SD (default: 30490). Notably, clients are also required to communicate their IP address, protocol, and port details to ensure the service can effectively dispatch event notifications.

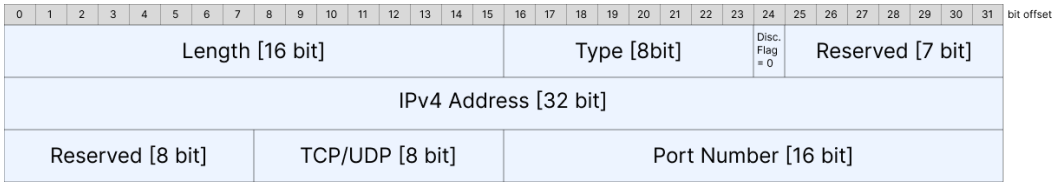
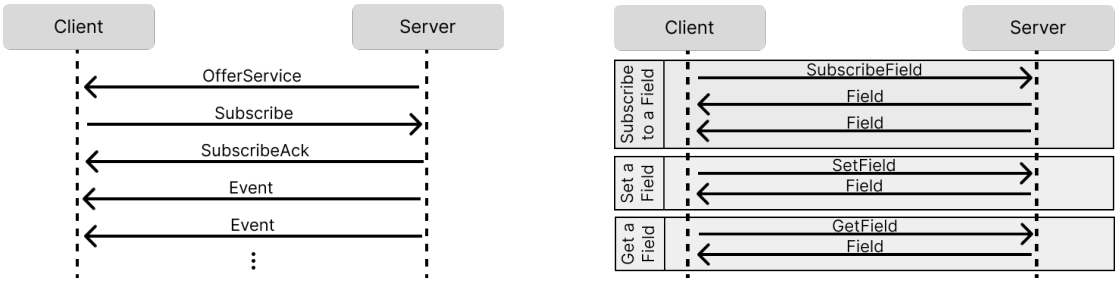


Figure 2.9.: SOME/IP-SD IPv4 Endpoint Option based on [5]

2.3.3. SOME/IP Communication Patterns

The basic SOME/IP communication patterns are depicted in Figures 2.10 and 2.11.

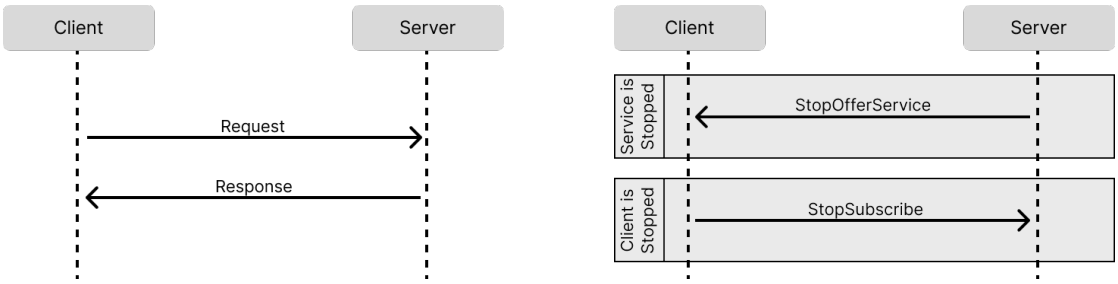
One of the most common communication patterns is depicted in Figure 2.10a. A server can publish services to which clients can subscribe. The server is now able to transmit



(a) Publish and Subscribe

(b) Field Communication

Figure 2.10.: Subscription to Eventgroup and Field Communication



(a) Request and Response (RPC)

(b) Stop Messages

Figure 2.11.: RPC and Stop Communication

data to clients via Events. Notably, the publish and subscribe mechanism is handled by SOME/IP-SD, the data however is transmitted via plain SOME/IP.

In Figure 2.10b the different options for Filed communication are shown. A field represents a status or a property and has a valid value. The client will receive the field's value immediately after subscribing. The current value can be retrieved by calling a *Get* function, the value of the Field can be set by calling a *Set* function and the *Notifier* functionality sends an event on change or cyclically. Compared with regular events, Fields represent an underlying property that is available at all times that the parent service is alive, while the Event is only valid at the time the event is happening.

The request and response scheme (Figure 2.11a), allows the client to request information from a server, who in turn provides the information to the client. In addition, it is also possible to send a fire&forget request. Hence, the client will expect no response from the server.

To stop the communication between the client and server, either the server can send a *StopOfferService* message or the client can end his subscription with a *StopSubscribe* message.

2.4. Intrusion Detection System (IDS)

An Intrusion Detection System is a fundamental cybersecurity mechanism intended to detect various cybersecurity violations, ranging from external attacks to the impersonation of legitimate users or insiders abusing their privileges [6]. IDS are usually deployed on

host and/or network environments. A host-based system analyzes mainly events related to the operating system, whereas network-based systems focus on network-related events, including packet headers, connection attempts, and data flow patterns. Additionally, the literature distinguishes between signature-based and anomaly-based intrusion detection. Signature-based systems use predefined patterns and strategies to detect mostly known attacks. Anomaly-based architectures, on the other hand, estimate the system's ”normal” behavior and trigger an alarm when deviations from this behavior exceed a predefined threshold [31, 11]. Anomaly-based systems are typically further subdivided into three categories: statistical-based, knowledge-based, and ML-based. In statistical-based systems, no prior knowledge about ”normal” activities is required. The IDS behavior is represented from a probabilistic perspective by using statistical properties and principles. Knowledge-based systems on the other hand aim to capture the system's claimed behavior using available data such as protocol specifications and network traffic instances. ML-based systems utilize machine learning algorithms to create models that can identify complex patterns and relationships within the data [22]. To implement an IDS for the SOME/IP protocol using LSTM-AE the focus in this thesis is set on an Anomaly-based Network-IDSs incorporating ML.

2.5. Artificial Intelligence (AI)

The task of defining Artificial Intelligence is not as straightforward as it initially seems. This derives from the problem that intelligence itself is something humans struggle to define properly to this day. Essentially the field of AI is concerned with understanding and developing machines that can compute how to act effectively and safely in various situations. The field of AI encompasses multiple areas, such as computer vision, robotics, machine learning, neural networks, and much more [34].

2.5.1. Machine Learning (ML)

Traditionally, algorithms are sets of clear instructions designed to carry out specific tasks. They process an input according to a predetermined pattern of actions, producing outputs. However, some problems are too complex to establish a fixed pattern. As a result, a new type of algorithm, capable of learning from data, is required. These are known as Machine Learning Algorithms. Although they are governed by rules and do not ”think” or are ”creative”, Machine Learning Algorithms are still more flexible and can be used to solve

many variations of the same problem. Their rules are derived from the data provided and the algorithm's implementation. The process through which the algorithm learns these rules from the data is called training. This process produces a model that represents the patterns and relationships within the data, continuously refining its internal structure to better capture the complexities of the data it is learning from. The main objective is to produce a model that can effectively solve problems within the same domain from which the data was derived. Typically, Machine Learning Algorithms can be categorized into three main types based on the type of training needed: Reinforcement Learning, Supervised Learning, and Unsupervised Learning[21].

In **Reinforcement Learning**, the main objective is to find a policy or strategy to maximize rewards or to minimize punishments. The model gets adjusted accordingly. In **Supervised Learning**, the model is trained on labeled data, the objective is to learn a function that maps inputs to outputs. For example, network traffic data is labeled as either normal or abnormal. During the training process, the ML algorithm attempts to identify the characteristics that distinguish normal network traffic from abnormal network traffic by comparing its predictions to the actual labeled data. The labels are crucial for fine-tuning the model during training and to verify the results. In **Unsupervised Leaning**, on the other hand, the training data is not labeled and the system learns to find characteristics of the data autonomously. A well-known example of an Unsupervised Learning approach are Auto-Encoders. In this approach, the algorithm compresses the data to a latent state, where only the relevant information remains. Next, the ML algorithm attempts to reconstruct the original data from this latent state as accurately as possible. To fine-tune, the model gets adjusted based on a particular loss function, that compares the original data to the reconstructed data. Section 2.6.2 provides more detailed information about the inner workings of AEs [34].

2.5.2. Deep Learning (DL) and Neural Networks (NN)

The foundational principle of (artificial) neural networks is inspired by biological neural networks, such as those in the human brain, where multiple neurons are interconnected to process information. NN are mostly trained using DL methods, a subset of machine learning techniques. These methods involve adjusting the network's parameters (weights) to minimize the error in predictions. This adjustment process is called **backpropagation**. Essentially, DL strengthens or weakens complex algebraic connections based on the inputs and desired outputs. Those connections build a computational path between multiple nodes, usually organized into many layers through which the data passes. NNs are capable

of learning complex patterns in large amounts of data. Traditionally, each node (also known as a unit) calculates the weighted sum of inputs from its predecessor nodes and then applies a (nonlinear) activation function to produce an output. Nonlinear functions provide a sufficiently large network with the advantage of representing arbitrary functions. During training, the data passes through multiple layers, with each layer transforming the representation produced by the previous layer through several computations before reaching the output. The layers between the input and output are usually referred to as **hidden layers**, or a black-box. However, a significant challenge in training NN is **overfitting**, which refers to the model's inability to generalize well to new data. This occurs when the model learns the data patterns to exactly, resulting in a model that performs well on training data but poorly on unseen test data.

The detailed processing of the data at each layer during training is determined through the model's implementation. Even though many different approaches of NN implementations exist, most of which are tailored to specific use cases, understanding Feedforward Neural Networks (FFNNs) and Recurrent Neural Networks (RNNs) is essential for the following chapters [34]. FFNN are characterized by their unidirectional connections. They form a directed acyclic graph with designated input and output nodes, allowing information to flow through the network without loops. Each node within the network computes its input and passes the results to its successor nodes. In contrast, RNNs have connections between nodes that form a directed cycle, allowing information to be fed back into the network. This structure enables RNNs to maintain an internal state (memory), therefore inputs received at an earlier time step affect the output from the current input. This makes RNNs particularly useful for tasks involving sequential data. A major concern with RNNs is the *vanishing/exploding gradient* problem. This refers to an issue that arises during the training of the model, where the gradients can become very small (vanishing) or very large (exploding) during backpropagation. This can cause the network to either learn very slowly or become unstable and learn chaotically. Gradients are essentially the adjustments made to the network's weights to minimize errors and improve learning. To address this problem and preserve information over many time steps, special RNN architectures have been developed over the years, one of which is the LSTM architecture.

2.6. LSTM and AE

The hybridization of LSTM and AE for network intrusion detection is a concept introduced by Hnamte et al. [16]. This approach combines the advantages of processing sequential

data, while still utilizing the benefits of AE for anomaly detection. To provide a clearer understanding of the following chapters, each DL architecture type is described separately and in more detail in the following Subsections.

2.6.1. Long Short-Term Memory (LSTM)

Long Short-Term Memory is a type of RNN that is well-suited for sequence prediction by avoiding the vanishing/exploding gradient problem. An LSTM-based architecture contains several cells (or units) queued next to each other with each cell containing a forget gate, an input gate, and an output gate. Furthermore, a cell state (long-term memory) and a cell output (short-term memory) from the previous time step are incorporated within the LSTM-Cell. A single LSTM-Cell is depicted in Figure 2.12.

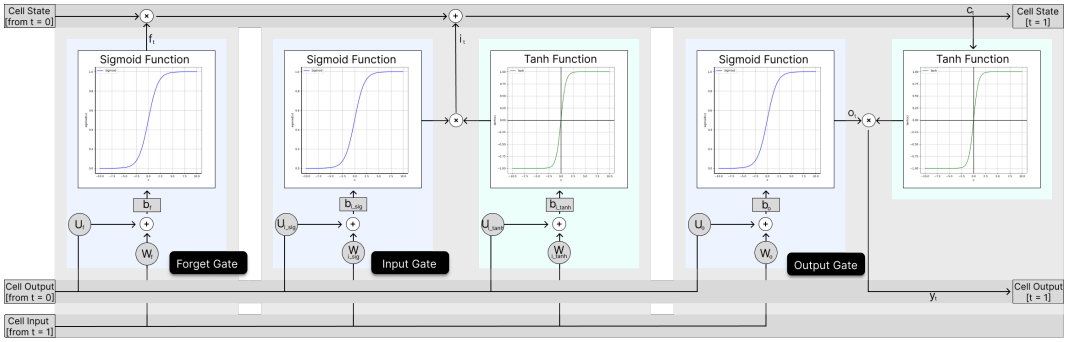


Figure 2.12.: Single LSTM-Cell (Unit) transforming input data, previous cell output, and cell state through a sequence of gates (forget, input, output), resulting in a cell output that captures and maintains temporal dependencies.

In this representation of LSTM, all units of one kind are pooled into a vector. Therefore, f_t is the vector of the forget gate activations, i_t is the vector of the input gate activations, c_t is the vector of the cell state, o_t is the vector of the output gate activations, and y_t is the vector of the cell output activation. Usually, the input vector x_t of an input sequence at time **t** is fed together with the cell state c_{t-1} and the cell output y_{t-1} from the previous sequence to an LSTM-Cell.

The top of the figure contains the **cell state** (Equation 2.3), also referred to as the long-term memory. This is the path through which gates can add or remove information with some minor linear operations. The **forget gate** (Equation 2.1) is the first step in the LSTM cell which influences which and how much information from the concatenated

$x - t$ and y_{t-1} gets passed through to the cell state. After activation, the output (f_t) will be multiplied with the cell state (c_{t-1}). The **input gate** (Equation 2.2) is composed of two sub-operations and will decide which information is going to be stored in the cell state. First, the sigmoid operation (blue) is responsible for which values will be updated, and the tanh operation (green) creates a vector of new values that could be added to the state. Secondly, the cell state will be fully updated to c_t, by adding the pointwise product (i_t) of the previous two operations to the state. Finally, the **output gate** (Equation 2.4) decides what the new output y_t (Equation 2.5) will be. This is done by passing the newly updated cell state through a tanh operation and multiplying it by the sigmoid gate (o_t) output. The matrices W_f,$W_{i_{sig.}}$,$W_{i_{tanh}}$,W_o correspond to the weights of the connections between the cell input and the forget gate, input gate (sigmoid and tanh activation), and output gate, respectively. The matrices U_f,$U_{i_{sig.}}$,$U_{i_{tanh}}$,U_o correspond to the weights of the connections between the cell output with one-step delay (t-1) and the forget gate, input gate (sigmoid and tanh activation), and output gate, respectively. The vectors b_f,$b_{i_{sig.}}$,$b_{i_{tanh}}$,b_o are the bias vectors of the forget gate, input gate (sigmoid and tanh activation), and output gate, respectively. Typically, LSTM utilizes sigmoid [$\sigma(x) = \frac{1}{1+e^{-x}}$] and tanh [$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$] activation functions. $\odot$represents the pointwise multiplication of two vectors. The LSTM memory cell forward pass rules are defined as follows:

$$\text{forget gate:} \quad f_t = \sigma(W_f x_t + U_f y_{t-1} + b_f) \tag{2.1}$$

$$\text{input gate:} \quad i_t = \sigma(W_{i_{sig.}} x_t + U_{i_{sig.}} y_{t-1} + b_{i_{sig.}}) \odot \tanh(W_{i_{tanh}} x_t + U_{i_{tanh}} y_{t-1} + b_{i_{tanh}}) \tag{2.2}$$

$$\text{cell state:} \quad c_t = c_{t-1} \odot f_t + i_t \tag{2.3}$$

$$\text{output gate:} \quad o_t = \sigma(W_o x_t + U_o y_{t-1} + b_o) \tag{2.4}$$

$$\text{cell output:} \quad y_t = o_t \odot \tanh(c_t) \tag{2.5}$$

2.6.2. Auto-Encoder (AE)

An Auto-Encoder is a special type of FFNN and unsupervised learning algorithm. AEs designed to efficiently compress (encode) input data to a lower-dimensional representation and to form a bottleneck (latent space). The encoder can be compared to regular Principal Component Analysis (PCA). The main distinction between the encoder and PCA is that the encoder can learn non-linear dependencies using non-linear activation functions. Then

the decoder tries to reconstruct the original input from this compressed representation. For an AE to work effectively, the data should have dependencies across dimensions. If all the dimensions are dependent, the AE cannot learn a lower dimensional representation. The main challenge for building an AE is balancing input data sensitivity. It needs to be sensitive enough to reconstruct the original data successfully, but not too sensitive so it does not overfit. The general architecture of an AE is depicted in Figure 2.13.

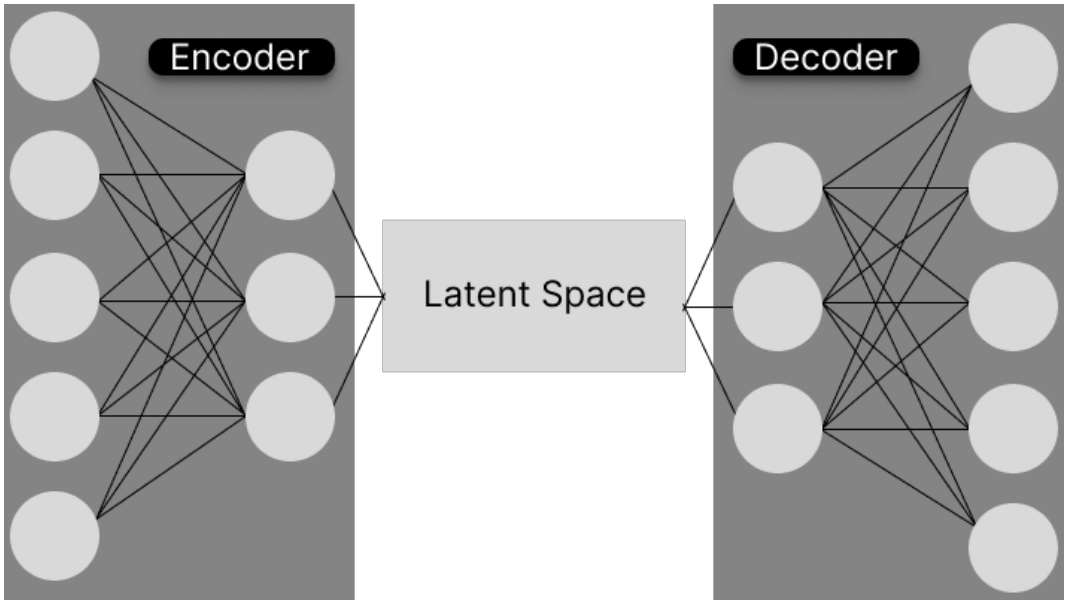


Figure 2.13.: Basic Auto-Encoder Architecture, compressing input data through an encoder to a latent space, followed by the reconstruction through a decoder.

Essentially, AE tries to learn a function $h_{w,b}(X) \approx X$, where X represents the original data signals $(x_1, x_2, ..., x_n)$. This implies that AE tries to learn an approximation of the real data to minimize the difference between the original input and its reconstruction. The loss function (reconstruction error) of an AE can be measured by a multitude of different functions. Two examples would be the Mean Absolute Error (MAE) (Equation 2.6) or the Mean Squared Error (MSE) (Equation 2.7) function.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|\hat{x}_i - x_i| \qquad (2.6)$$

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{x}_i - x_i)^2 \qquad (2.7)$$

2.7. Explainable Artificial Intelligence (XAI)

The fear of losing control in the context of AI is often discussed in scenarios ranging from highly fictional, such as robots taking over the world, to more realistic ones, like the unpredictable behavior of autonomous vehicles or the risk of them being hacked. In safety-critical environments, the consequences of losing control can be catastrophic, potentially leading to severe injuries or even the loss of life. As the trend towards SDVs continues, security protocols are becoming increasingly complex, sometimes relying on AI themself to handle vast amounts of data and mitigate highly sophisticated attack scenarios. In the context of network intrusion detection, it is important to understand why certain network activities are classified as dangerous and vice versa. This can consolidate trust in these systems and enhance their reliability by providing the possibility for humans to review and understand incidents within the network.

In general terms XAI is a set of methods and tools that can be adopted to make ML models understandable to humans by providing insights into the returned results [23]. It is important to note that interpretability and explainability are mostly used synonymously in the literature, potential differences are not considered in this thesis. Linardatos et al. [24] highlighted several dimensions through which explanation methods can be categorized (Figure 2.14).

