Journal of Systems Architecture 156 (2024) 103283

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

A convolutional autoencoder architecture for robust network intrusion detection in embedded systems

Niccolò Borgioli [a],*, Federico Aromolo [a], Linh Thi Xuan Phan [b], Giorgio Buttazzo [a]

[a] *Scuola Superiore Sant'Anna, Pisa, Italy*
[b] *University of Pennsylvania, Philadelphia, USA*

ARTICLE INFO

Keywords:
Intrusion detection
Artificial neural networks
Autoencoder
Unsupervised learning
Poisoning robustness
Explainable AI

ABSTRACT

Security threats are becoming an increasingly relevant concern in cyber–physical systems. Cyber attacks on these systems are not only common today but also increasingly sophisticated and constantly evolving. One way to secure the system against such threats is by using intrusion detection systems (IDSs) to detect suspicious or abnormal activities characteristic of potential attacks. State-of-the-art IDSs exploit both signature-based and anomaly-based strategies to detect network threats. However, existing solutions mainly focus on the analysis of statically defined features of the traffic flow, making them potentially less effective against new attacks that cannot be properly captured by analyzing such features. This paper presents an anomaly-based IDS approach that leverages unsupervised neural models to learn the expected network traffic, enabling the detection of unknown novel attacks (as well as previously-known ones). The proposed solution uses an autoencoder to reconstruct the received packets and detect malicious packets based on the reconstruction error. A careful optimization of the model architecture allowed improving detection accuracy while reducing detection time. The proposed solution has been implemented on a real embedded platform, showing that it can support modern high-performance communication interfaces, while significantly outperforming existing approaches in both detection accuracy, inference time, generalization capability, and robustness to poisoning (which is commonly ignored by state-of-the-art IDSs). Finally, a novel mechanism has been developed to explain the detection performed by the proposed IDS through an analysis of the reconstruction error.

1. Introduction

As the reliance on networked systems in cyber–physical systems (CPS) continues to increase, so does the number of cyberattacks targeting these systems. To ensure that these systems are safe, it is important to be able to detect and respond to such attacks in real time. One way to accomplish this is by using intrusion detection systems (IDSs), which can identify network anomalies that may indicate the presence of an attack.

Modern network intrusion detection techniques typically use signature-based and anomaly-based strategies. The former rely on predefined rules or signatures for attack detection, which is efficient and works well for known attacks, but can easily be bypassed by attackers who use new or modified attack methods. The latter are often slower but better suited for detecting new attacks, since they do not require any prior knowledge of the malicious traffic. Existing work on this front primarily focuses on analyzing a predefined set of statistical features extracted from the network packet flow [1–10]. By using a fixed number of features, usually between 30 and 100, one can reduce the computational complexity of the problem and thus increase detection speed. However, the main drawback of this approach is that an attacker could craft a specifically tailored adversarial packet taking into account these features to bypass detection [11].

In order to improve the detection performance of network IDS, many techniques based on machine learning (ML) have been proposed in recent years, leveraging different kinds of artificial neural networks [1,2]. Such techniques follow either the supervised or unsupervised learning paradigm. With supervised learning, a neural network is trained to recognize malicious traffic using a labeled dataset. Supervised learning requires labeling each data sample used to train the neural network, which typically contains millions of packets. Labeling datasets of this size is an expensive and time-consuming process. Furthermore, techniques based on supervised learning are susceptible to new categories of attacks that were not covered in the dataset used to train the network; as a result, they require frequent retraining to ensure effective detection performance over time. In contrast, with unsupervised learning, the neural network is trained to recognize the

* Corresponding author.
E-mail address: borgioli.niccolo@gmail.com (N. Borgioli).

https://doi.org/10.1016/j.sysarc.2024.103283
Received 1 February 2024; Received in revised form 14 August 2024; Accepted 20 September 2024
Available online 28 September 2024

expected behavior of the network traffic and detect deviations from such a behavior, without the need for a labeled dataset.