Firstly, the scope of explainable models can be distinguished by local and global interpretability. Local interpretability methods focus on explaining a single prediction made by the model. In contrast, global interpretability methods aim to explain the overall behavior of the model. The author also highlights that interpretability methods can be applied to various data types, including tabular data (e.g. spreadsheets and databases), text data (e.g. documents), image data (e.g. photographs), and graph data represented in the form of nodes and edges. In terms of the purposes of interpretability, multiple distinctions can be made. Some methods are designed to create inherently interpretable models that are straightforward and easily understandable by humans. Other methods aim to explain black-box or complex models after they have been trained. These ”post-hoc” methods provide explanations for predictions made by models that are typically difficult to explain. Additionally, some interpretability techniques are focused on enhancing the fairness of a model, ensuring that its decisions are unbiased. Lastly, there are methods aimed at testing the sensitivity of predictions, and assessing how changes in input data can affect the model’s output. Another important dimension in the taxonomy is whether the interpretability methods are model-specific or model-agnostic. Model-specific methods are tailored for specific DL architectures or groups of models. In contrast, model-agnostic

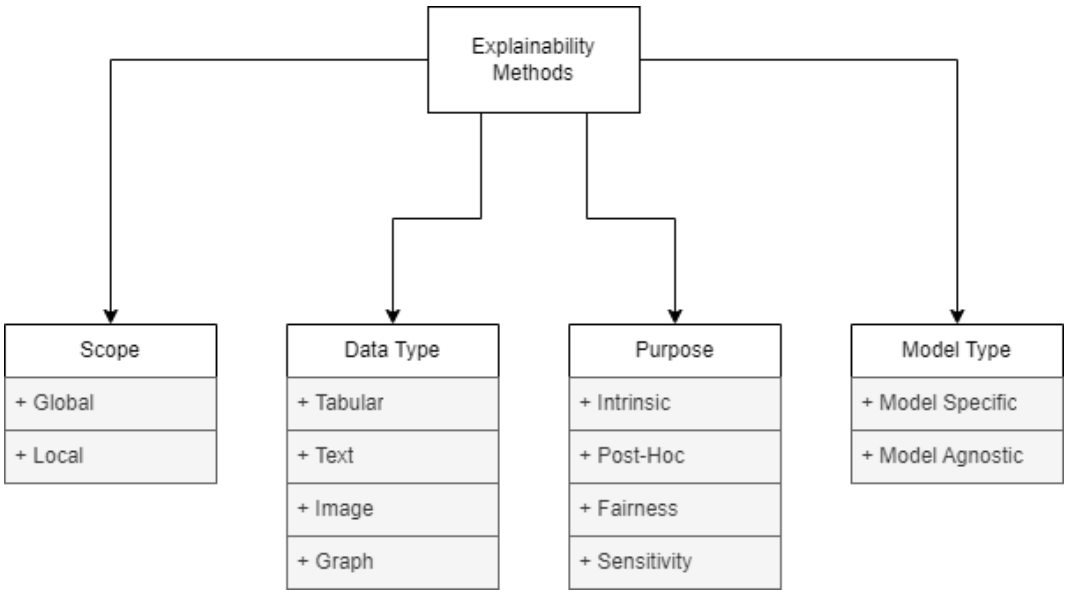


Figure 2.14.: Explainability methods based on [24], categorized by scope, data type, purpose, and model type, illustrating the various dimensions and approaches for interpreting machine learning models.

methods can be applied to any model, offering flexibility and broad applicability across various use cases.

3. Related Work

This chapter provides a comprehensive review of studies focused on the vulnerabilities of SOME/IP and the exploration of numerous Intrusion Detection approaches tailored to this protocol. Furthermore, works on LSTM-AE within the context of IDS are highlighted. At last various approaches regarding the explainability of LSTM and AE models were inspected.

3.1. SOME/IP Vulnerabilities and Intrusion Detection

Herold et al. [15] employed Complex Event Processing (CEP) to detect anomalies in SOME/IP network traffic. They tested against various attacks, including **Wrong Interface**, **Spoofed Client-ID**, **Error on Error/Event**, **Request without Response**, **Response without Request**, and **Disturbed Timing** (e.g. Denial of Service (DoS) attack by flooding a service with requests). For detection, CEP and the Event Processing Language were used to develop rules that identify anomalies. However, the performance evaluation indicates that the IDS is unsuitable for online use, as activating multiple rules significantly increased processing time. Additionally, SOME/IP-SD was not considered in this work.

Zelle et al. [36] introduced and evaluated three MITM attacks on the SOME/IP protocol through a formal analysis approach: the Copycat Attack, the De-association Attack on the Service Offer, and an Attack on Publish/Subscribe. Furthermore, two security extensions were presented to mitigate those attacks. The solutions aim to enable authentication and authorization of the provision and use of SOME/IP services. To realize this, the first approach utilizes the use of certificates and digital signatures to establish symmetric keys. The second approach requires an additional authorization server to enable only efficient symmetric cryptography. The results show the feasibility of the approaches. However, the second approach might be more reasonable to use in Automotive communication, due to the typically scarce resources of ECUs.

Alkhatib et al. [1] proposed an IDS utilizing a RNN approach for the SOME/IP Protocol. Similar to the work by Herold et al., the attack types Error on Error/Event, Request without Response, and Response without Request were addressed. However, Alkhatib et al. generated a publicly available synthetic dataset and identified important SOME/IP features for model training. To evaluate the IDS performance, Area Under the Curve (AUC) values, Receiver Operating Characteristics (ROC) curves, and F1 scores were calculated for each attack class and normal traffic. Although this IDS demonstrated very good results for the tested attacks, the results were not verifiable.

Gail et al. [9] explored Decision Tree-Based Rule Derivation on the SOME/IP protocol, to enhance the explainability and comprehensiveness of ML techniques. Their work aimed to address the challenge of verifying that anomalies detected in network traffic indeed originate from an attack. The main objectives of this work included the detection of attacks, the automated generation of rules, and the provision of easily verifiable results for human operators. The attacks tested are the Wrong Interface attack, Error on Error/Event attack, Request without Response attack, and Response without Request attack. The evaluation demonstrated that single-message attacks, such as the Wrong Interface attack, were easily identified, and corresponding rules could be generated. However, attacks involving a sequence of packets, such as Requests without Response, could not be detected.

In a subsequent paper, Gail et al. [10] tested an alternative ML technique for rule generation on the SOME/IP protocol. This study implemented Genetic Programming with a Human-in-the-Middle concept, which offered the benefits of model transparency and understandability. The generated rules were verifiable, and decisions could be logged and traced, allowing for the extraction of patterns that matched known attacks. In contrast to the previous work by Gail et al., the focus were sequential or in parallel coordinated attacks such as the De-association attack [36], Distributed Denial of Service (DDoS) attack, and Error-on-Event attack. The single message attack Wrong Interface attack was tested as well. To generate and document rules, a new Domain Specific Language called RuleEth was developed. The fitness function F1-Score was utilized. Rules generated can be compiled to C++ code and, therefore be used for resource-efficient, real-time attack detection. The results show that real SOME/IP traffic, enriched with multiple different attacks, could be flagged as such.

Luo et al. [26] introduced an IDS for the SOME/IP protocol combining rule-based and AI-based detection approaches. The AI detection component employs a multi-gated recurrent alongside a Bayesian optimization process to detect anomalies in SOME/IP headers, SOME/IP-SD messages, message intervals, and payloads. To generate meaningful SOME/IP data, the authors utilized a toolchain of Prescan, Simulink, and Vector CANoe,

producing relevant IVN data such as camera data, ADAS data, body data, and attack data. Precision, recall, and F1-score were used to evaluate the model accuracy.

3.2. LSTM and AE for Intrusion Detection

Mushtaq et al. [31] proposed a hybrid framework integrating AE for optimal feature selection and LSTM for classification tasks. The AE was used to reduce the dimensionality of data, retaining only the most relevant features and eliminating noise. The next step was to employ an LSTM to classify the data into normal and anomaly categories based on the features extracted by the AE. The AE-LSTM model was trained and evaluated on the NSL-KDD dataset, a widely used benchmark in IDS research. The study highlights that the reduced feature subset obtained from AE improves the model's performance and efficiency, by significantly reducing the testing and modeling time. The proposed model achieves a moderate classification accuracy, detection rate, and false alarm rate.

Similarly, Mahmoud et al. [28] applied the AE-LSTM approach to the NSL-KDD dataset. In their study, an AE with three encoder layers and three decoder layers was used. The output features were then fed into the LSTM model for Intrusion Detection. The model was trained and tested using both a 5-class classification (DoS, Probe, R2L, U2R, Normal) and a binary classification (Malicious, Normal). The results demonstrate promising F1-Scores for the 5-class classification and the binary classification.

Hnamte et al. [16] introduced a two-stage DL-based IDS by hybridizing an LSTM and an AE The primary objective was to forecast both short-term and long-term network attacks. The model was trained and evaluated using the CICIDS2017 and CSE-CICDIS2018 datasets, which are recognized benchmarks in intrusion detection research and are considered more suitable for real-time detection compared to the NSL-KDD dataset. In the first stage of this work, the data was preprocessed by refining and normalizing it. In the second stage, the cleaned data was used to train and evaluate the LSTM-AE. The Encoder and Decoder layers were constructed using LSTM, thus achieving the hybridization. The performance was assessed using two error functions MSE and MAE. The results indicated that MAE outperformed MSE in terms of accuracy performance.

4. Data Discussion

Before training a ML model, it is essential to gather and properly understand the data that will be used. The quality and characteristics of the dataset play a critical role in determining the performance and reliability of the model. In this chapter, I will discuss the dataset utilized for both training and evaluating the model. Additionally, I will outline the preprocessing and sequentialization steps applied to the data.

4.1. Data Analysis

The scarcity of publicly available SOME/IP data, as highlighted in various publications [1, 26], poses a significant challenge. The literature provides a SOME/IP data generator that produces only SOME/IP header data with randomly filled payloads. To address this limitation, Luo et al. [26] employed a pipeline of various tools to generate datasets containing realistic SOME/IP packets along with their corresponding payloads. The datasets generated by this method were used in this thesis.

4.1.1. Simulation Analysis

To generate the data, Luo et al. [26] created a simulation to mimic a vehicle equipped with various sensors, driving in four different traffic situations. All scenarios (Table 4.1) involve a main vehicle following another vehicle. The data generated reflects the vehicle's SOME/IP data and communication patterns tailored to these scenarios.

The first scenario describes the leading vehicle driving in a straight line and at a constant speed. The second scenario depicts the leading vehicle driving in a straight line with segments of acceleration. The third scenario simulates the vehicles on a congested road with alternating starting and stopping. The fourth scenario simulates the vehicles driving

Scenario Number	Short Scenario Description
1	Straight line and constant speed.
2	Straight line and segments of acceleration.
3	Congested road with alternating starting and stopping.
4	Both straight and curved segments, varying speeds, and accelerations.

Table 4.1.: All simulation scenarios with their corresponding short descriptions. Based on [26].

on a road with both straight and curved segments, varying speeds, and accelerations. Two main datasets were derived from the simulation. The first dataset was created to focus on rule-based detection, and the second was designed for AI-based detection. In this thesis, I will focus on the dataset created for AI detection. A total of 12 Comma-separated value (CSV) files, together with the data description sheet, are provided by the author on GitHub [27]. For each scenario, three CSV files were created: one containing the original data from the simulation, one infused with replay attack data, and another with tampered attack data. The attacks used, are further discussed in Subsection 4.1.3.

4.1.2. In-Vehicle Network Analysis

To generate data from the traffic scenarios, Luo et al. [26] deployed several SOME/IP service nodes and integrated them into the simulation. Each service node represents a domain within the IVN and includes a service with various methods to send data, as well as a client to receive data. The implementation of each service, method, and client follows the SOME/IP standard, with the input and output parameters of the nodes managed through the service subscription and publication mechanism of the SOME/IP protocol. Figure 4.1 illustrates the relationships between the different service nodes, including (High Performance Computing (HPC), ADAS, Graphical User Interface (GUI), Body&Dynamics). Since all clients in the generated data share the same ID, they are omitted from the figure.

The **HPC** node represents the central computing unit and runs the Adaptive Cruise Control (ACC) service (ID 0x1472), calculating throttle opening, brake fluid pressure as well as the relative speed and headway (collision) time of the preceding vehicle. This is achieved by processing sensor data from the ADAS node and kinetic vehicle information from

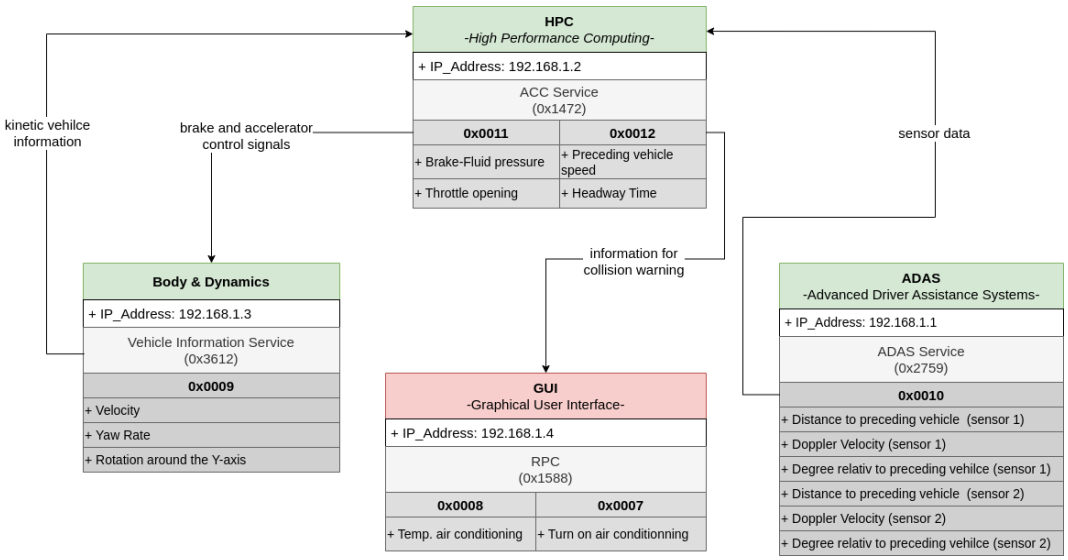


Figure 4.1.: The four SOME/IP service nodes (HPC, ADAS, GUI, Body&Dynamics) representing the vehicles IVN and the SOME/IP data generated within each method. Each node is assigned a unique IP address, and the bold hexadecimal numbers indicate the *method_id* of the services. These nodes generate payloads in the form of up to six signals based on the method used. Based on [26].

the Body&Dynamics node. The **ADAS** node containing an equally named service with the (ID 0x2759), processes sensor data, such as the distance to the preceding vehicle, Doppler speed, and angle difference, which are extracted from two separate cameras. The processed data is sent then to the HPC node. The **Body & Dynamics** node processes the vehicle's kinetic information, such as vehicle speed, heading angle, and longitudinal offset, and sends this information to the HPC node, by utilizing its vehicle-information-service (ID 0x3612). Notably, the **GUI** node (marked in red) is intended to simulate RPC to control the air conditioning and was only integrated into the datasets for rule-based detection and is therefore not considered in this thesis. However, data packets within the AI dataset may have the GUI node as an endpoint, which is necessary to simulate signals for the driver, such as collision warnings or the relative speed of the preceding vehicle. Each service included in the AI-detection dataset sends messages continuously every 10 milliseconds (ms), except for the ADAS service, which sends its data every 40 ms in defined cycles. This creates a continuous, time-dependent communication flow, with a full communication cycle consisting of 13 packets. Table 4.2 illustrates the relationship

between the communication cycle, *service_id*, and *method_id*.

Node	service_id	method_id	cycle	Packets in a full communication cycle
HPC	0x1472	0x11	10 ms	4
		0x12	10ms	4
B&D	0x3612	0x09	10 ms	4
ADAS	0x2759	0x10	40 ms	1

Table 4.2.: Data communication cycles in the context of their respective nodes, service_ids, and method_ids. Based on [26].

In total, the AI-detection dataset contains *2,480,172* packets. This is divided into *790,503* packets for scenario1, *1,031,034* packets for scenario2, *443,637* packets for scenario3, and *214,998* packets for scenario4. Each scenario's packets are further subdivided into three equally sized sub-datasets: one normal, one replay-attack-infused, and one tampered-attack-infused dataset.

4.1.3. Attack Analysis

The security characteristics of an AE based IVN is by design rather robust due to its fixed topology and point-to-point communication method. This predetermined structure makes it challenging for attackers to gain access and introduce rogue or malicious devices into the network. Although not impossible as research by Zelle et al. [36] has shown, were an MITM was utilized to impersonate service or client and thereby enable the attacker to redirect or alter packets. Another feasible approach outlined by Luo et al. [26] is the tampering of the vehicle sensors themselves and thereby infiltrating the network straight from its data source. To simulate a scenario where the network has already been compromised, Luo et al. create the two attack-infused datasets, replay and tamper. The attacks were designed to solely focus on the payload, thus the header and communication cycle of the attack data meet the system requirements and interlink smoothly with the normal data. A replay attack describes the concept of capturing and re-feeding (replaying) normal data back to the network. A tamper attack on the other alters data packets to change their meaning or function. The **replay-attack-infused** dataset, is infused with

multiple sequences each containing 26 consecutive attack packets. The **tamper-attack-infused** dataset on the other hand has attack sequences in different sizes ranging from one attack packet to 10 consecutive attack packets. Figure 4.2 illustrates the discrepancies between the Break-Fluid Pressure data of the normal, replay, and tampered data in scenario3.

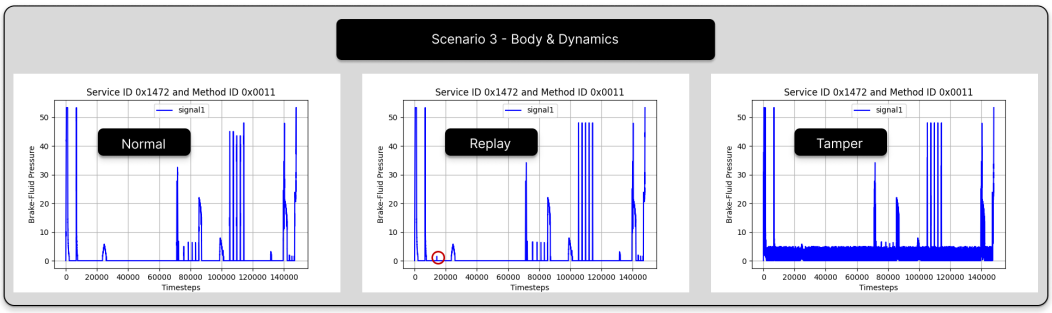


Figure 4.2.: Normal, replay-attack-infused, and tampered-attack-infused data from scenario3. Signal1 with service_id 0x0147 and method_id 0x0011 (Break-Fluid Pressure). The red circle marks one example discrepancy to the normal data.

The complete history of attack-infused data in the context of their *service_id* and *method_id* are illustrated in Figures 2 to 9 of the appendix.

4.1.4. Packet Analysis

Luo et al. [26] employed a layer-by-layer unpacking technique, essential for Ethernet communication, to construct the dataset. Each initial SOME/IP packet traverses through the layers of the OSI model, with the corresponding header for each layer being added to build the packet (feature) structure: *time, mac_dst, mac_src, ipv4_protocol, srcport, dstport, service_id, method_id, someip_length, client_id, session_id, protocol_version, interface_version, message_type, return_code, signal1, signal2, signal3, signal4, signal5, signal6, label*. The features from *service_id* to *return_code* represent the SOME/IP header, while the six signals represent the SOME/IP payload, as discussed in Subsection 2.3.1 and illustrated in Figure 2.3. The *mac_src* and *mac_dst* features represent the MAC source or destination of the packet. The *srcport* and *dstport* features indicate the source or destination port numbers, respectively. The *ipv4_protocol* feature represents Internet Protocol version 4, used in the network layer of the OSI model. For the following chapters, I will refer to the header as encompassing all features from *mac_dst* to *return_code*, and the payload

as the features ranging from *signal1* to *signal6*. The *time* and *label* features are excluded from these definitions. The *label* feature feature represents the ground truth or packet classification as either normal (1), replayed (2), or tampered (0). The *time* feature is not considered because it is detached from the unpacking process.

4.2. Data Preprocessing

Effective data preprocessing is crucial for preparing the dataset to effectively train and evaluate the LSTM-AE model. This process involves transforming raw data into a more refined version that meets the model's requirements. The following subsections outline the approach and methods I used to transform the data provided by Luo et al. on GitHub [27]. I concatenated the datasets from scenario1 to scenario4 into a single dataset for each data type (normal, replay, tamper), respectively. For future reference, these concatenated datasets will be referred to as **scenario5**.

4.2.1. Feature Relevance Analysis

Ordinarily, feature relevance analysis is one of the first steps in data preprocessing. It ensures the quality of the data and that it meets the model's prerequisites. In this case, however, a synthetic, well-documented dataset [27] is used, and since the attacks within the dataset were specifically designed to focus solely on the payload, a feature analysis is less critical. Nonetheless, for the sake of thoroughness, a correlation matrix was created using the normal data from scenario5. Figure 10 in the appendix illustrates the correlations between the different features. Please note that, due to its size, the correlation matrix is provided primarily to convey the general idea. The detailed relationships between the features are recorded in Figure 4.3, which visualizes the dependencies between the MAC features, port features, and method/service IDs.

Notably, constant features, those with values that remain unchanged across all samples in the dataset, were omitted from the analysis. Additionally, categorical data was one-hot encoded before creating the correlation matrix, a technique used to convert categorical data into binary features. The constant features excluded from the analysis are: *ipv4_protocol, someip_length, client_id, protocol_version, interface_version, message_type,* and *return_code*.

The results of the correlation matrix, along with the information provided in the paper, support the assumption that only features focusing on the payload, specifically *service_id,*

```
['dstport_0x9C45', 'mac_dst_0x0284CF3BBE01']
['service_id_0x1472', 'mac_src_0x0284CF3BBE01']
['method_id_0x0012','dstport_0x9C4D','mac_dst_0x0284CF3BBE03','srcport_0x9C4C']
['method_id_0x0011','dstport_0x9C47','mac_dst_0x0284CF3BBE02','srcport_0x9C45']
['method_id_0x0010','service_id_0x2759','mac_src_0x0284CF3BBE00','srcport_0x9C44']
['method_id_0x0009','service_id_0x3612','mac_src_0x0284CF3BBE02','srcport_0x9C47']
```

Figure 4.3.: Fully correlated features extracted from the correlation matrix (10) and encaptured between square brackets. [26]

method_id, signal1, signal2, signal3, signal4, signal5, signal6, and *label* should be used for the remaining process of data normalization and sequentialization.

4.2.2. Data Normalization

Data normalization is a crucial step in the data preprocessing phase, as it can significantly impact the model's performance and training efficiency. Data normalization describes the process of scaling the data to a uniform range, ensuring that each feature contributes equally to the model's learning process and preventing any bias [12]. The Min-Max normalization method (Equation 4.1) was utilized, scaling the features of the dataset to a common scale between 0 and 1.

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{4.1}$$

This scaling technique sets the minimum value (X_{min}) within a feature set to zero and the maximum value (X_{max}) to one. The remaining values are scaled accordingly between zero and one. X' represents the scaled value, and X is the value to be scaled.

The payload features (signals) and the header features were normalized separately, as the content and scale of the payload features within a packet can vary significantly depending on the node that generated them. The normalization process for the header features involved merging the *service_id* and *method_id* features into a single feature called *message_id*, reducing the number of features generated by one-hot encoding. The resulting four header features are: *message_id_3612_09, message_id_2759_10, message_id_1472_11,* and *message_id_1572_12*. The normalization process for the payload features was as follows: Each payload signal was normalized within the context of its corresponding *message_id*. This separation of signals is necessary because the quantity and scale of signals vary depending on their associated *message_id*. For example, packets with the

message_id_1472_11 (ACC service) contain two signals in their payload, one for brake fluid pressure and another for throttle opening. In contrast, packets with the *message_id* 0x0759_0010 (ADAS service) contain six differently scaled signals representing sensor data. The general data normalization process for all CSV files is illustrated in Figure 4.4.

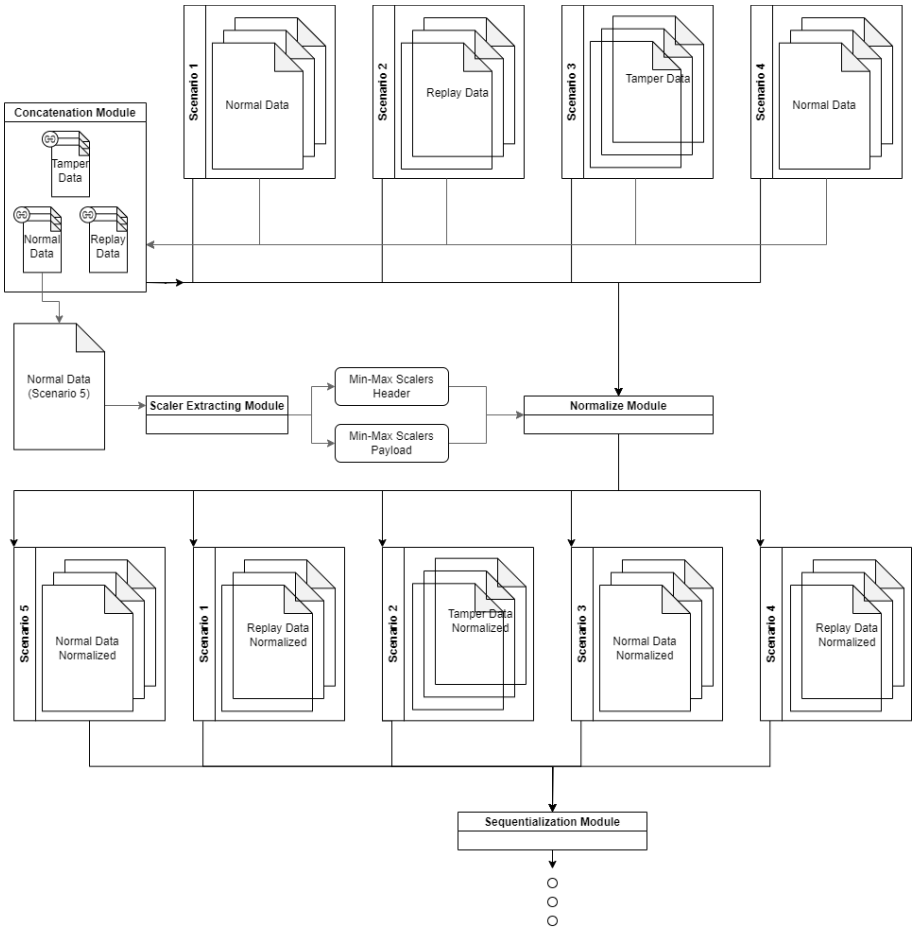


Figure 4.4.: Normalization pipeline of all AI-detection datasets provided by Luo et al. [26]

The CSV files containing only benign data were concatenated into a new file and normalized to extract the scalers for the payload and header features. These scalers were then used to normalize all 12 CSV files separately, including the previously concatenated benign data files. This approach ensures that all scenarios can be analyzed and evaluated separately while maintaining uniform scaling across the dataset.

4.2.3. Data Sequentialization

After normalization, the data needed to be sequentialized, a crucial step before feeding it into the model. LSTM-based models are specifically designed to handle and learn from sequential data, where the order of data points is important. This allows the model to capture time-relevant dependencies during the training phase [12]. As a baseline, the initial tests utilized a sequence length of 91 and a sliding step of 30, based on the research by Luo et al. [26]. The sequence length represents the number of packets (timesteps) within a single sequence fed to the model, while the sliding step refers to the number of packets skipped before creating a new sequence. Adjusting the sequence length and sliding step can optimize the training process. A larger sequence length allows the model to consider more historical information at each step. In contrast, the sliding step can be used to artificially increase the amount of training data by overlapping sequences, rather than connecting them consecutively. Another important aspect of data sequentialization is the separation of the data into training and testing sequences. However, since an LSTM-AE model utilizes unsupervised learning and is therefore designed to learn the network's normal behavior only the benign data needed to be separated into training and validation sequences. A 90-10 data split (90% training and 10% validation) was used instead of the typical 80-20 split, to increase the amount of benign data available for training. Sequentialization was performed on the normalized data from each simulation scenario separately, ensuring that the validation sequences included every traffic situation. To streamline the training process, all training and validation sequences were combined into two comprehensive sets. The final training set contains *24,791* sequences, and the validation set *906* sequences. The remaining replay-attack-infused and tampered-attack-infused datasets were used solely for testing and evaluation purposes and were not divided. The shape of every sequence is illustrated in the Table 4.3.

Notably, to create a natural testing data flow, the sliding step was adjusted to equal the sequence length during the sequentialization of the validation and attack-infused data.

Scenario	Data Type	Number of Sequences	Number of Packets per Sequence	Number of Features per Packet
1	Training Validation Replay & Tamper	7902 289 2895	91	10
2	Training Validation Replay & Tamper	10308 377 3776	91	10
3	Training Validation Replay & Tamper	4434 162 1625	91	10
4	Training Validation Replay & Tamper	2147 78 1625	91	10
5	Training Validation Replay & Tamper	24791 906 9084	91	10

Table 4.3.: Sequence shapes of all scenarios, given a sequence length of 91 and a sliding step of 30 for the training data. Attack and validation data utilize a sequence length and sliding step of 91.

5. System Design

This work is inspired by the innovative approach of Hnamte et al. [16], who tested a Two-Stage DL Model for Network Intrusion Detection using LSTM-AE. The primary distinction of this thesis lies in its application to the SOME/IP protocol, which is mostly used in safety-critical environments. Consequently, the explainability of the model is a focal point, given the necessity for transparent and interpretable decision-making in such contexts.

5.1. Technical Setup

The model was built and evaluated in JupyterLab using Python 3.8.10 with TensorFlow/Keras 2.13.1 development libraries. The development platform is a Linux server system, incorporating an Intel(R) Xeon(R) Gold 6230R CPU @ 2.10GHz, 256 GB of memory, and an NVIDIA RTX A6000 with 48 GB of GDDR6 memory. The experimental platform is a laptop with an Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz and 16 GB of memory.

5.2. LSTM-AE Architecture

Understanding the model architecture plays a crucial role before the implementation process. The mock-up design for the encoder is presented in Figure 5.1. This visualization illustrates how the data flows through the encoder, forming the bottleneck and creating the latent space. A more detailed version of the complete data flow through the LSTM-AE network is visualized and explained in Section G of the appendix.

A sequence of length *Sequence Length (s_l)* containing several timesteps (in this case SOME/IP packets) is fed into the first layer of the network. The outer layer length/size

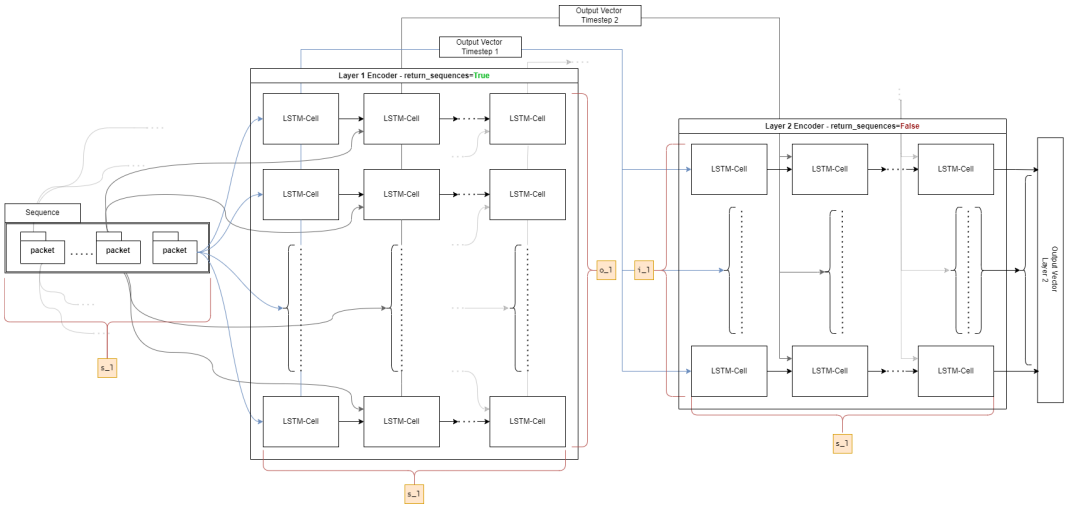


Figure 5.1.: LSTM-AE data flow of the encoder. S_l indicates the *sequence length*, U_i and U_o

(o_l) represents the number of LSTM cells incorporated in the first layer of the encoder. The inner layer length/size (i_l) represents the number of LSTM cells in the second layer of the encoder. The **layer length** refers to the number of neurons (or in this case, LSTM cells) in each individual layer, used to process the input data. After the encoding process, the latent space, depicted here as *Output Vector Layer 2*, is fed into a Repeat Vector (Appendix 14), where it is duplicated, forming an input sequence of the same length as the original. This repeated sequence is then fed into the decoder (Appendix 13). The decoder processes this data through two decoding layers and, together with an additional *TimeDistrubuted* layer (Appendix 16), attempts to reconstruct the original input sequence. The o_l and i_l in the decoder are inversely proportional to that in the encoder. The number of layers for both the encoder and decoder is inspired by the work reviewed in previous studies [16, 28, 31].

5.3. LSTM-AE Implementation

The implemented architecture of the encoder, decoder, and model compilation are depicted in Listings 5.1 to 5.3, respectively. Listing 5.1 shows the encoder implementation of the LSTM-AE model. First, the input shape of the data is specified by the number of *timesteps*

(in this case, packets) per sequence and the number of features per timestep. Next, the first encoding layer is defined, where the size of the layer, the *activation function*, and the *recurrent_activation* function are specified. The o_l (*outer_size*) and i_l (*inner_size*) hyperparameters are determined during the fine-tuning process detailed in Section 5.5 of this thesis. The activation functions are set to tanh and sigmoid, as these are commonly used in the context of LSTM cells [16, 28, 31] and are therefore employed in this thesis as well. After the final encoding layer, the *RepeatVector* duplicates the output of the encoder and passes it to the decoder in Listing 5.2. For clarity in the code, the *RepeatVector* layer is included with the encoder, although technically, it is not part of the encoding process.

```
1 inputs = Input(shape=(timesteps, n_features))
2 encoded = LSTM(outer_size, activation='tanh', recurrent_activation='sigmoid',
      return_sequences=True)(inputs)
3 encoded = LSTM(inner_size, activation='tanh', recurrent_activation='sigmoid')
     (encoded)
4 encoded = RepeatVector(timesteps)(encoded)
```

Listing 5.1: Encoder of LSTM-AE, stacked with RepeatVector layer.

The implementation of the decoder (Listing 5.2) uses the same activation functions as the encoder and reverses the number of LSTM cells per timestep, going from *inner_size* to *outer_size*. Following this, the *TimeDistributed* layer compresses the decoded data to its original dimensions, reconstructing the sequence.

```
5 decoded = LSTM(inner_size, activation='tanh', recurrent_activation='sigmoid',
      return_sequences=True)(encoded)
6 decoded = LSTM(outer_size, activation='tanh', recurrent_activation='sigmoid',
      return_sequences=True)(decoded)
7 decoded = TimeDistributed(Dense(n_features))(decoded)
```

Listing 5.2: Decoder of LSTM-AE, stacked with TimeDistributed layer.

Listing 5.3 shows the final step in implementing the model's architecture: compiling the model. In this step, the loss function (*loss_fuc*) and the optimizer function, which in this case is Adaptive Moment Estimation (Adam), are selected. Along with the Adam optimizer its control parameters *beta1*, *beta2*, and *learning rate* need to be set. The **loss function** serves as the model's performance measure by quantifying the difference between the predicted values and target values. The **optimizer** is the algorithm used to minimize the loss function, while the **learning rate** dictates how aggressively or conservatively the model adjusts to errors during training. **Beta1** (β_1) controls the momentum decay, while **beta2** (β_2) controls the scaling decay.

```
lstm_ae = Model(inputs, decoded)
lstm_ae.compile(Adam(learning_rate=learning_rate, beta_1=beta1, beta_2=beta2)
    , loss=loss_fuc)
```

Listing 5.3: Compile the LSTM-AE model.

Listing 5.4 depicts the *fit* function, which expects several input parameters and is used to train the model. The *train_sequences* parameter represents the data used for training. Since the LSTM-AE is an unsupervised learning model and the training data serves as both the input and the prediction target of the AE model, it is passed to the fit function twice. The *training_epochs* parameter represents the model's number of iterations going through the entire training data and the *batch_size* determines how many training sequences the model processes before updating its weights. Next, the validation data is passed to the fit function, which consists of data unseen by the model and is used to monitor the model's performance and its ability to generalize to new, unseen data. To prevent overfitting during training and to extract the best model weights after training, the *early_stopping* and *model_checkpoint* callbacks from the *tensorflow.keras.callbacks* library were incorporated into the training process.

```
history = lstm_ae.fit(
    train_sequences, train_sequences,
    epochs=training_epochs,
    batch_size=training_batch_size,
    validation_data=(test_sequences, test_sequences),
    callbacks=[early_stopping, model_checkpoint]
)
```

Listing 5.4: Training the LSTM-AE model with early stopping and checkpointing

With the general architecture of the model established, the next step is to set the aforementioned hyperparameters. To do this effectively, it is crucial to define how the model's performance will be measured and evaluated.

5.4. Evaluation Metrics

To evaluate the model's performance and ensure comparability, the same metrics commonly found in other examined research papers were employed, including accuracy, precision, recall, and F1-score. These metrics are based on the model's predictions, and confusion

matrix terms specifically true positive (TP), false negative (FN), false positives (FP), and true negative (TN). In the context of anomaly detection, **accuracy** (Equation 5.1) measures the proportion of correct predictions (both positives and negatives) out of all predictions. However, accuracy alone may not be sufficient, therefore mercies such as precision, recall, and f1-score are often used alongside. **Precision** (Equation 5.2) indicates the proportion of correctly predicted positive instances out of all instances predicted as positive. **Recall** (Equation 5.3), also known as sensitivity or true positive rate, measures the proportion of actual anomalies that were correctly identified by the model. The **F1-Score** (Equation 5.4) is the harmonic mean of precision and recall, balancing the trade-off of both metrics. In the context of this thesis, **TP** represents an anomaly predicted as such, **FN** an anomaly falsely predicted as normal, **FP** a normal packet predicted as an anomaly, and TN a correctly predicted normal packet.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{5.1}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{5.2}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5.3}$$

$$\text{F1-Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.4}$$

These metrics are calculated for every scenario (one to five) to evaluate the model. To derive the confusion matrix terms (TP, FN, FP, and TN) from the model's predictions and categorize data as benign or anomalous, a threshold must first be calculated.

5.4.1. Threshold

The threshold represents the maximum reconstruction error of the model still considered acceptable. Any reconstructed sequence or packet exceeding this threshold is marked as an anomaly, signaling a potential attack on the system. For an AE-based anomaly detection model the threshold for a reconstruction error-based AE model can be calculated at different levels, with the main ones, in this case, being sequence-level and packet-level. A threshold based on the packet-level, or **packet-based threshold**, is calculated

by calculating the MSE or MAE across all features for each packet within a sequence. A threshold based on the sequence-level, or **sequence-based threshold**, is calculated by calculating the MSE or MAE across both the packets and features within each sequence. After calculating these errors, a percentile is selected, below which a certain percentage of the reconstruction errors falls. This percentile represents the threshold.

The evaluation approach depends, among other things, on the labels available. In the attack-infused datasets used in this thesis, each packet is labeled individually. To still apply sequence-based evaluation, every sequence is labeled based on whether it contains at least one malicious packet. This ensures that all attacks can potentially be detected, even at the sequence level.

5.5. Model fine tuning

The flexibility of neural networks comes with the challenge of fine-tuning numerous hyperparameters, which control the model's behavior during training. These include the loss function, optimizer, learning rate, batch size, and layer size.

Selecting appropriate hyperparameters can significantly enhance the model's performance. However, there is no universal set of hyperparameters for different models. As a result, finding suitable hyperparameters often involves trial and error, or examining previous work found in the literature. To set acceptable parameters, I drew inspiration from the work of Hnamte et al. [16], and Luo et al [26]. Hnamte et al. evaluated their model using the loss functions MSE (Equation 2.7) and MAE (Equation 2.6), and employed the Adam optimizer with two different learning rates: 0.001 and 0.0001. Conversely, Luo et al. utilized Bayesian Optimization to fine-tune the Adam optimizer within a learning rate range of 0.01 to 0.001, as well as the momentum decay hyperparameter β_1 (range: 0.9 to 0.9999), and the scaling decay hyperparameter β_2 (range: 0.9 to 0.9999). This approach, based on Bayesian inference and Gaussian processes, aims to find the optimal value of a high-cost function in as few iterations as possible [32].