In this paper, unsupervised learning is exploited to develop a new generation of IDSs that are capable of detecting general and new kinds of network attacks (including zero days) without using fixed features. The idea is to train a 1D convolutional autoencoder to reconstruct packets coming from normal network traffic; then, at runtime, the autoencoder is used to reconstruct the received packets and detect anomalies and malicious packets based on the reconstruction error. Unlike existing work, our approach uses the full packet as input, thus allowing for a more in-depth analysis that makes it more difficult for an attacker to avoid detection. Moreover, our solution not only improves the detection accuracy, but also reduces the detection time. Hence, while most of the existing works mainly focus on deploying the IDS on high-performance servers, our solution can effectively be deployed on embedded edge devices in the context of cyber–physical applications.

To evaluate the effectiveness of the proposed architecture in resource-constrained CPS, we conducted an extensive experimental evaluation on real embedded devices using a dataset of real-world network traffic. The results demonstrate that the proposed solution can reliably detect new attacks with high accuracy, outperforming existing methods [9,12] in detecting a variety of common attack types. The capability of the proposed approach to detect network attacks in an unsupervised manner, without the need for labeled data, makes it a valuable tool for safeguarding CPS against security threats.

A challenge in designing an IDS for cyber–physical systems is real-time detection: not only the system should be able to accurately detect a broad range of attacks, but it should do so in a timely fashion. As a first step towards this goal, this paper presents extensive experiments evaluating the timing properties of the proposed system on resource-constrained embedded devices commonly used in CPS, both with and without GPU acceleration. Such an understanding provides useful insight for optimizing the real-time performance of the IDS.

This work also presents a way to improve inference throughput by performing the inference of multiple packets in parallel (via packet batching). To optimize the performance of the networking stack, modern network interfaces implement the New API (NAPI) mechanism, which is intended to reduce the overhead of packet receiving. The idea of such a mechanism is to defer incoming message handling until a sufficient number of packets is received, so that they can all be processed at once, thus reducing the overhead due to interrupts. This work leverages such a mechanism to perform inference on batches of packets and studies the effect of batch size on the achieved throughput and latency. The experimental results show that using batching during inference can substantially increase the bandwidth supported by the proposed IDS up to 1 Gbps with a limited increase in the introduced latency.

Although machine learning models can learn to recognize patterns and make predictions based on large amounts of data, they can be vulnerable to attacks by malicious actors who introduce poisoned data into the training dataset. This can cause the network to learn incorrect or biased patterns, leading to inaccurate predictions that could be exploited by the attacker. Therefore, it is important for a learning-based IDS to be robust against dataset poisoning to ensure the reliability and safety of their predictions in real-world applications. Unfortunately, this is not adequately accomplished in many existing solutions: for instance, Nkashama et al. [13] showed that contaminating just 5% of the training set can disrupt the detection capabilities of several existing IDS approaches, causing a very significant loss in accuracy. On the contrary, this work addresses such an issue by introducing random noise in the input during the training phase. Thanks to this countermeasure, the network not only better learns the features of the packets (thus enhancing its detection capability), but also greatly improves its robustness to dataset poisoning, making the proposed approach almost invulnerable to this kind of attack. This also broadens the practical applicability of the proposed models, as they can be trained directly in the actual deployment setting without a need for a fully controlled training environment.

Explaining the decisions taken by machine learning models is still an open challenge. Thus, explainable AI (XAI) methods have been investigated in recent years to enhance the transparency and interpretability of machine learning systems. In this work, we introduce a novel heatmap-based mechanism exploiting the reconstruction error to help visualizing why a specific packet has been classified either as normal or malicious. Using this methodology is not only useful to validate the model correctness, but can also give useful insights about how a novel attack might work.

Contributions. In summary, the paper makes the following novel contributions:

- It proposes a method for detecting network attacks based on unsupervised learning;
- It evaluates the accuracy and inference time of the proposed model on an implementation for a real edge platform under different realistic configurations;
- It compares the performance of the proposed model with state-of-the-art IDSs;
- It evaluates the robustness of the proposed technique to dataset poisoning; and
- It proposes an explainability method to understand the detection decisions.

Paper organization. Section 2 provides an overview of the related work and the datasets considered in the paper; Section 3 presents the proposed IDS method; Section 4 presents the experiments performed to assess the accuracy, timing performance, poisoning robustness and explainability of the proposed system, and the comparison with state-of-the-art methods; Section 5 provides a discussion of the experimental results; and, finally, Section 6 presents some concluding remarks and directions for future work.