First, two models using the hyperparameters (learning rate, β_1, β_2, and layer length) described by Luo et al. and depicted in Table 5.1 were trained and evaluated to select the appropriate loss function.

Although the initial evaluation results were sobering on a packet-level (Appendix D.1), they indicate that the MSE-based model slightly outperforms the MAE-based model. However, both loss functions have their merits, as the metrics vary depending on the traffic scenario,

Hyperparameter	Value
Outer layer length	62*
Inner layer length	31
learning rate	0.0043259137
β_1	0.939844012507
β_2	0.943045819607

Table 5.1.: Hyperparameters based on [27]. (*) Since Luo et al. utilized a GRU model which usually uses a single layer length, the outer layer length was estimated based on an educated guess.

the threshold calculation, and whether sequence-level or packet-level evaluation is used. To make this decision, I choose to focus primarily on scenario5, which incorporates all traffic scenarios and the recall metric, which is a vital indicator for safety-critical applications where a priority is to maximize TP and to minimize FN. Next, both models were compared on their performance on a packet-level and sequence-level. Notably, the sequence-base threshold can be used for both sequence-level and packet-level evaluation. To decide which threshold to use for the packet-level evaluation, the performance of the models was compared with the different thresholds. Here the MAE-based model achieved better results than the sequence-based threshold. The MSE-based model, on the other hand, achieved better results with the packet-based threshold.

The conclusion derived from this experiment was to use the MSE-based model with a packet-based threshold for packet-level evaluation and a sequence-based threshold for sequence-level evaluation. This model with a validation loss of *7.27e-5* and *61,514* parameters will be referred to as the **Base Model**. The models were trained and evaluated with the sequences depicted in Table 4.3. Additionally, the number of normal and malicious packets/sequences are depicted in Subsection C.1 of the appendix. The complete metrics history of this experiment can be found in Subsection D.1 of the appendix. The batch size was empirically determined in the context of the smallest validation loss for the MSE-based model to be 10.

The remaining hyperparameters (layer size, learning rate, beta1, beta2) were fine-tuned using Bayesian Optimization [32], with the target parameter being the minimization of the validation loss.

5.5.1. Bayesian Optimization

The optimization range and optimal combination of hyperparameters resulting from Bayesian Optimization are shown in Table 5.2.

Hyperparameter	Range	Results
Outer layer size	[50, 200]	94
Inner layer size	[10, 190]	88
learning rate	[0.0001, 0.01]	0.005295088673159155
β_1	[0.9, 0.9999]	0.9183221105343581
β_2	[0.9, 0.9999]	0.9303938000716578

Table 5.2.: Optimal combination of hyperparameters derived from the Bayesian Optimization.

Reviewing the results over multiple iterations (Figure 5.2) revealed that several hyperparameter combinations achieved a similar validation loss close to zero. However, since some of these combinations involved larger layer sizes, the original choice was maintained.

The model resulting from the Bayesian Optimization has *235,958* parameters and will be referred to as the **Optimized Model**. This model was trained and evaluated using the sequences depicted in Table 4.3.

5.5.2. Model variations

To further enhance the analysis of the LSTM-AE architecture in the context of attack detection, two additional models were created and evaluated using the optimized hyperparameters: an LSTM-AE model with a smaller sequence length (**Smaller Sequence Model**) and an LSTM-AE model using data from a single service node (**Single Node Model**), thereby minimizing the variety of differently scaled signals.

For the Smaller Sequence Model, the sequence length was reduced from seven to four communication cycles, resulting in a sequence length of 52. The sliding step was adjusted to 17 for the training data. The resulting model has *235,958* parameters. The Single Node Model was trained and evaluated using data from the *Body & Dynamics* service node.

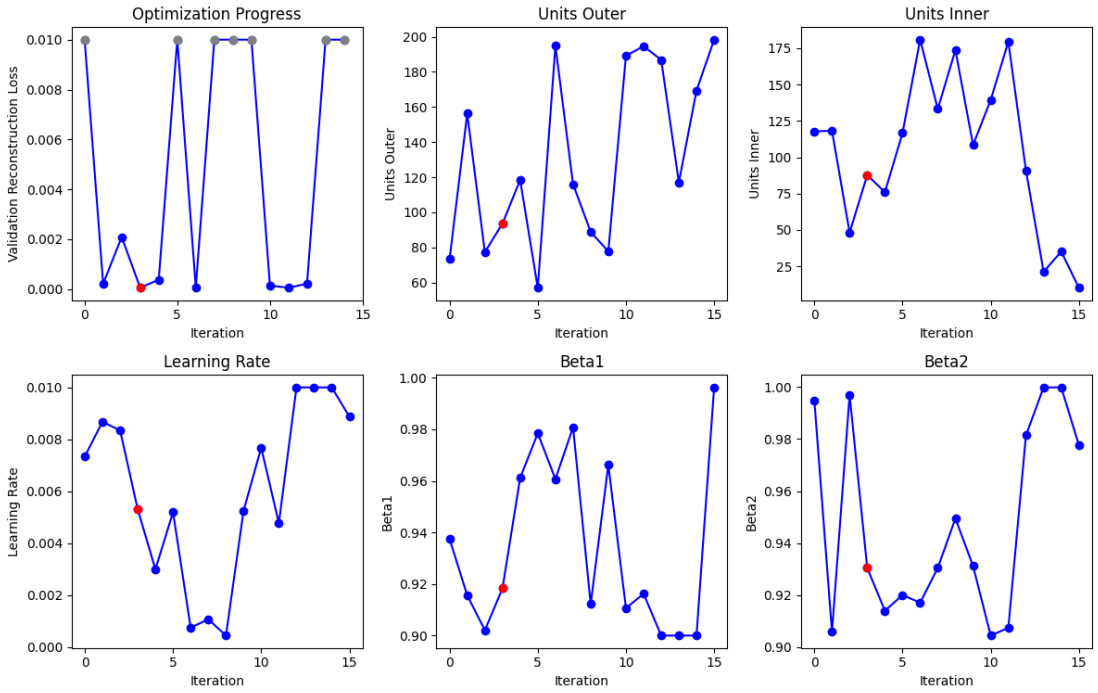


Figure 5.2.: The parameter development of the considered hyperparameters over 15 Iterations, with the red point marking the iteration with the best validation loss. The grey points all exceeded a validation loss of 0.01 and were artificially adjusted to demonstrate the relevant iterations better.

Although this node only has three signals, it was selected over the ADAS node. This choice was made because the ADAS node, with its 40 ms communication cycle, generates only a quarter of the data produced by the other nodes. The sequence length was again set to seven times the communication cycle, resulting in a length of 28, and the sliding step for the training data was adjusted to eight. The resulting model has *232,661* parameters. The sequence shapes for the Smaller Sequence Model and Single Node Model are depicted in Tables 17 and 20 of the appendix, respectively. Additionally, the number of normal and malicious packets/sequences to evaluate the Smaller Sequence Model and Single Node Model are depicted in Subsections C.2 and C.3 of the appendix.

6. Evaluation

This chapter summarizes the experimental results from all four models by comparing their performance metrics in the context of attack detection. Additionally, the inference time of the model on a laptop is calculated and compared to that on the Linux server.

6.1. Training and Validation loss

Figure 6.1 to 6.4 show the training loss and validation loss of all four models over multiple epochs. The training and validation loss converge close to zero in less than 10 epochs, across the different models.

Model	Val. Loss	Sequence-based Threshold	Packet-based Threshold
Base Model	7.268e-5	12.22e-4	7.17e-4
Optimized Model	4.912e-5	6.41e-4	5.66e-4
Smaller Seq. Model	2.638e-5	2.32e-4	2.32e-4
Single Node Model	1.450e-5	1.09e-4	0.90e-4

Table 6.1.: rounded to four significant figures

The best weights for the different models were obtained through the *early_stopping*, resulting in the validation losses depicted in Table 6.1. The results indicate that the validation loss decreases with each new model iteration, from the Base Model to the Single Node Model. However, the Single Node Model is not directly comparable to the other models in terms of training and validation loss, as it exclusively uses data from a

single service node. Additionally, Table 6.1 displays the sequence-based and packet-based thresholds resulting from the different models, calculated using the 99th percentile of the validation data's reconstruction error, closely correlated with the decreasing validation losses.

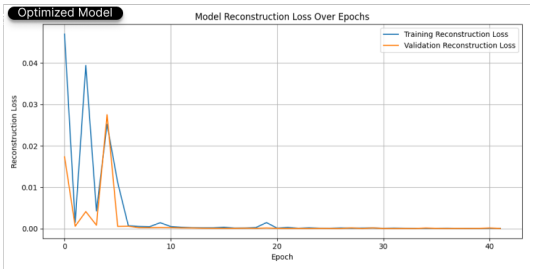


Figure 6.1.: Training and Validation loss of the Optimized Model

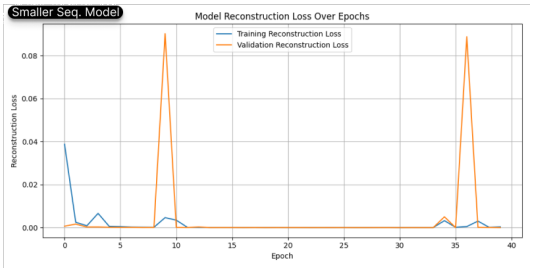


Figure 6.2.: Training and Validation loss of the Smaller Seq. Model

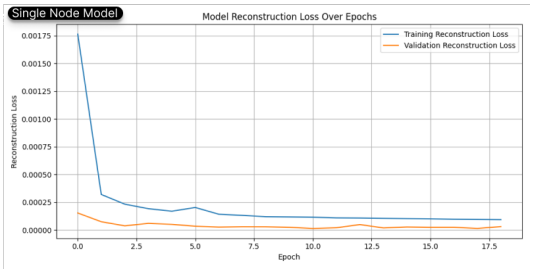


Figure 6.3.: Training and Validation loss of the Single Node Model

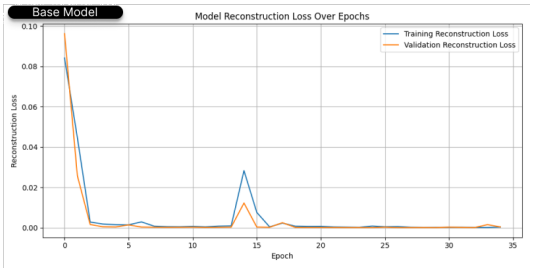


Figure 6.4.: Training and Validation loss of the Base Model

6.2. Analysis of LSTM-AE attack detection

In the following Subsections, the evaluation results of the four LSTM-AE models are analyzed. The replay and tamper attack detection are examined separately within the context of scenario5.

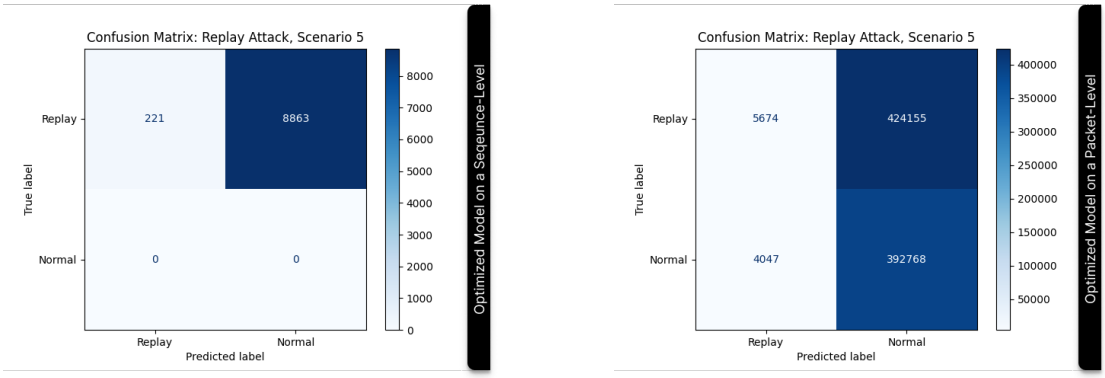
6.2.1. Replay attack evaluation

Table 6.2 compares the performance of the four LSTM-AE models in detecting replay attacks in scenario5, in the context of their validation loss. The performance across all four models is quite similar, and despite an improved validation loss, the metrics do not show significant improvement. The accuracy at the packet level remains below 49%, while the recall and F1-Score metrics are even lower, under 9% and 5% respectively for all four models. This clearly indicates that the models are not effectively detecting replay attacks. At the sequence level, the accuracy drops below 5% for all models, suggesting that most sequences are incorrectly categorized as normal, given that the replay attack dataset contains no normally labeled sequences, provided a sequence length of 91. Notably, due to the data imbalance on a sequence-level the metrics for Precision, Recall, and F1-Score are omitted.

Model	Val. Loss	Evaluation level	Accuracy	Precision	Recall	F1-Score
Base Model	7.268e-5	seq.	0.0203	-	-	-
		pkt	0.4811	0.5299	0.0179	0.0345
Optimized Model	4.912e-5	seq.	0.0203	-	-	-
		pkt	0.4820	0.5837	0.0132	0.0258
Smaller Seq. Model	2.638e-5	seq.	0.0391	-	-	-
		pkt	0.4808	0.5244	0.0167	0.0323
Single Node Model	1.450e-5	seq.	0.0418	-	-	-
		pkt	0.4834	0.5381	0.0455	0.0839

Table 6.2.: Evaluation metrics for replay attack detection in scenario5 across four different LSTM-AE models, presented in the context of their validation loss. All models utilize an MSE-based loss function, with a sequence-based threshold at the sequence level (seq.) and a packet-based threshold at the packet level (pkt.).

This assumption is further supported by analyzing the confusion matrix for replay attack detection in scenario5 at the sequence level for the optimized model, as shown in Figure 6.5a. The matrix illustrates that only 221 sequences are correctly categorized as attacks, while 8,862 sequences are incorrectly categorized as normal. The packet-level confusion



(a) sequence-level

(b) packet-level

Figure 6.5.: Confusion matrix of Optimized model of replay attack detection in scenario5.

matrix, depicted in Figure 6.5b, illustrates this more precisely, with a total of 816,923 packets being categorized as normal, although less than half of them actually are.

Because of the overall low metrics results across all models regarding replay attack detection an analysis of the models across the different traffic scenarios is refrained from. Nevertheless, the metrics are all displayed in Section D of the appendix.

6.2.2. Tamper attack evaluation

Table 6.3 compares the performance of the four LSTM-AE models in detecting tamper attacks in scenario5, in the context of their validation loss. In this Subsection, the Single Node Model is analyzed separately, as it was specifically built to examine the behavior of the LSTM-AE architecture with less variety in scale of the individual signals.

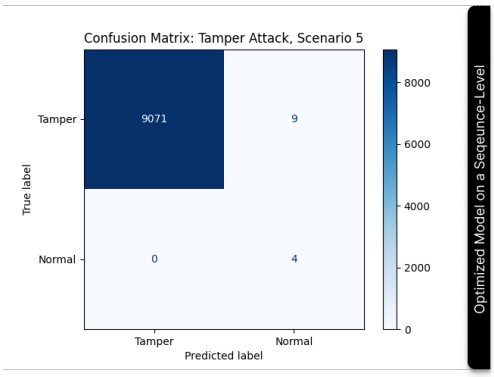
The following analysis focuses on the Base Model, Optimized Model, and Smaller Sequence Model. The results indicate that as the validation loss decreases, the accuracy, precision, and F1-Score also decrease at the packet-level. However, the recall metric increases at the packet level. Comparing recall to precision reveals that recall increases by approximately 16.09%, whereas precision decreases by about 17.08 %. This suggests that precision is more sensitive to changes in validation loss, as evidenced by the decreasing F1-Score. Ironically, this indicates that the Base Model outperforms all other models in terms of accuracy, precision, and F1-Score, despite having the highest validation loss. When

Model	Val. Loss	Evaluation level	Accuracy	Precision	Recall	F1-Score
Base Model	7.268e-5	seq.	0.9964	1.0	0.9964	0.9982
		pkt.	0.7210	0.5277	0.6605	0.5867
Optimized Model	4.912e-5	seq.	0.9990	1.0	0.9990	0.9995
		pkt.	0.5682	0.3887	0.7692	0.5164
Smaller Seq. Model	2.638e-5	seq.	0.9953	0.9999	0.9954	0.9976
		pkt.	0.5028	0.3569	0.8214	0.4976
Single Node Model	1.450e-5	seq.	0.9684	1.0	0.9684	0.9839
		pkt.	0.6782	0.4754	0.7276	0.5750

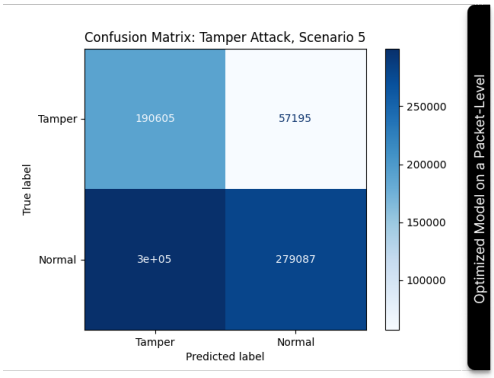
Table 6.3.: Evaluation metrics for tamper attack detection in scenario5 across four different LSTM-AE models, presented in the context of their validation loss. All models utilize an MSE-based loss function, with a sequence-based threshold at the sequence level (seq.) and a packet-based threshold at the packet level (pkt.).

focusing on the sequence-level evaluation, the Optimized Model achieves the highest accuracy. The Recall metric follows a similar trend as the accuracy, which can be attributed to the precision achieving a perfect score of one, further leading to a high F1-Score. However, the results for Precision, and consequently the F1-Score should be interpreted with caution due to the low number of normally labeled sequences (Table C). Notably, the Smaller Sequence Model exhibits the overall worst metrics, except for recall, which is the highest among all models.

The confusion matrix for the sequence-level and packet-level evaluation of the Optimized Model in the context of scenario5 is depicted in Figure 6.6. It highlights the model's accuracy at the sequence level, with only nine sequences being incorrectly classified as benign network traffic. However, when focusing on the packet-level evaluation, it becomes evident that the model has more difficulty recognizing tamper attacks, at a lower level. When examining the metrics of the Single Node Model, it becomes evident that the model does not significantly outperform the others. For instance, while the F1-Score improves compared to the Optimized Model and the Smaller Sequence Model, it is slightly worse than that of the Base Model. The accuracy follows a similar pattern, with only the recall metric surpassing that of the Base Model. However, when compared to the recall metric of



(a) sequence-level



(b) packet-level

Figure 6.6.: Confusion matrix of Optimized model of tamper attack detection in scenario5.

the Optimized Model and Smaller Sequence Model, the Single Node Model underperforms. It is important to mention that comparisons between the Single Node Model and the other models should be made cautiously, as the Single Node Model is built on less and differently scaled data.

6.3. Interpretation of LSTM-AE attack detection

As in Subsection 6.2.2, the focus is first set on interpreting the Base Model, Optimized Model, and Smaller Sequence Model. The evaluation results for replay and tamper attack detection, analyzed in 6.2, indicate that the LSTM-AE models are not capable of detecting replay attacks on the network at neither on the sequence-level nor the packet-level. However, the results for tamper attacks at the sequence-level are more promising, with the best model achieving an accuracy of approximately 99.90%. When examining the evaluation results at the packet level, it becomes evident that the models were unable to achieve satisfactory outcomes, with accuracy and recall metrics peaking at 72.10% and 82.14%, respectively. Notably, the metrics are obtained from different models. The trend of increasing recall with a decreasing validation loss suggests a negative correlation for the recall metric at the packet level, while accuracy, precision, and F1-Score display a positive correlation with validation loss. The above-mentioned results suggest that the model was unable to improve the overall anomaly detection at the packet level, and the opposing

trends of recall and precision could be attributed to the model's lower validation loss. A lower validation loss results in a lower threshold, leading to more anomalies being detected by default, which artificially decreases the number of FN where anomalies are falsely classified as normal. However, the number of normal packets with a high reconstruction increases as well. This could be attributed to the LSTM-AE architecture and how the data is processed. When a sequence containing multiple malicious packets is processed, the malicious packets may affect the reconstruction of the entire sequence, including the benign data within that sequence.

The evaluation results from the Single Node Model indicate that its performance is within a similar range compared to the other models, without significant improvement or decline. Considering that this model was not specifically fine-tuned and was built using hyperparameters derived from Bayesian analysis designed for a model with different scaling and data amounts, it is likely that the metrics could be improved with a fine-tuning process tailored to the model's specific needs. However, it is unlikely that the Single Node Model will achieve significantly better results at the packet-level due to the same architectural limitations described, previously.

6.4. Scenario Evaluation

In this section, I will analyze the tamper attack detection performance of the Optimized Model across the four different traffic scenarios. The analysis for replay attack detection is omitted due to the poor results. While the other models could be analyzed similarly, this is refrained from for the sake of clarity. However, the metric results for all traffic scenarios are provided in Section D of the appendix.

Table 6.4 illustrates the metrics results across the different traffic scenarios in the context of the number of training sequences.

The results show that the metrics improve with each consecutive traffic scenario, peaking at scenario3 in terms of accuracy, precision, and F1-Score at the packet level. However, the metrics in scenario4 decline compared to scenario3. Conversely, the recall metric behaves oppositely to the other metrics. Analyzing the correlations (Appendix 11) highlights the contrasting trends between recall and the other metrics in relation to the number of sequences.

When focusing on the sequence-level, it is evident that the trend of sequence-level evaluation outperforming packet-level evaluation persists across all four traffic scenarios.

Scenario	No. Train Seq.	Evaluation Level	Accuracy	Precision	Recall	F1-Score
1	7,902	seq pkt.	0.9997 0.4973	- 0.3557	0.9997 0.8328	- 0.4985
2	10,308	seq pkt.	0.9995 0.5456	1.0000 0.3776	0.9995 0.7949	0.9997 0.5120
3	4,434	seq. pkt.	0.9975 0.6309	0.9994 0.4292	0.9981 0.7080	0.9988 0.5344
4	2,147	seq pkt.	0.9962 0.5725	0.9987 0.3893	0.9975 0.7512	0.9981 0.5128

Table 6.4.: Sequence-level and packet-level evaluation metrics for tamper attack detection across four different traffic scenarios of the Optimized Model. '-' indicates that there are no normally labeled sequences, rendering precision and F1-Score inconclusive.

Analyzing the correlations reveals a strong positive relationship between the number of training sequences and each of the metrics at the sequence-level (Appendix 11). However, the number of sequences alone does not fully explain the model's behavior, as evidenced by scenario2, which has the most sequences but not the highest metrics.

It is difficult to pinpoint the exact reasons for these performance variations. The discrepancies could be attributed to factors such as data imbalance in the different training datasets, the variety in scale across the different traffic scenarios, or the order in which the data is fed to the model.

6.5. Explainability of Reconstructions

Disclaimer: *All explanations and analyses in this section are based on single observations and may not be representative of all models or the general LSTM-AE architecture. The conclusions drawn here require further investigation. However, the general concept can be explained in this manner.*

The results of an LSTM-AE model may, to some extent, be explainable by design. This stems from the fact that the model itself does not directly make predictions in a categorical sense. Instead, the predictions result from the model's ability to recognize known data patterns and effectively compress and decompress those patterns. When data significantly deviates from the norm, the model will struggle to reconstruct it, resulting in a high reconstruction error. This characteristic can help explain why certain sequences are categorized as anomalous and can be used to analyze the model's shortcomings or strengths. By examining the reconstruction error, it is possible to identify sequences that are poorly reconstructed, potentially indicating an attack on the network. Analyzing the reconstruction error of the validation data can provide insights into the model's performance and potential weaknesses.

The following analysis focuses on the explainability of the Optimized Model in the context of its reconstruction error, independent of the threshold. Figure 6.7 illustrates signal2 of a benign sequence before and after reconstruction.

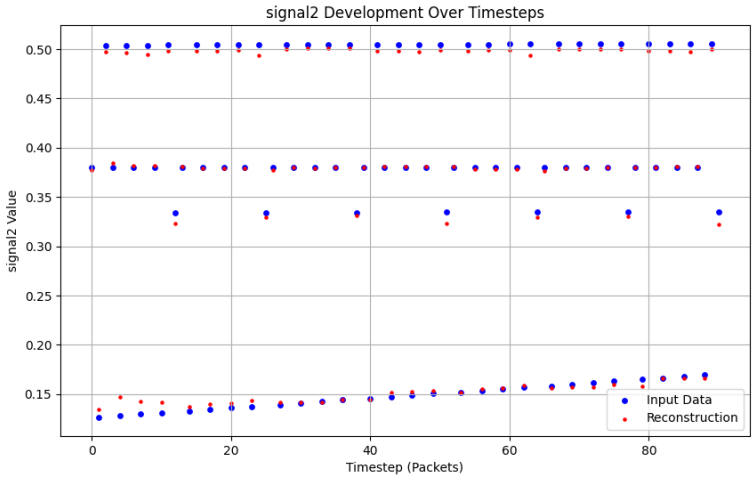


Figure 6.7.: Benign sequence of signal2 before and after reconstruction.

The Figure illustrates that the model can recognize data patterns to some extent but still struggles with reconstructing certain data points. Upon closer examination, a pattern of four horizontal lines is depicted, representing the different Signals from the service nodes. The data points between 0.3 and 0.35 can be attributed to the ADAS service due to its longer communication cycle compared to the other services. The complete validation data

for signal2 is presented in Figure 12 of the appendix, highlighting that the model still has room for improvement.

After analyzing the validation data, the focus shifts to explaining the tampered data. Figure 6.8 illustrates a tampered sequence and the model's reconstructions. In this case, the input outliers are easily identifiable. By examining the abscissa, the reconstructions may suggest where the model expected the data points to be. This assumption is further supported by Figure 6.9, which depicts all the tampered data for signal2 from scenario4. Although the input data appears evenly scattered, the reconstructed data reveals a pattern, particularly between the values 0 and 0.2.

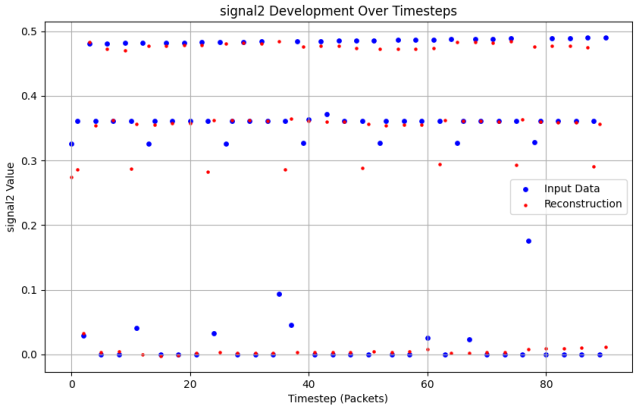


Figure 6.8.: Tampered sequence of signal2 before and after reconstruction.

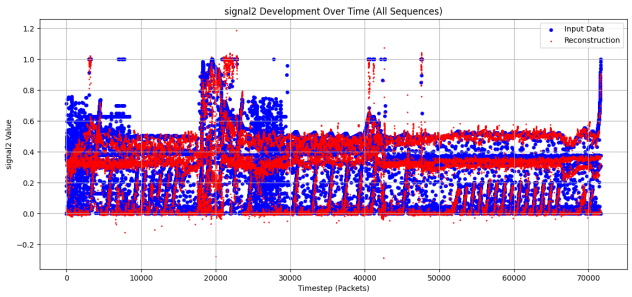


Figure 6.9.: Tampered signal2 sequences from scenario4

Another interesting insight derived from the reconstructions is how severely other features of a packet are influenced by the tampered sequences. Figure 6.10 illustrates the message_id feature from both the validation data (bottom of the figure) and the tamper attack-infused data (top of the figure). While the reconstructed validation data is less precise at certain points, it is not as scattered as the reconstructions of the tampered data.

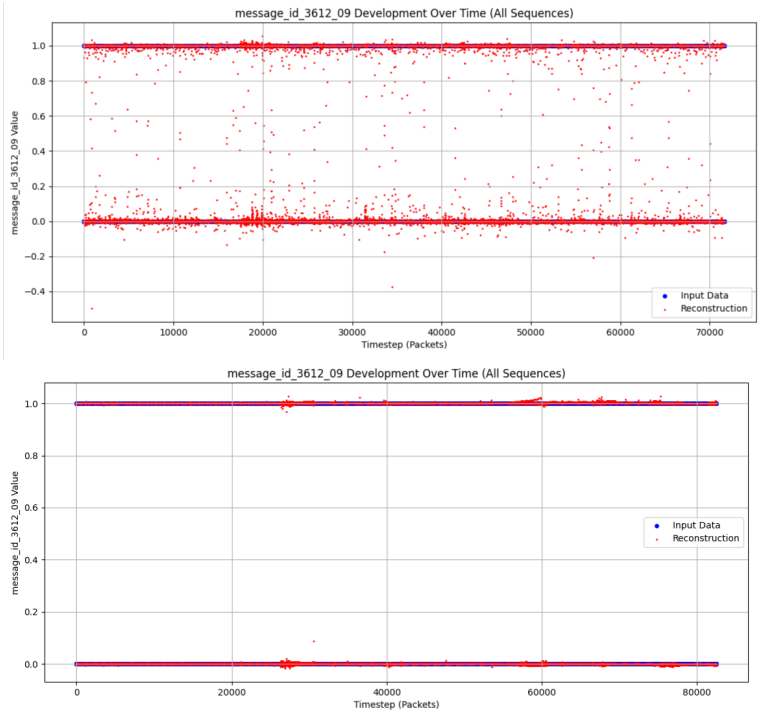
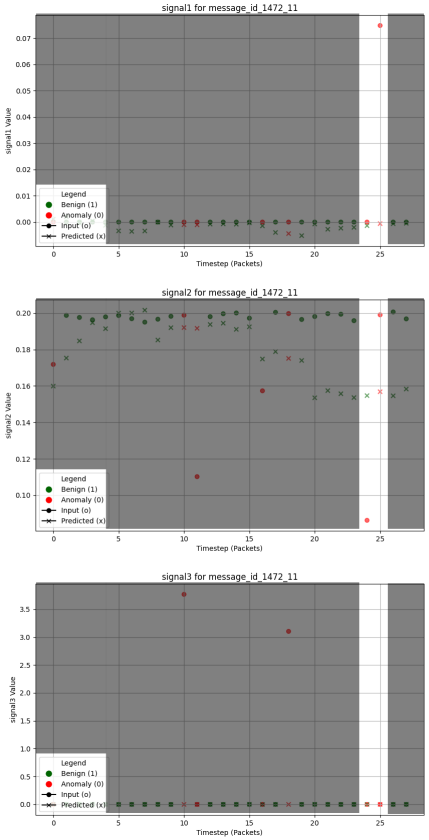


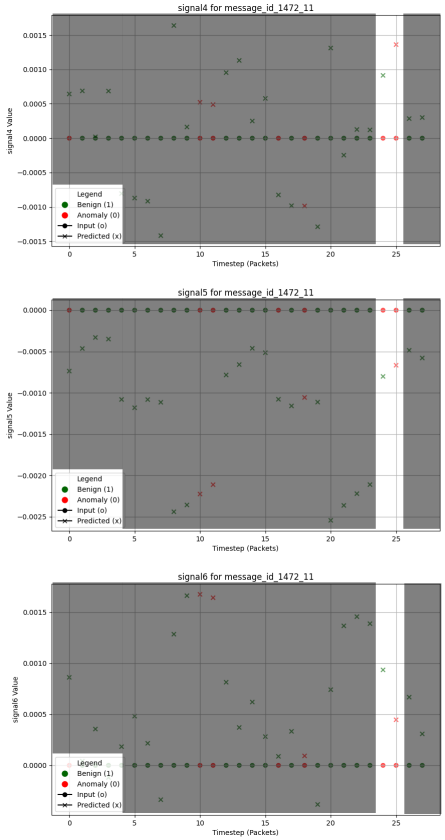
Figure 6.10.: message_id from validation (bottom) and tampered data (top).

Analyzing the reconstructions at the packet level reveals that reconstruction errors can be misleading and may not provide clear insights into the data. Figure 6.11 depicts the tampered signals one to six from a single sequence of the HPC service node. The 24th and 25th packets are highlighted across all signals. The 25th packet represents a TP, an anomaly correctly detected, while the 24th packet represents a FN, an anomaly that was not recognized. It is important to note that the HPC service node only transmits two signals, meaning the other input signals are zero. The detected anomaly in the 25th packet likely stems from signal1, as it is the only data point different from zero, yet the model predicts it as zero. Conversely, the anomaly in the 24th packet probably originates

from signal2. The model continues the pattern from the previous predictions and the reconstruction error does not exceed the necessary threshold to be categorized as an anomaly.



(a) signal1 to signal3.



(b) signal4 to signal6.

Figure 6.11.: Tampered signals and reconstructions from Body & Dynamics Node.

6.6. Inference time

Especially in safety-critical environments, the inference time plays a vital role in selecting security systems. The IEEE802.1DG seminar [18] defines the real-time requirement of in-vehicle traffic for safety-relevant control to be below 1 ms. The **Inference time** refers to the amount of time it takes for a machine learning model to make a prediction.

Device	Measurement	Optimized Model	Smaller Sequence Model	Base Model
Server	Inference time per sequence (ms)	0.5727	0.4164	0.5812
	Inference time per packet	0.0063	0.0080	0.0064
	Number of model parameters	235,958	235,958	61,514
Laptop	Inference time per sequence (ms)	1.4445	0.8608	0.9965
	Inference time per packet	0.0159	0.0166	0.0101
	Number of model parameters	235,958	235,958	61,514

Table 6.5.: Inference time and parameter number of the Base Model, Optimized Model, and Smaller Sequence Model on a Laptop and Linux Server system. Batch size of 906 for Optimized Model and Base Model (seq. length 91). Batch size of 1587 for Smaller Seq. Model. (seq. length 52).

Device	Measurement	Optimized Model	Smaller Sequence Model	Base Model
Server	Inference time per sequence (ms)	110.4896	107.1882	111.2938
	Inference time per packet	1.2142	2.0613	1.2230
	Number of model parameters	235,958	235,958	61,514
Laptop	Inference time per sequence (ms)	78.5091	60.7965	79.0594
	Inference time per packet	0.8627	1.1692	0.8688
	Number of model parameters	235,958	235,958	61,514

Table 6.6.: Inference time and parameter number of the Base Model, Optimized Model, and Smaller Sequence Model on a Laptop and Linux Server system. Batch size of one for Optimized Model and Base Model (seq. length 91). Batch size of 1 for Smaller Seq. Model (seq. length 52).

7. Discussion

In this chapter, I will compare my evaluation results with those found in the literature and highlight the strengths and weaknesses of the LSTM-AE architecture for this specific use case. Additionally, in the context of Future Work (Section 7.4), a short feasibility analysis was conducted on a Rasberry PI 4 Model B, boosted with a Coral USB Accelerator, capable of performing 4 tera-operations per second (TOPS), using 0.5 watts for each TOPS with its on-board Edge Tensor Processing Unit (TPU) coprocessor.

7.1. Comparison to other models

This section will compare the performance of the **Optimized Model** for **tamper attack detection** to similar models from the literature (Table 7.1). The results for replay attack detection are omitted, as the model was unable to detect this type of attack.

The best results were achieved in the sequence-level performance evaluation of the proposed model. The LSTM-AE model shows similar performance at the sequence-level, compared to the other models. However, due to the significant data imbalance, the precision and F1-Score results should be interpreted with caution. When comparing these results to the packet-level evaluation, it becomes clear that the model performs poorly at the packet level.

7.2. Model Discussion

The evaluation results presented in Section 6.2 for all four models indicate that a model based on LSTM-AE architecture is not suitable for replay attack detection. This limitation can be attributed to the nature of replay data, which closely mimics benign data patterns.

Reference	Model	Dataset	Features	Accuracy	Precision	Recall	F1-Score
Mushtaq et al. [31]	AE-LSTM	KD-DTest+	30	89%	88%	94%	91%
Mahmoud et al. [28]	AE-LSTM	NSLKDD	41	98.89%	98.89%	98.88%	98.88%
Hnamte et al. [16]	LSTM-AE	CI-CIDS2017	81	99.99%	99.99%	99.99%	99.99%
Luo et al. [26]	Multi-Gru	[27]	6	97.76%	99.98%	98.89%	99.43%
This Thesis (Pkt.)	LSTM-AE	[27]	10	56.82%	38.87%	76.92%	51.64%
This Thesis (Seq.)	LSTM-AE	[27]	10	99.90%	100%	99.90%	99.95%

Table 7.1.: Performance comparison of the proposed LSTM-AE model (Optimized Model for tamper attack detection) with other models.

Since the model is trained on these patterns, it fails to distinguish between replayed and normal data. When focusing on tampered attacks, it is evident that the model's performance is significantly better, with a peak accuracy of approximately 99.90% at the sequence-level. However, the model's performance drops drastically when categorizing individual packets (packet-level). The maximum accuracy of 72.10% was achieved with the Base Model. When comparing the different models at the packet level in the context of their validation loss, it becomes apparent that the recall metric has a negative correlation with validation loss, while accuracy, precision, and F1-Score show a positive correlation.

This could indicate that the model does not genuinely improve in tamper attack detection behavior. A possible explanation might be that the reconstruction errors across the different models do not change significantly, remaining relatively consistent. However, as the validation loss decreases, the threshold is adjusted downward, leading to more packets being categorized as malicious by default. It is important to note that this conclusion is based solely on the evidence provided by the metrics and is not conclusively proven. This trend is broken by the Single Node Model, which was implemented to test how an LSTM-AE model performs with a single data scale per signal. While the model's performance

results are similar to those of the other models, it was not fine-tuned and was simply trained using the hyperparameters derived from Bayesian Optimization.

In terms of explainability, the reconstruction error of an LSTM-AE model can be used to analyze the model's ability to reconstruct benign data patterns and to highlight its weaknesses. However, while analyzing the reconstructions may provide some insights into the model's performance, it does not fully explain the model's predictions it is not suitable for analyzing individual predictions. This limitation can be attributed to the model's approach of compressing and decompressing the sequence as a whole, which can result in malicious packets affecting the reconstruction of benign data packets. Nonetheless, it may be useful for visualizing certain patterns and expectations of the model. This is illustrated in Figure 6.11a for signal1, where the model's prediction aligns perfectly with the benign data pattern. However, this alignment is somewhat circumstantial and requires a tedious analysis process. It would be interesting to further investigate this with a model that performs better at the packet-level.

To conclude this section, the LSTM-AE model performs well for tamper attack detection at the sequence level. Additionally, it exhibits some interesting characteristics in terms of explainability that need further investigation.

7.3. Reality Check

When it comes to integrating an LSTM-AE model into an IDS system, further research is still necessary. A major concern is the model's performance on real-world data. The artificial dataset used in this thesis may not accurately reflect the complexities of a real environment, where data can be noisy and outliers are inevitable. Additionally, the dataset used in this study only represents four different traffic situations, whereas, in reality, traffic scenarios are much more variable. Moreover, the model's performance at the packet level does not meet the necessary requirements for a safety-critical IDS system. Although the sequence-level evaluation shows better results, further research is needed to investigate the model's performance with differently balanced attack datasets.

7.4. Future Work

This section outlines areas that were not addressed during the implementation and evaluation of the LSTM-AE model but would be valuable for future research and applications.

7.4.1. Permormance

To enhance the model's performance, various techniques can be explored, such as dropout, adjusting the number of layers, increasing the amount of data, and experimenting with different methods of feeding data to the model. Another approach could involve fine-tuning the Single Node Model. Additionally, exploring the Gated Recurrent Unit (GRU)-AE architecture type instead of LSTM-AE could be a promising alternative.

If the model's performance can be improved, it would be valuable to integrate it into an IDS pipeline and measure its real-time performance, particularly on embedded systems with GPU acceleration. For integration into a lightweight IDS system, the model itself should be lightweight. TensorFlow offers the possibility of transforming the model into a more lightweight version. However, initial tests indicate that the model needs to meet certain requirements, which are not fulfilled by this specific architecture type (Appendix 1). To meet these requirements, the model would need to be adjusted to eliminate *Flex Operations*, which are not natively supported by TensorFlow Lite. This limitation complicates the integration of the model into an experimental setup involving a Rasberry PI 4 Model B, enhanced with a Coral USB Accelerator. The specifically designed "PyCoral API (the pycoral module) is built atop the TensorFlow Lite Python API" [25].