2. Related work

This section begins with an overview of existing datasets, including the one selected for the design and evaluation of the proposed approach, and then discusses the state-of-the-art of related IDS techniques based on neural networks.

2.1. Datasets

Towards a general IDS that is capable of detecting novel attacks in real time, we evaluated several existing datasets for both training and testing our models. Since our goal is to develop a packet-based IDS to be deployed on each edge device of a network, we targeted a dataset that provides raw network packet captures (pcap files) collected from different network nodes. In the proposed approach, unsupervised learning is used to train the network using benign packets only so that it can learn to detect anomalies in the presence of malicious packets. We considered the following existing datasets, which cover a range of representative use case scenarios for different environments and are widely used in literature to assess IDSs performances: NSL-KDD [24], UNSW-NB15 [25], CIC-IDS2017 [26], X-IIoTID [27], TON_IoT [28], and EDGE-IIOTSET [29].

NSL-KDD. This dataset is a rebalanced version of the widely used KDDCUP'99 [30] dataset (which presented some statistical issues). It contains about one million single connection vectors, each of which described by 41 features and labeled as either normal or malicious. However, this dataset presents some limitations which prevents its adoption in the present work: **(i)** it does not provide raw packets, but just a subset of features describing a connection (a set of packets), and **(ii)** the data contained in the dataset were generated more than two

Table 1
A comparison of the proposed approach with state-of-the-art anomaly-based IDSs. The "Timing performance" column reports whether the method was assessed with respect to inference time and, if so, shows the order of magnitude of the packet processing latency. The "Poisoning robustness" and "Explainability" columns indicate whether the method was assessed with respect to dataset poisoning and explainability, respectively.

Work	Method	Input type	Learning paradigm	Accuracy	Timing performance	Poisoning robustness	Explainability
Gao et al. [5]	DBN	Flows features	Supervised	93%	No	No	No
Alom et al. [14]	DBN	Flows features	Supervised	97%	No	No	No
Yousefi-Azar et al. [6]	Autoencoder	Flows features	Unsupervised	83%	No	No	No
Vaiyapuri and Binbusayyis [15]	Autoencoder	Flows features	Unsupervised	92%	No	No	No
Truong-Huu et al. [16]	GAN autoencoder	Flows features	Unsupervised	82%	No	No	No
Kwon et al. [17]	CNN vs. LSTM	Flows features	Supervised	67%	No	No	No
Malaiya et al. [12]	LSTM	Flows features	Supervised	99%	No	No	No
Kathareios et al. [7]	Multistage AE	Flows features	Supervised	90%–98%	seconds	No	No
Andreas et al. [18]	BLSTM	Flows features vs. packets features	Supervised	76%–96%	No	No	No
Dromard et al. [8]	Algorithm	Flows features	Supervised	93%	No	No	No
Alam et al. [19]	Autoencoder	Flows features	Unsupervised	92%–95%	microseconds	No	No
Carrera et al. [20]	Mix	Flows features	Unsupervised	80%–90%	milliseconds-seconds	No	No
Mirsky et al. [9]	Ensemble autoencoders	Flows features	Unsupervised	24%–98%	milliseconds	No	No
Tekiner et al. [10]	Various classifiers	Flows features	Supervised	99%[a]	No	No	No
King et al. [21]	GNN	Flows features	Unsupervised	97%[a]	seconds	No	No
Roy et al. [22]	DNN	Flows features	Supervised	99%	No	No	Yes
Mane et al. [23]	DNN	Flows features	Supervised	96%	No	No	Yes
This work	**LSTM and 1D CNN AE**	**Full packets**	**Unsupervised**	**99%**	**milliseconds**	**Yes**	**Yes**

[a] The corresponding accuracy value is not referred to a wide range of attack classes, but only to the specific one considered in the work.

decades ago and are no longer representative of modern network traffic and attacks.

UNSW-NB15. This dataset contains about 100 GB of collected raw benign and malicious packet data generated in a controlled simulation environment. It provides both a labeled CSV file with the extracted flow features and the raw pcap files recorded during the simulation, along with a report on each performed attack. However, this dataset provides only flow labels but not per-packet labels, making it unsuitable for our purposes.