7.4.2. Explainablity

The explainability of ML models in safety-critical environments is crucial. The LSTM-AE model has certain architectural characteristics that need further investigation. Additionally, tools like SHapley Additive exPlanation (SHAP) can be utilized to explain how specific features impact the reconstruction error. The SHAP method used to explain ML models is based on Shapley values. These values are used to measure the importance of input features. Shapley values are based on cooperative game theory and aim to quantify each player's contribution (in this case, a feature) in a game where several players collaborate to acquire a reward (the prediction in this case). Each player's reward is determined by

their contribution to the result. Hence, a Shapely value represents the average marginal contribution of a feature in a particular instance across all possible combinations of features. The marginal contribution of a feature is calculated as the difference between the prediction made when the feature is included and the prediction made when the feature is excluded. This calculation is performed for all possible subsets where the feature's contribution is present. The weighted average of these contributions is the Shapley value.

8. Conclusion

This thesis analyzed the feasibility of an LSTM-AE model architecture for intrusion detection on SOME/IP payload data. The LSTM-AE model leverages unsupervised learning to effectively compress and reconstruct data. A high reconstruction error could indicate an attack on the system. The LSTM component integrates time dependence into the model, enabling it to detect attacks over entire data sequences. In total, four different models were created and compared using data provided by the literature. This data was generated from an IVN simulation, depicting four service nodes communicating over the SOME/IP protocol and sending sensor and processed data over the network. The primary focus was on detecting attacks on the payload, leading to the omission of various other features. Two types of attacks were analyzed: replay and tampering attacks. The models performance was evaluated in detecting attacks on both entire sequences and individual packets. The results show that LSTM-AE models are not effective in identifying replay attacks and perform poorly in packet-level detections, with an accuracy of 56.82%. However, at the sequence level, the models can detect tamper attacks with an accuracy of 99.90%. Additionally, the LSTM-AE architecture exhibits interesting characteristics in terms of explainability, which need further investigation. In conclusion, the LSTM-AE architecture demonstrates promising results in detecting tamper attacks across entire data sequences, though further research is needed to fully explore its potential and limitations.

Appendix

A. Signals

A.1. HPC

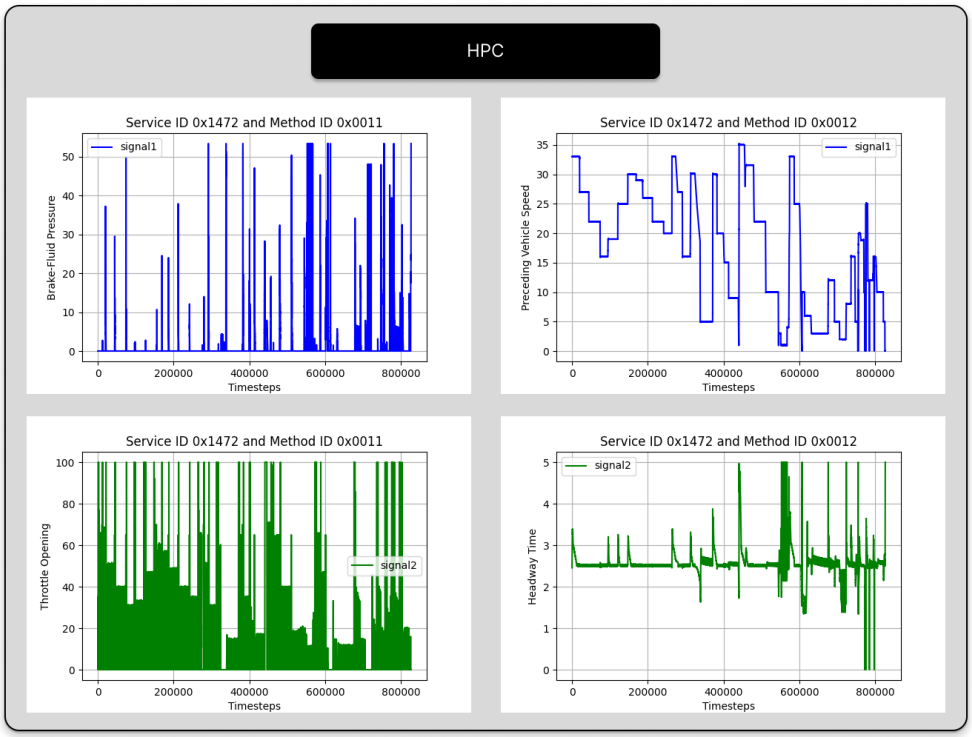


Figure 1.: Normal data: Every signal from the HPC node.

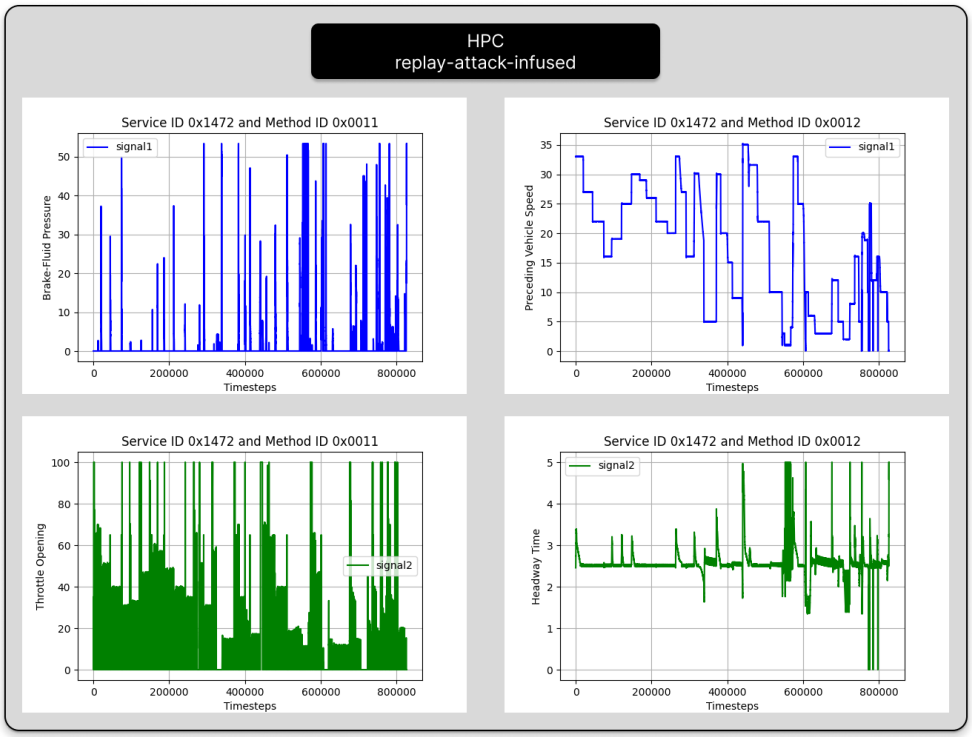


Figure 2.: Replay-attack-infused data: Every signal from the HPC node.

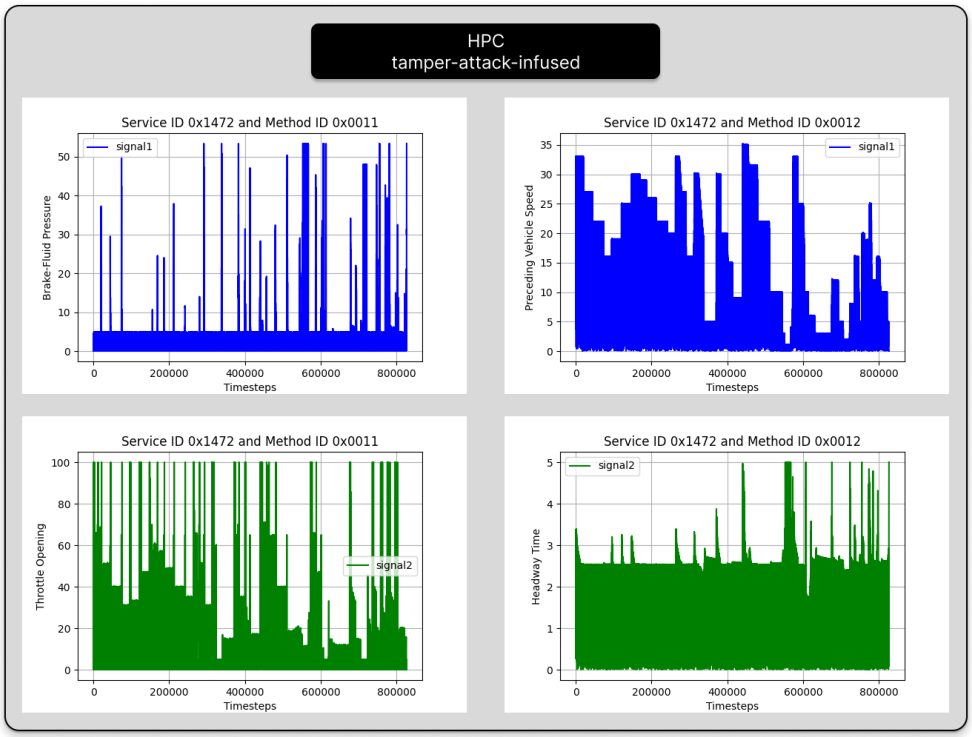


Figure 3.: Tamper-attack-infused data: Every signal from the HPC node.

A.2. ADAS

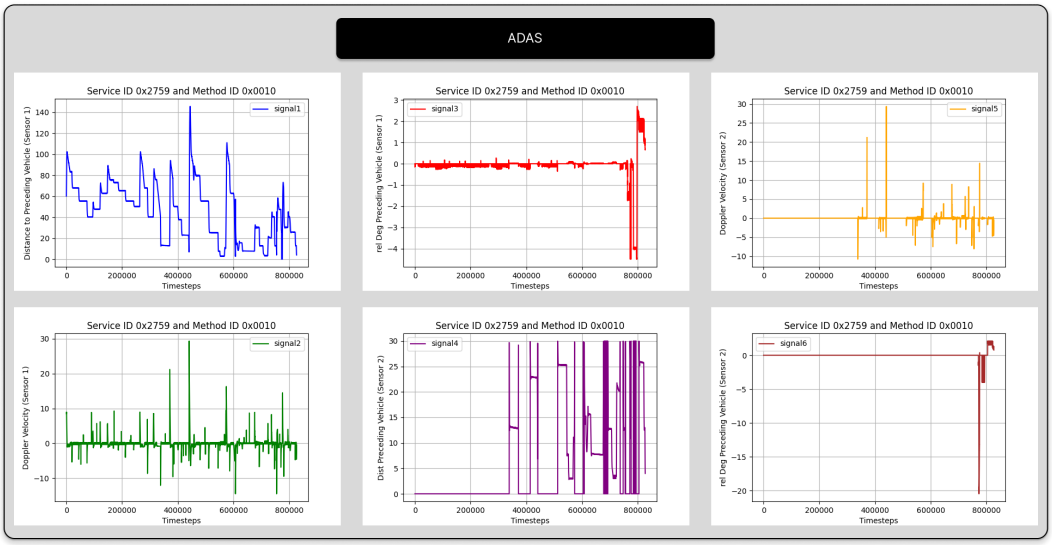


Figure 4.: Normal data: Every signal from the ADAS node.

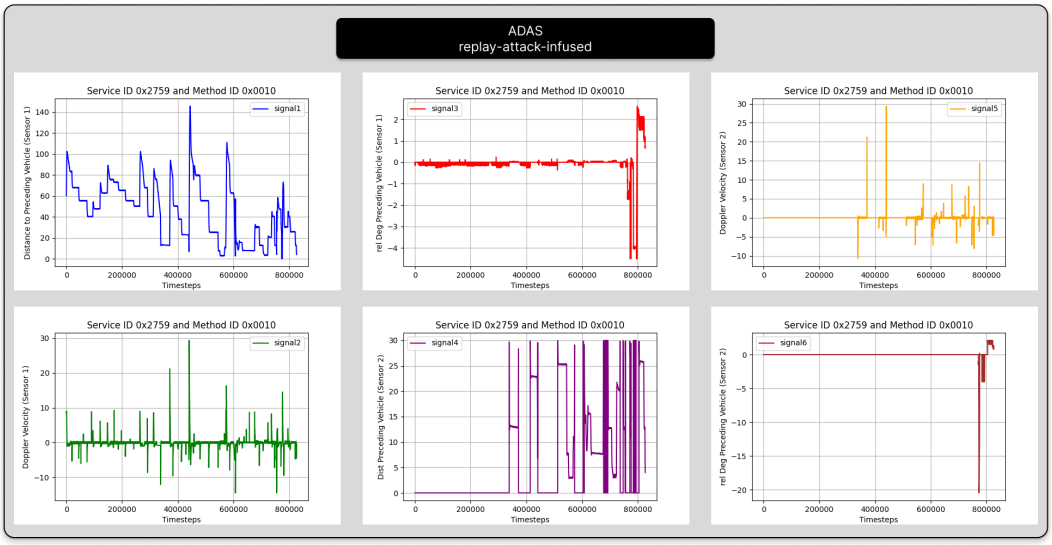


Figure 5.: Replay-attack-infused data: Every signal from the ADAS node.

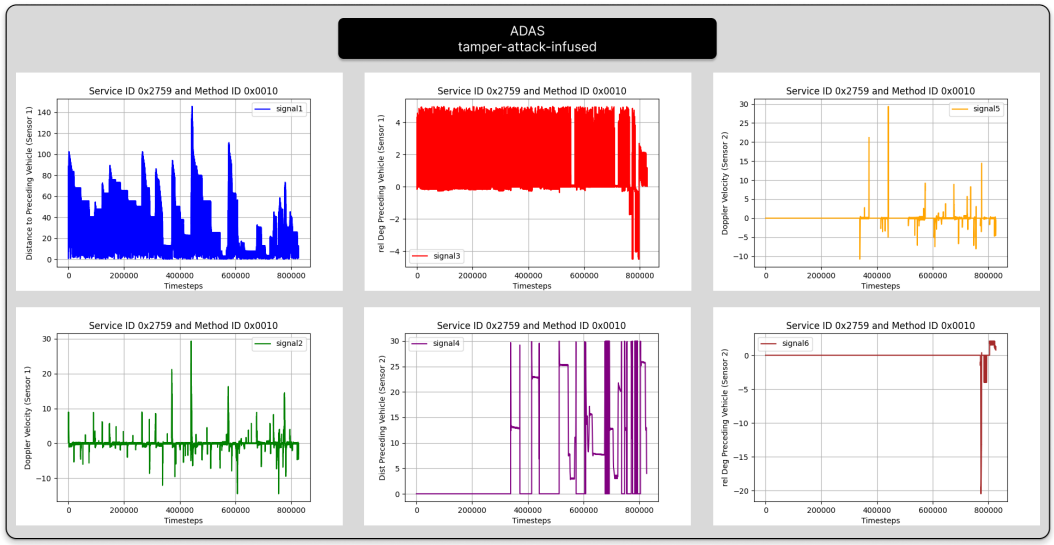


Figure 6.: Tamper-attack-infused data: Every signal from the ADAS node.

A.3. Body & Dynamics

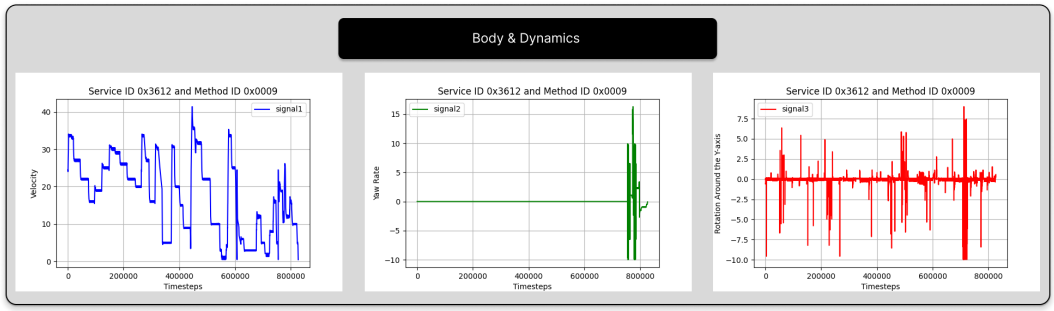


Figure 7.: Normal data: Every signal from the Body & Dynamics node.

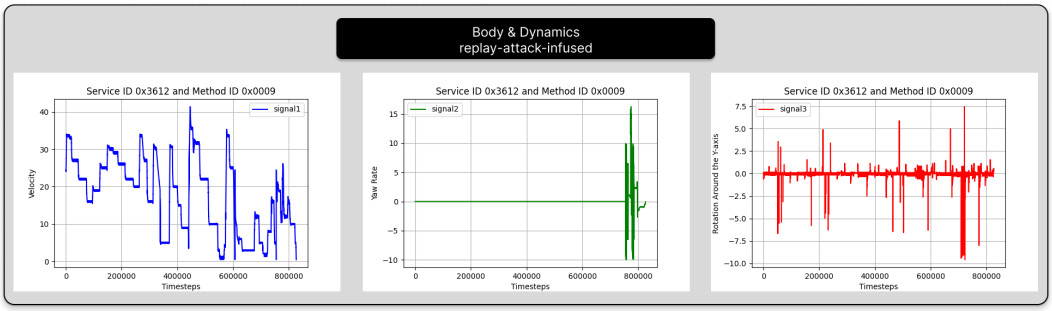


Figure 8.: Replay-attack-infused data: Every signal from the Body & Dynamics node.

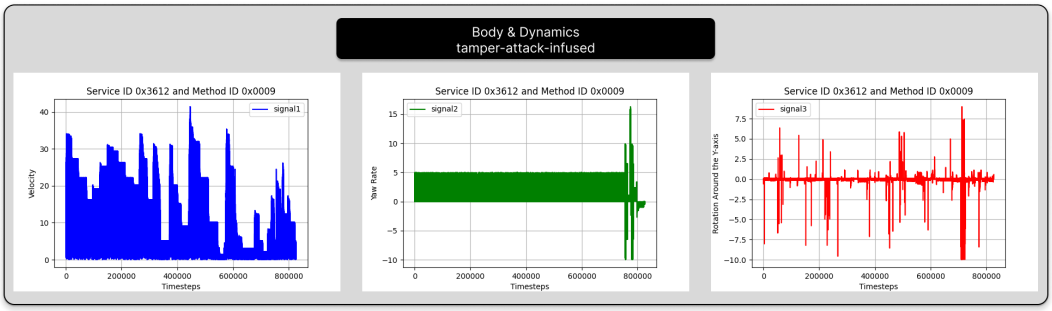


Figure 9.: Tamper-attack-infused data: Every signal from the Body & Dynamics node.

B. Correlation Matrix

B.1. Feature Analysis

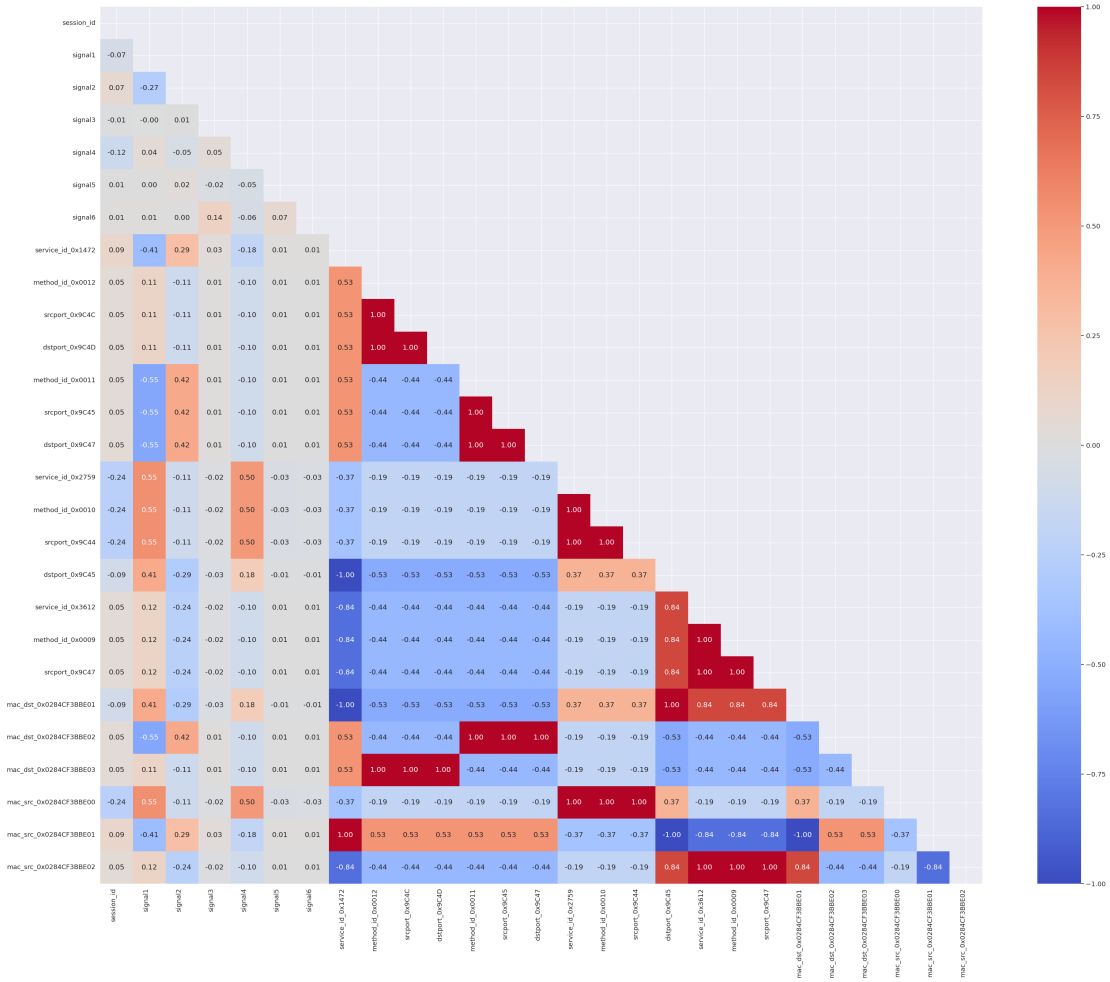


Figure 10.: Correlation matrix of the dataset with constant features excluded. [26]

B.2. Scenario Evaluation

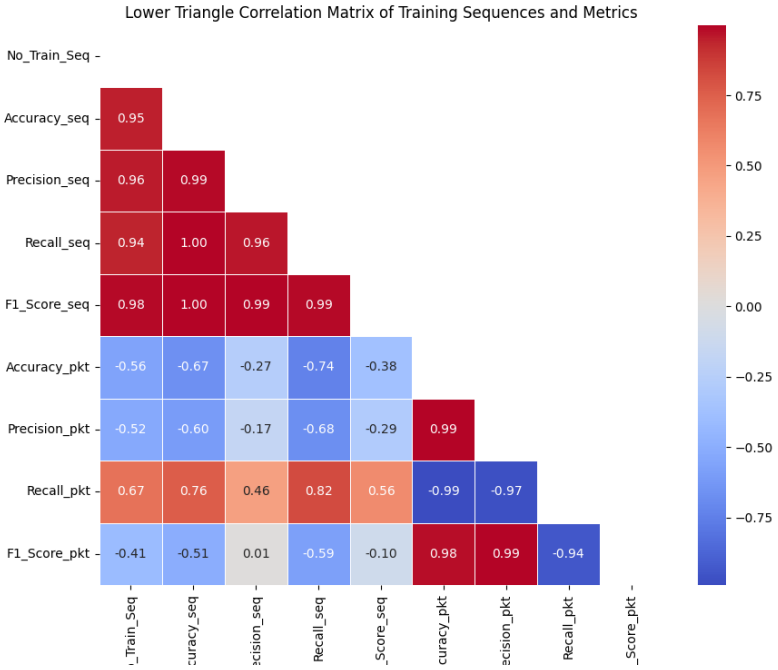


Figure 11.: Correlation matrix of the metrics in the context of the amount of training data, on a sequence-level (seq.) and packet-level (pkt.).

C. Attack Dataset Balance

C.1. Sequence length 91

Scenario	Attack Type	Number of Attack Sequences	Number of Normal Sequences
1	Replay	2895	0
	Tamper	2895	0
2	Replay	3776	0
	Tamper	3775	1
3	Replay	1625	0
	Tamper	1621	4
4	Replay	787	0
	Tamper	786	1
5	Replay	9084	0
	Tamper	9080	4

Table 1.: Number of attack and normal sequences for each scenario and attack type at the sequence level, given a sequence length of 91.

Scenario	Attack Type	Number of Attack Packets	Number of Normal Packets
1	Replay	136,994	126,451
	Tamper	79,031	184,414
2	Replay	178,675	164,941
	Tamper	103,050	240,566
3	Replay	76,882	70,993
	Tamper	44,250	103,625
4	Replay	37,236	34,381
	Tamper	21,450	50,167
5	Replay	429,829	396,815
	Tamper	247,800	578,844

Table 2.: Number of attack and normal packets for each scenario and attack type at the packet-level, given a sequence length of 91.

C.2. Sequence length 52

Scenario	Attack Type	Number of Attack Sequences	Number of Normal Sequences
1	Replay	5067	0
	Tamper	5067	0
2	Replay	6609	0
	Tamper	6606	3
3	Replay	2843	0
	Tamper	2837	6
4	Replay	1378	0
	Tamper	1375	3
5	Replay	15898	0
	Tamper	15888	10

Table 3.: Number of attack and normal sequences for each scenario and attack type at the sequence-level, given a sequence length of 52.

Scenario	Attack Type	Number of Attack Packets	Number of Normal Packets
S1	Replay	137,015	126,469
	Tamper	79,044	184,440
S2	Replay	178,698	164,970
	Tamper	103,050	240,618
S3	Replay	76,879	70,957
	Tamper	44,250	103,586
S4	Replay	37,258	34,398
	Tamper	21,450	50,206
S5	Replay	429,857	396,839
	Tamper	247,800	578,896

Table 4.: Number of attack and normal packets for each scenario and attack type at the packet-level, given a sequence length of 52.

C.3. Single Node Data

Scenario	Attack Type	Number of Attack Sequences	Number of Normal Sequences
S1	Replay	2895	0
	Tamper	2895	0
S2	Replay	3776	0
	Tamper	3775	1
S3	Replay	1625	0
	Tamper	1621	4
S4	Replay	787	0
	Tamper	786	1
S5	Replay	9084	0
	Tamper	9080	4

Table 5.: Number of attack and normal sequences for each scenario and attack type from the Body & Dynamics service node at sequence-level, given a sequence length of 28.

Scenario	Attack Type	Number of Attack Packets	Number of Normal Packets
S1	Replay	42,152	38,908
	Tamper	24,374	56,686
S2	Replay	54,977	50,751
	Tamper	31,679	74,049
S3	Replay	23,656	21,844
	Tamper	13,468	32,032
S4	Replay	11,457	10,579
	Tamper	6,585	15,451
S5	Replay	132,255	122,097
	Tamper	76,111	178,241

Table 6.: Number of attack and normal packets for each scenario and attack type from the Body & Dynamics service node, given a sequence length of 28.

D. Metrics

D.1. MAE vs. MSE

Sequence-Level metrics

Scenario	Attack-Type	Accuracy	Precision	Recall	F1-Score
1	Replay	0.0024	1.0000	0.0024	0.0048
	Tamper	0.9813	1.0000	0.9813	0.9906
2	Replay	0.0093	1.0000	0.0093	0.0184
	Tamper	0.9756	0.9997	0.9759	0.9877
3	Replay	0.0234	1.0000	0.0234	0.0457
	Tamper	0.9508	0.9994	0.9513	0.9747
4	Replay	0.1398	1.0000	0.1398	0.2453
	Tamper	0.9962	0.9987	0.9975	0.9981
5	Replay	0.0196	1.0000	0.0196	0.0384
	Tamper	0.9750	0.9998	0.9752	0.9873

Table 7.: **Sequence-level** evaluation metrics for different scenarios and attack types, of an LSTM-AE model, with **MAE**-based loss function, and **sequence-based threshold**. Due to the insufficient number of normally labeled sequences in the attack datasets, given a **sequence length of 91**, Precision and F1-Score need to be considered with caution and do not produce meaningful results.

Scenario	Attack-Type	Accuracy	Precision	Recall	F1-Score
1	Replay	0.0069	1.0000	0.0069	0.0137
	Tamper	0.9983	1.0000	0.9983	0.9991
2	Replay	0.0095	1.0000	0.0095	0.0189
	Tamper	0.9976	1.0000	0.9976	0.9988
3	Replay	0.0222	1.0000	0.0222	0.0433
	Tamper	0.9932	1.0000	0.9932	0.9966
4	Replay	0.1169	1.0000	0.1169	0.2093
	Tamper	0.9962	1.0000	0.9962	0.9981
5	Replay	0.0203	1.0000	0.0203	0.0397
	Tamper	0.9964	1.0000	0.9964	0.9982

Table 8.: **Sequence-level** evaluation metrics for different scenarios and attack types, of an LSTM-AE model, with **MSE**-based loss function, and **sequence-based threshold**. Due to the insufficient number of normally labeled sequences in the attack datasets, given a **sequence length of 91**, Precision and F1-Score need to be considered with caution and do not produce meaningful results.

Packet-Level metrics

Scenario	Attack-Type	Accuracy	Precision	Recall	F1-Score
1	Replay	0.4800	0.4972	0.0058	0.0115
	Tamper	0.6950	0.4942	0.7123	0.5836
2	Replay	0.4822	0.5607	0.0193	0.0373
	Tamper	0.6955	0.4944	0.6757	0.5710
3	Replay	0.4812	0.5218	0.0254	0.0485
	Tamper	0.7227	0.5348	0.5631	0.5486
4	Replay	0.4884	0.5253	0.1666	0.2530
	Tamper	0.6452	0.4405	0.6828	0.5355
5	Replay	0.4814	0.5247	0.0272	0.0517
	Tamper	0.7159	0.5208	0.6533	0.5796

Table 9.: **Packet-level** evaluation metrics for different scenarios and attack types, of an LSTM-AE model, with **MAE**-based loss function, **sequence length of 91**, and **sequence-based threshold**.

Scenario	Attack-Type	Accuracy	Precision	Recall	F1-Score
1	Replay	0.4798	0.4688	0.0025	0.0050
	Tamper	0.7290	0.5495	0.5359	0.5426
2	Replay	0.4806	0.5583	0.0051	0.0102
	Tamper	0.7207	0.5369	0.4999	0.5177
3	Replay	0.4803	0.5125	0.0088	0.0173
	Tamper	0.7143	0.5293	0.4076	0.4605
4	Replay	0.4843	0.5224	0.0941	0.1595
	Tamper	0.6947	0.4908	0.5147	0.5025
5	Replay	0.4804	0.5156	0.0122	0.0238
	Tamper	0.7386	0.5777	0.4752	0.5215

Table 10.: **Packet-level** evaluation metrics for different scenarios and attack types, of an LSTM-AE model, with **MAE**-based loss function, **sequence length of 91**, and **packet-based threshold**.

Scenario	Attack-Type	Accuracy	Precision	Recall	F1-Score
1	Replay	0.4800	0.5040	0.0055	0.0108
	Tamper	0.7439	0.5654	0.6319	0.5968
2	Replay	0.4802	0.5110	0.0089	0.0174
	Tamper	0.7344	0.5515	0.6127	0.5805
3	Replay	0.4806	0.5243	0.0111	0.0217
	Tamper	0.7162	0.5254	0.5351	0.5302
4	Replay	0.4833	0.5258	0.0638	0.1137
	Tamper	0.7083	0.5114	0.5863	0.5463
5	Replay	0.4807	0.5275	0.0124	0.0243
	Tamper	0.7399	0.5626	0.5948	0.5783

Table 11.: **Packet-level** evaluation metrics for different scenarios and attack types, of an LSTM-AE model, with **MSE**-based loss function, **sequence length of 91**, and **sequence-based threshold**.

Scenario	Attack-Type	Accuracy	Precision	Recall	F1-Score
1	Replay	0.4802	0.5106	0.0076	0.0149
	Tamper	0.7194	0.5243	0.6961	0.5982
2	Replay	0.4801	0.5035	0.0141	0.0274
	Tamper	0.7127	0.5160	0.6771	0.5857
3	Replay	0.4807	0.5180	0.0174	0.0336
	Tamper	0.7146	0.5200	0.6033	0.5586
4	Replay	0.4838	0.5225	0.0825	0.1425
	Tamper	0.6953	0.4933	0.6406	0.5574
5	Replay	0.4811	0.5299	0.0179	0.0345
	Tamper	0.7210	0.5277	0.6605	0.5867

Table 12.: **Packet-level** evaluation metrics for different scenarios and attack types, of an LSTM-AE model, with **MSE**-based loss function, **sequence length of 91**, and **packet-based threshold**.

D.2. Optimized Model metrics

Scenario	Attack-Type	Accuracy	Precision	Recall	F1-Score
1	Replay	0.0135	1.0000	0.0135	0.0266
	Tamper	0.9997	1.0000	0.9997	0.9998
2	Replay	0.0140	1.0000	0.0140	0.0277
	Tamper	0.9995	1.0000	0.9995	0.9997
3	Replay	0.0215	1.0000	0.0215	0.0422
	Tamper	0.9975	0.9994	0.9981	0.9988
4	Replay	0.1512	1.0000	0.1512	0.2627
	Tamper	0.9962	0.9987	0.9975	0.9981
5	Replay	0.0243	1.0000	0.0243	0.0475
	Tamper	0.9990	1.0000	0.9990	0.9995

Table 13.: **Sequence-level** evaluation metrics for different scenarios and attack types, of an LSTM-AE model, with **optimized hyperparameters**, **MSE**-based loss function, and **sequence-based threshold**. The threshold is calculated by averaging over the sequences. Due to the insufficient number of normally labeled sequences in the attack datasets, given a **sequence length of 91**, Precision and F1-Score need to be considered with caution and do not produce meaningful results.

Scenario	Attack-Type	Accuracy	Precision	Recall	F1-Score
1	Replay	0.4812	0.6533	0.0048	0.0095
	Tamper	0.4973	0.3557	0.8328	0.4985
2	Replay	0.4828	0.6593	0.0110	0.0217
	Tamper	0.5456	0.3776	0.7949	0.5120
3	Replay	0.4822	0.6133	0.0111	0.0217
	Tamper	0.6309	0.4292	0.7080	0.5344
4	Replay	0.4846	0.5357	0.0660	0.1175
	Tamper	0.5725	0.3893	0.7512	0.5128
5	Replay	0.4820	0.5837	0.0132	0.0258
	Tamper	0.5682	0.3887	0.7692	0.5164

Table 14.: **Packet-level** evaluation metrics for different scenarios and attack types, of an LSTM-AE model with **optimized hyperparameters**, **MSE**-based loss function, **sequence length of 91**, and **packet-based threshold**.

D.3. Smaller Sequence Model

Scenario	Attack-Type	Accuracy	Precision	Recall	F1-Score
1	Replay	0.0184	1.0000	0.0184	0.0360
	Tamper	0.9988	1.0000	0.9988	0.9994
2	Replay	0.0244	1.0000	0.0244	0.0476
	Tamper	0.9974	1.0000	0.9974	0.9987
3	Replay	0.0239	1.0000	0.0239	0.0467
	Tamper	0.9884	1.0000	0.9884	0.9941
4	Replay	0.2046	1.0000	0.2046	0.3398
	Tamper	0.9949	0.9993	0.9956	0.9974
5	Replay	0.0391	1.0000	0.0391	0.0752
	Tamper	0.9953	0.9999	0.9954	0.9976

Table 15.: **Sequence-level** evaluation metrics for different scenarios and attack types, of an LSTM-AE model, with **optimized hyperparameters**, **MSE**-based loss function, and **sequence-based threshold**. Due to the insufficient number of normally labeled sequences in the attack datasets, given a **sequence length of 52**, Precision and F1-Score need to be considered with caution and do not produce meaningful results.

Scenario	Attack-Type	Accuracy	Precision	Recall	F1-Score
1	Replay	0.4802	0.5198	0.0042	0.0084
	Tamper	0.4417	0.3351	0.8751	0.4847
2	Replay	0.4800	0.5001	0.0114	0.0223
	Tamper	0.5003	0.3567	0.8295	0.4989
3	Replay	0.4817	0.5694	0.0138	0.0269
	Tamper	0.5920	0.4019	0.7443	0.5220
4	Replay	0.4847	0.5228	0.1030	0.1720
	Tamper	0.4904	0.3490	0.8116	0.4881
5	Replay	0.4808	0.5244	0.0167	0.0323
	Tamper	0.5028	0.3569	0.8214	0.4976

Table 16.: **Packet-level** evaluation metrics for different scenarios and attack types, of an LSTM-AE model with **optimized hyperparameters**, **MSE**-based loss function, **sequence length of 52**, and **packet-based threshold**.

Scenario	Data Type	Number of Sequences	Number of Packets per Sequence	Number of Features per Packet
1	Training Validation Replay & Tamper	13,947 506 5,067	52	10
2	Training Validation Replay & Tamper	18,192 660 6,609	52	10
3	Training Validation Replay & Tamper	7,826 284 2,843	52	10
4	Training Validation Replay & Tamper	3,792 137 1,378	52	10
5	Training Validation Replay & Tamper	43,757 1,587 15,898	52	10

Table 17.: Sequence shapes used to train and evaluate the **Smaller Sequence Model**. The training data utilizes a sequence length of 52 and a sliding step of 17 for all scenarios. Attack and validation data utilize a sequence length and sliding step of 52.

D.4. Single Node Model

Scenario	Attack-Type	Accuracy	Precision	Recall	F1-Score
1	Replay	0.0052	1.0000	0.0052	0.0103
	Tamper	0.9900	1.0000	0.9900	0.9950
2	Replay	0.0085	1.0000	0.0085	0.0168
	Tamper	0.9738	1.0000	0.9738	0.9867
3	Replay	0.0406	1.0000	0.0406	0.0781
	Tamper	0.9483	1.0000	0.9482	0.9734
4	Replay	0.3596	1.0000	0.3596	0.5290
	Tamper	0.9250	1.0000	0.9249	0.9610
5	Replay	0.0418	1.0000	0.0418	0.0803
	Tamper	0.9684	1.0000	0.9684	0.9839

Table 18.: **Sequence-level** evaluation metrics for different scenarios and attack types, of an LSTM-AE model, with **optimized hyperparameters**, **MSE**-based loss function, and **sequence-based threshold**. Due to the insufficient number of normally labeled sequences in the attack datasets, given a **sequence length of 28**, Precision and F1-Score need to be considered with caution and do not produce meaningful results. Only data from **Body & Dynamics Node** was incorporated in the training and evaluation of the model.

Scenario	Attack-Type	Accuracy	Precision	Recall	F1-Score
1	Replay	0.4800	0.5000	0.0022	0.0043
	Tamper	0.6684	0.4681	0.7555	0.5781
2	Replay	0.4804	0.5203	0.0098	0.0192
	Tamper	0.6842	0.4823	0.7353	0.5826
3	Replay	0.4872	0.5864	0.0463	0.0859
	Tamper	0.7349	0.5428	0.6627	0.5968
4	Replay	0.4982	0.5253	0.3615	0.4283
	Tamper	0.5521	0.3746	0.7449	0.4985
5	Replay	0.4834	0.5381	0.0455	0.0839
	Tamper	0.6782	0.4754	0.7276	0.5750

Table 19.: **Packet-level** evaluation metrics for different scenarios and attack types, of an LSTM-AE model with **optimized hyperparameters**, **MSE**-based loss function, **sequence length of 28**, and **packet-based threshold**. Only data from **Body & Dynamics Node** was incorporated in the training and evaluation of the model.

Scenario	Data Type	Number of Sequences	Number of Packets per Sequence	Number of Features per Packet
1	Training Validation Replay & Tamper	9,118 289 2,895	52	3
2	Training Validation Replay & Tamper	11,894 377 3,776	52	3
3	Training Validation Replay & Tamper	5,116 162 1625	52	3
4	Training Validation Replay & Tamper	2,478 78 787	52	3
5	Training Validation Replay & Tamper	28,606 906 9,084	28	3

Table 20.: Sequence shapes used to train and evaluate the **Single Node Model**. The training data utilizes a sequence length of 28 and a sliding step of 8 for all scenarios. Attack and validation data utilize a sequence length and sliding step of 28.

E. Reconstructions

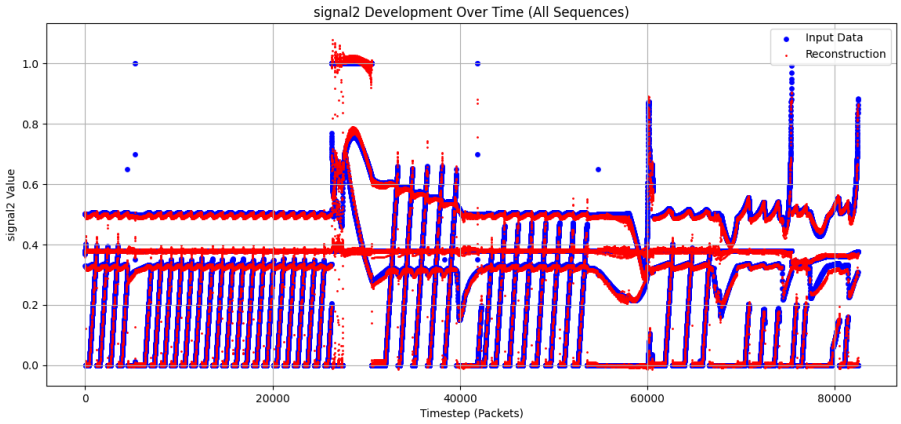


Figure 12.: Complete signal2 validation data before and after reconstruction.