CIC-IDS2017. This dataset contains raw packet captures of both benign and malicious packets collected in a controlled network over 5 days. The dataset is provided both in the form of a CSV file with labeled flow information and both as raw pcap files with indications of when attacks start and end for each day. However, this dataset also does not provide per-packet labeling, making it unsuitable for our purposes. Recent work also discovered several flaws affecting the CIC-IDS2017 dataset in the labeling of packets and in the traffic capture, including errors such as duplicate packets and incorrect labeling [31].

X-IIoTID. This dataset contains extracted features related to network traffic, system and application logs, and system resource usage (e.g., CPU and memory usage). However, this dataset does not provide raw packet data, which is necessary to perform training and evaluation of the proposed approach, which is based on leveraging full packet information.

TON_IoT. This dataset includes data sources of different types including IoT services telemetry, operating system logs, and network traffic captures. The network traffic captures include both normal and malicious traffic for nine different classes of attack. This dataset provides both the raw pcap files with the recorded network traffic and the csv files with selected features based on correlation matrix analysis.

EDGE-IIOTSET. This dataset collects network traffic of a realistic network composed of more than 10 different types of IoT devices. The authors collected the traffic generated by each device in a separate pcap file and performed 14 different types of attacks. The dataset provides both the raw pcap files and the CSV files with the extracted features from each packet, along with its label.

Based on the observed characteristics of each of the considered datasets, we decided to adopt the EDGE-IIOTSET for the present work, not only because it is the only one that provides labeled raw packets, but also because it considers the widest range of device types (and thus protocols) along with one of the most recent set of attacks.

2.2. Related work on anomaly-based network IDS

Network IDSs relying on anomaly detection are responsible for monitoring the network traffic for suspicious and malicious packets. Unlike traditional signature-based IDSs, network anomaly detection techniques provide an effective way to detect new types of attacks for which a defining signature is not yet available for pattern matching. These techniques typically leverage machine learning (ML) to classify each packet as either normal or suspicious based on a set of relevant features characterizing both the packet itself and the flow of data to which the packet belongs [1,2]. Their classification performance strongly depends on the quality of the feature selection procedure carried out on the training set [3,4].

Earlier work on network anomaly detection relies on classification algorithms such as support vector machines, decision trees, genetic algorithms, and k-nearest neighbors (k-NN) algorithms [32]. With the advent of deep learning, research efforts in network intrusion detection have shifted towards applying neural architectures to the classification problem, given their proven effectiveness in general anomaly detection tasks [1,33]. Common network IDS architectures include Convolutional Neural Networks (CNNs), recurrent neural networks (RNNs), Long Short-Term Memory Networks (LSTMs), and deep belief networks (DBNs), often deployed in autoencoder topologies to enable unsupervised learning of packet flow features. Systematic studies on deep learning methods for network anomaly detection have also been carried out, for example, by Ahmad et al. [1] and Lansky et al. [2].

Table 1 gives an overview of the related work on ML-based network IDSs. For each considered work, the table reports the detection technique, the type of input data provided to the IDS (either flow-based features or full packets), the adopted ML paradigm (supervised or unsupervised), the achieved detection accuracy, the order of magnitude of the packet processing latency (if an evaluation of timing performance was provided), and whether poisoning robustness and explainability concerns were considered. For comparison, the table also provides a similar summary for the present work, highlighting how it is positioned with respect to the state of the art. In the following, more details are provided on these related approaches.

2.2.1. Input and learning paradigm

Gao et al. [5] proposed one of the first network IDS architectures leveraging deep learning. They applied a DBN to solve the classification problem, achieving an accuracy of 93%. However, the training was performed on the KDDCUP'99 packet flow dataset, which presents some statistical issues and is now considered mostly obsolete [29]. Later, Alom et al. [14] proposed a similar DBN architecture, trained on a more recent dataset, and reported an accuracy of 97%.

With the aim of improving the detection accuracy on new types of attacks, Yousefi-Azar et al. [6] adopted an autoencoder architecture trained in an unsupervised fashion to generate a reconstruction of the input traffic flows and discriminate between normal and anomalous traffic based on the reconstruction error. As the approach is unsupervised, it does not require labeled data or feature selection; however, the accuracy attained by the method in the related experiments only reached 83%. Vaiyapuri and Binbusayyis [15] utilized a convolutional autoencoder architecture which leverages flow features to perform classification, achieving accuracy levels of up to approximately 92%. Truong-Huu et al. [16] adopted an unsupervised deep learning approach that leverages generative adversarial networks (GANs) to reconstruct network packets, and achieved a similar accuracy level (82%).