F. TensorFlow Light Warning

Listing 1: TensorFlow light conversion warning.

```
[...]: W tensorflow/[...]
TFLite interpreter needs to link Flex delegate in order to run the model
    since it contains the following Select TFop(s): Flex ops:
    FlexTensorListReserve, FlexTensorListSetItem, FlexTensorListStack
```

G. LSTM-AE Architecture

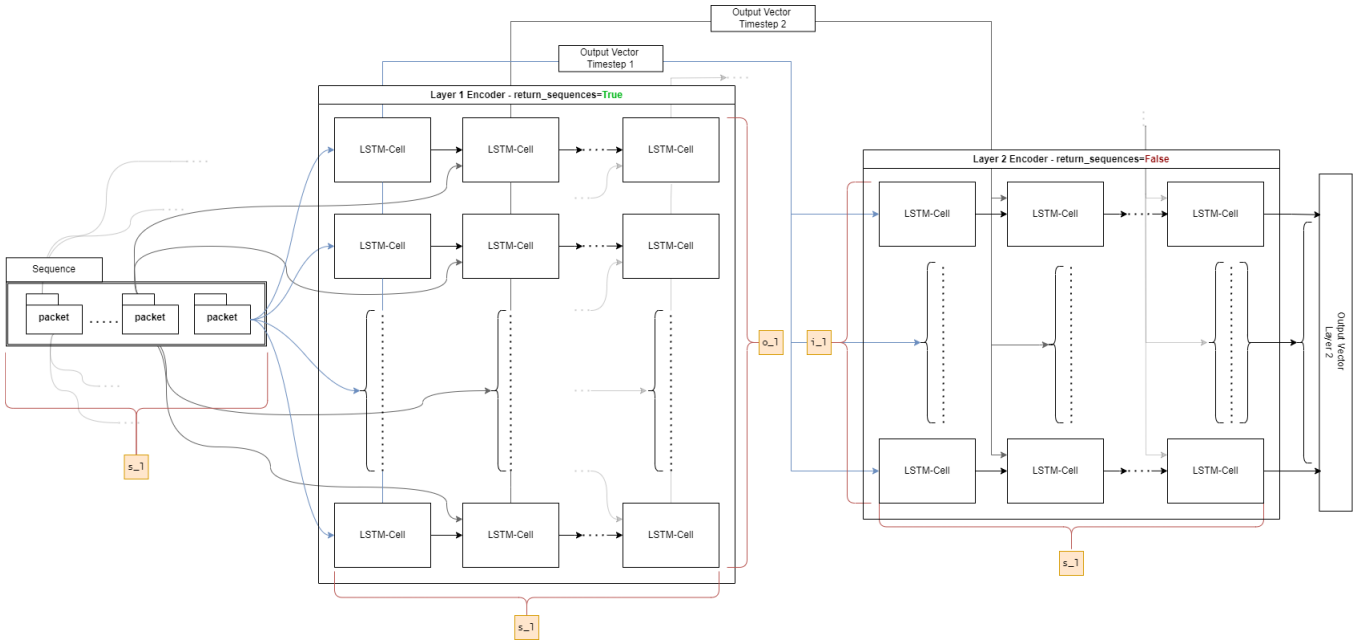


Figure 13.: Input of a sequence into the LSTM-AE network. s_l represents the sequence length, o_l represents the length (size) of the outer layer, and i_l the length of the inner layer. Setting return_sequences to true allows each cell to emit a signal at each timestep.

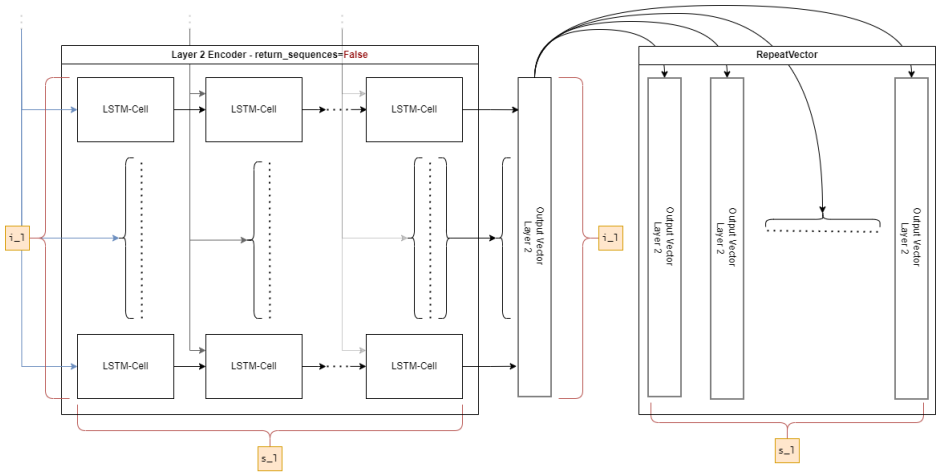


Figure 14.: The RepeactVector multiplies (s_l times) the output from the encoder and acts as a bridge to the decoder.

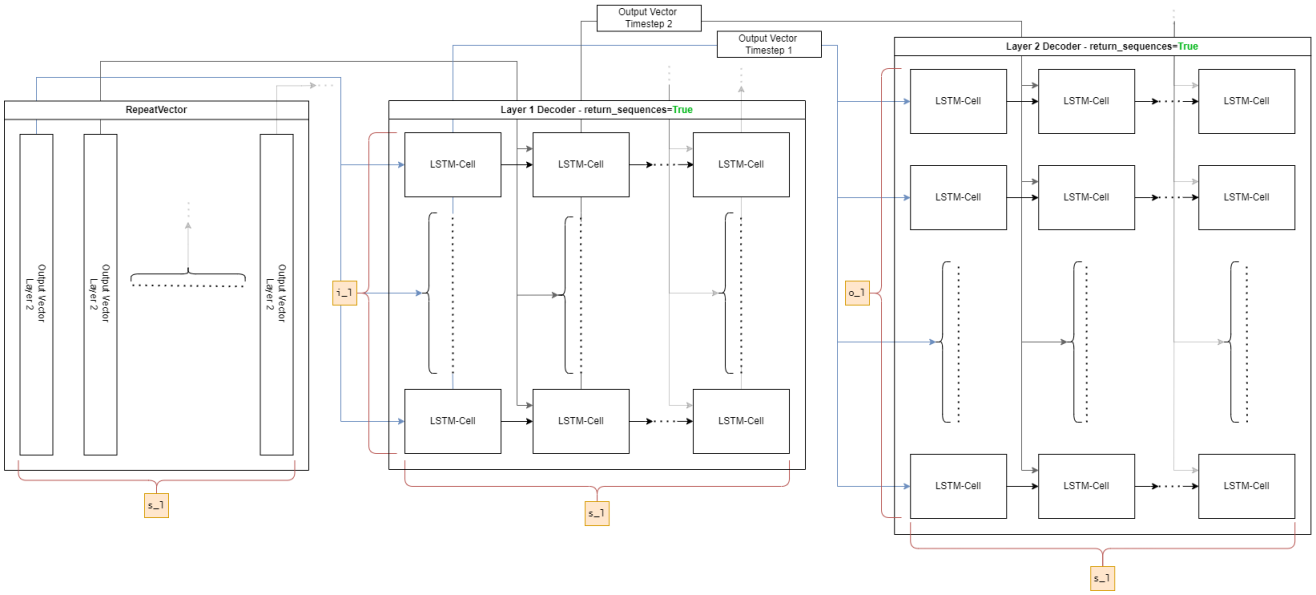


Figure 15.: The data from the RepeatVector gets fed to the decoder, over s_l time steps. The decoder decompresses the data by increasing the neurons from the inner to the outer layer.

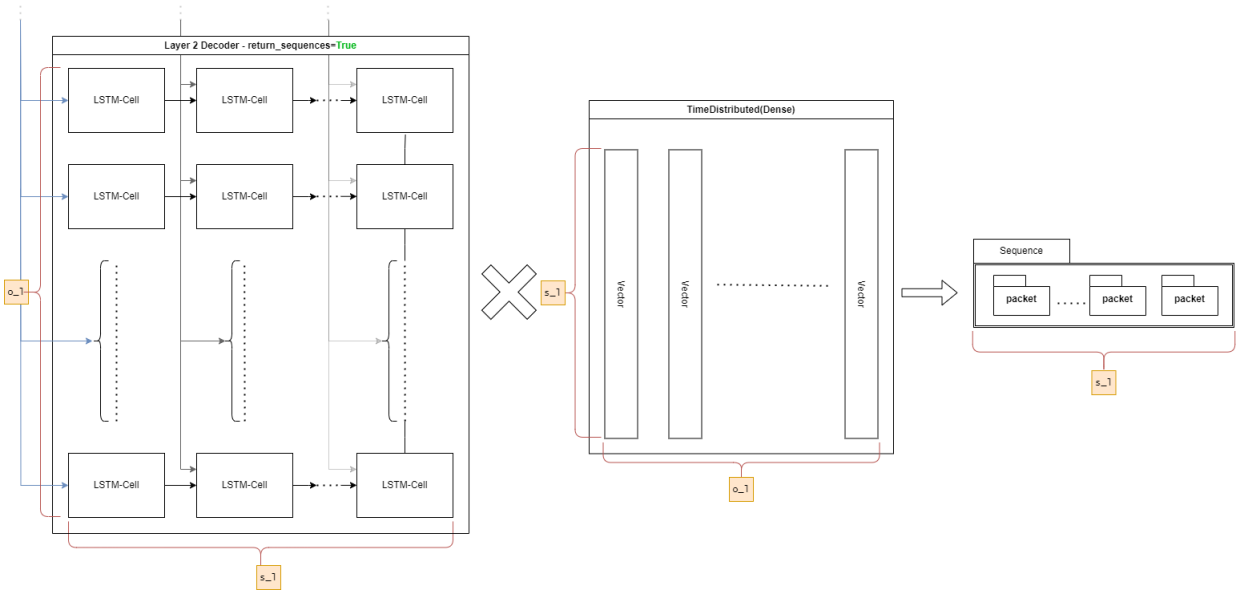


Figure 16.: The TimeDistributed layer creates a vector with a length equal to the number of features outputted from the previous layer. A matrix multiplication between the output of the decoder and the TimeDistributed layer produces the reconstructed sequence.

Abbreviations

ACC Adaptive Cruise Control 35, 41

Adam Adaptive Moment Estimation 46, 49

ADAS Advanced Driver-Assistance Systems 7, 33, 35, 36, 41, 52, 61, 108

AE Auto-Encoder 8, 11, 24–28, 31, 33, 37, 47, 48, 70

AETH Automotive Ethernet 7, 8, 11–15

AI Artificial Intelligence 8, 9, 11, 23, 29

AUC Area Under the Curve 32

AUTOSAR AUTomotive Open System ARchitecture 15

BMW Bayerische Motoren Werke 12

CEP Complex Event Processing 31

CSV Comma-separated value 35, 41

DCU Domain Controller Unit 12

DDoS Distributed Denial of Service 32

DL Deep Learning 8, 11, 24, 26, 29, 33, 44

DoS Denial of Service 31, 33

ECU Electronic Control Units 7, 12, 15, 18, 31

FFNN Feedforward Neural Network 25, 27

FN false negative 48, 50, 59, 63

FP false positives 48

GRU Gated Recurrent Unit 70

GUI Graphical User Interface 35, 36, 108

HPC High Performance Computing 35, 36, 63, 108

i_l inner layer length/size 45, 46

IDS Intrusion Detection System 3, 7–9, 11, 22, 23, 31–33, 69, 70

IVN In-Vehicle Network 3, 7–9, 11–15, 18, 33, 35–37, 72, 108

LSTM Long Short-Term Memory 3, 8, 11, 25–27, 31, 33, 42, 46, 72

LSTM-AE hybridized Long Short-Term Memory and Auto-Encoder 3, 8, 9, 23, 31, 33, 39, 42, 44, 45, 47, 51, 54–56, 58–61, 67–70, 72

MAC Media Access Control 14, 38, 39

MAE Mean Absolute Error 28, 33, 49, 50

MITM Man-in-the-middle 8, 31, 37

ML Machine Learning 3, 9, 11, 23, 24, 29, 32, 34, 70

ms milliseconds 36

MSE Mean Squared Error 28, 33, 49, 50

NN Neural Networks 9, 11, 24, 25

o_l outer layer length/size 44–46

OPEN One Pair EtherNET 14

OSI Open Systems Interconnection 15, 38

RNN Recurrent Neural Network 25, 26, 32

ROC Receiver Operating Characteristics 32

RPC Remote Procedure Calls 9, 15, 22, 36, 108

s_l Sequence Length 44

SD Service Discovery 18

SDV Software-defined Vehicle 7, 9, 11, 29

SHAP SHapley Additive exPlanation 70

SOME/IP standardized Scalable service-Oriented MiddlewarE over IP 3, 7–9, 11, 15–18, 21–23, 31, 32, 34–36, 38, 44, 72, 108

SOME/IP-SD SOME/IP Service Discovery 18, 19, 21, 22, 31, 32

SVS Surround View System 14

TN true negative 48

TOPS tera-operations per second 67

TP true positive 48, 50, 63

TPU Tensor Processing Unit 67

TTL Time-to-Live 19

VCU Vehicle Processing Unit 13, 14, 108

XAI eXplainable Artificial Intelligence 11, 29

List of Figures

Listings

List of Tables

Bibliography

[1] Natasha Alkhatib, Hadi Ghauch, and Jean-Luc Danger. “SOME/IP Intrusion Detection using Deep Learning-based Sequential Models in Automotive Ethernet Networks”. In: *2021 IEEE 12th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. 2021.

[2] Zhou Andy et al. *Software-Defined Vehicles – A Forthcoming Industrial Evolution*. 2021. URL: https://www.iese.fraunhofer.de/blog/continuous-engineering-in-devops/ (visited on 06/14/2024).

[3] AUTOSAR. *About AUTOSAR*. URL: https://www.autosar.org/about/ (visited on 07/15/2024).

[4] AUTOSAR. *SOME/IP Protocol Specification*. Tech. rep. 2023.

[5] AUTOSAR. *SOME/IP Service Discovery Protocol Specification*. Tech. rep. 2023.

[6] Stefan Axelsson. “The base-rate fallacy and the difficulty of intrusion detection”. In: *ACM Trans. Inf. Syst. Secur.* (2000).

[7] *BroadR-Reach Physical Layer Transceiver Specification For Automotive Applications*. Broadcom Corporation. 2014. URL: https://www.ieee802.org/3/1TPCESG/public/BroadR_Reach_Automotive_Spec_V3.0.pdf (visited on 07/21/2024).

[8] Marco De Vincenzi et al. “A Systematic Review on Security Attacks and Countermeasures in Automotive Ethernet”. In: *ACM Comput. Surv.* (2024).

[9] Felix Gail et al. “Evaluation of Decision Tree-Based Rule Derivation for Intrusion Detection in Automotive Ethernet”. In: *2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. 2023.

[10] Felix Clemens Gail, Roland Rieke, and Florian Fenzl. “RulEth: Genetic Programming-Driven Derivation of Security Rules for Automotive Ethernet”. In: *Machine Learning and Knowledge Discovery in Databases: Applied Data Science and Demo Track*. 2023.

[11] P. García-Teodoro et al. “Anomaly-based network intrusion detection: Techniques, systems and challenges”. In: *Computers and Security* (2009).

[12] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly, 2019.

[13] 2024 Robert Bosch GmbH. *The software-defined vehicle*. 2024. URL: https://www.bosch-mobility.com/en/mobility-topics/software-defined-vehicle/ (visited on 06/14/2024).

[14] Resing Gregor. *The Software Defined Vehicle*. June 6, 2023. URL: https://www.ibm.com/blogs/digitale-perspektive/2023/06/the-software-defined-vehicle/ (visited on 06/14/2024).

[15] Nadine Herold et al. "Anomaly detection for SOME/IP using complex event processing". In: *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. 2016.

[16] Vanlalruata Hnamte et al. "A Novel Two-Stage Deep Learning Model for Network Intrusion Detection: LSTM-AE". In: *IEEE Access* (2023).

[17] IEEE. *History of IEEE*. URL: https://www.ieee.org/about/ieee-history.html?utm_source=wdw&utm_medium=lp-about&utm_campaign=history (visited on 07/15/2024).

[18] "IEEE Draft Standard for Time-Sensitive Networking Profile for Automotive In-Vehicle Ethernet Communications". In: *IEEE P802.1DG/D4.0, July 2024* (2024).

[19] "IEEE Standards for Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks-Specific requirements-Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications". In: *IEEE Std 802.3, 1998 Edition* (1998).

[20] "IEEE Standards for Local and Metropolitan Area Networks: Supplement - Media Access Control (MAC) Parameters, Physical Layer, Medium Attachment Units, and Repeater for 100Mb/s Operation, Type 100BASE-T (Clauses 21-30)". In: *IEEE Std 802.3u-1995 (Supplement to ISO/IEC 8802-3: 1993; ANSI/IEEE Std 802.3, 1993 Edition)* (1995).

[21] Kersting Kristian, Lampert Christoph, and Rothkopf Constantin. *Wie Maschinen lernen*. Springer Wiesbaden, 2019.

[22] Aleksandar Lazarevic, Vipin Kumar, and Jaideep Srivastava. "Managing Cyber Threats: Issues, Approaches, and Challenges". In: Springer US, 2005. Chap. Intrusion Detection: A Survey.

[23] Gianfagna Leonida and Di Cecco Antonio. *Explainable AI with Python*. Springer Cham, 2021.

[24] Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. “Explainable AI: A Review of Machine Learning Interpretability Methods”. In: *Entropy* (2021).

[25] Google LLC. *PyCoral API overview*. 2020. URL: https://coral.ai/docs/reference/py/ (visited on 08/28/2024).

[26] Feng Luo et al. “A Multi-Layer Intrusion Detection System for SOME/IP-Based In-Vehicle Network”. In: *Sensors* (2023).

[27] Feng Luo et al. *Dataset*. URL: https://github.com/yzyGo/Dataset-for-SOME-IP-IDS/tree/master (visited on 08/10/2024).

[28] Mohamed Mahmoud et al. “AE-LSTM: Autoencoder with LSTM-Based Intrusion Detection in IoT”. In: *2022 International Telecommunications Conference (ITC-Egypt)*. 2022.

[29] Kirsten Matheus and Thomas Königseder. “Automotive Ethernet”. In: Cambridge University Press, 2021. Chap. A Brief History of In-Vehicle Networking.

[30] Kirsten Matheus and Thomas Königseder. “Automotive Ethernet”. In: Cambridge University Press, 2021. Chap. A Brief History of Automotive Ethernet.

[31] Earum Mushtaq et al. “A two-stage intrusion detection system with auto-encoder and LSTMs”. In: *Applied Soft Computing* (2022).

[32] Fernando Nogueira. *Bayesian Optimization: Open source constrained global optimization tool for Python*. 2014. URL: https://github.com/bayesian-optimization/BayesianOptimization.

[33] Kalmar Ralf and Oliveira Antonino Dr.-Ing. Pablo. *Warum Software-defined Vehicles an Continuous Engineering in DevOps nicht vorbeikommen*. Apr. 6, 2023. URL: https://www.iese.fraunhofer.de/blog/continuous-engineering-in-devops/ (visited on 06/14/2024).

[34] Stuart Russell and Peter Norvig. *Artificial Intelligence: a Modern Approach*. Pearson Education, Limited, 2021.

[35] Ovidiu Vermesan et al. “Automotive Intelligence Embedded in Electric Connected Autonomous and Shared Vehicles Technology for Sustainable Green Mobility”. In: *Frontiers in Future Transportation* (2021).

[36] Daniel Zelle et al. “Analyzing and Securing SOME/IP Automotive Services with Formal and Practical Methods”. In: *Proceedings of the 16th International Conference on Availability, Reliability and Security*. 2021.