Kwon et al. [17] compared the experimental performance of CNN- and LSTM-based approaches for network anomaly detection, showing that increasing the number of layers in a CNN topology beyond a certain depth does not yield an accuracy improvement, and that LSTM approaches typically perform better. Malaiya et al. [12] further explored the performance of LSTM techniques for feature-based classification, and achieved 99% accuracy. The relative performance between CNN- and LSTM-based approaches observed in [17] is not generalizable, however. More recently, Borgioli et al. [34] showed that, for unsupervised learning anomaly detection, LSTM-based approaches outperform CNN-based approaches in terms of accuracy, but are much slower in terms of timing performance. In the present paper, the proposed convolutional autoencoder model was carefully optimized to achieve good timing performance and high detection accuracy with respect to LSTM-based strategies.

More recently, Andreas et al. [18] explored the design and accuracy trade-offs between the packet-based and the flow-based anomaly detection techniques, with reference to a Bidirectional LSTM (BLSTM) classifier based on either per-packet or flow-related features. In their experiments, the BLSTM classifier only reached 76% accuracy when considering per-packet features, while it reached an accuracy of 96% with flow-based features. Dromard et al. [8] proposed a custom algorithm to extract features from flows to recognize different anomaly classes based on some signatures, and obtained a classification accuracy of 93%.

Numerous network anomaly detection solutions focus on analyzing the network traffic of a whole network by using a dedicated machine that receives a copy of the whole network traffic from the router. For example, Tekiner et al. [10] proposed a mechanism to detect cryptojacking attacks targeting IoT devices by using statistical features about sequences of 10 packets of the network and testing those features with different kinds of classifiers. Similarly, King et al. [21] developed a framework to detect lateral movements using Graph Neural Networks to encode the topological features of the network traffic before feeding them to an RNN.

In general, prior work primarily relies on analyzing high-level statistical features that describe the network traffic for detection. In contrast, our work focuses on network anomaly detection by analyzing the full contents of each packet. As such, our proposed method requires neither feature selection during training nor feature extraction at runtime. While there exist some prior solutions [16,18] that investigated how to perform detection using the whole packet, they did not achieve the same level of accuracy as that of the proposed approach.

2.2.2. Timing performance

When deploying an IDS in a real environment, especially in CPS settings, we should not only evaluate the detection capabilities but also the time required to perform such detection and its implications.

Kathareios et al. [7] developed a multi-stage solution for anomaly detection and also evaluated the inference time in their experiments. Their method considers a preprocessing phase to extract features concerning the flow of packets and the overall network traffic. The extracted features are then used as input to an autoencoder, which attempts to reconstruct the features as closely as possible. Finally, the reconstruction error of the features produced by the autoencoder is used to detect anomalous flows. To contain the amount of false positives (i.e., normal packets misclassified as malicious packets), a second detection stage consisting in a classifier was adopted to further analyze suspicious flows. The proposed method achieved a true positive rate between 90% and 98%, with a 2% rate of false positives. The authors also evaluated the inference time of their method, which requires at least one second per prediction.

Alam et al. [19] proposed a memristor-based autoencoder to perform anomaly detection in low-power devices. They first trained the autoencoder offline and reached a detection accuracy of 95%. Then, they converted it for execution in the memristor on the low-power platform and the resulting accuracy was decreased to 92%. They also analyzed the impact of the proposed solution in terms of power consumption and inference time under different working configurations.

Carrera et al. [20] proposed to combine multiple unsupervised approaches to perform anomaly detection. Evaluations showed that their proposed solutions achieve an accuracy ranging from 80% to 90% depending on the different model combinations, and an inference time of up to 1.18 s.

Mirsky et al. [9] proposed a network IDS technique, named Kitsune, which leverages an ensemble of shallow autoencoders and an unsupervised learning approach for flow-based anomaly detection. Each autoencoder in the ensemble analyzes a different set of features to characterize the incoming traffic, and is trained to reconstruct such features for normal packet streams with low error. Given the reconstruction errors of each autoencoder, an additional shallow autoencoder serves as a voting mechanism to determine an overall reconstruction error for the packet flow. If the resulting error surpasses a given threshold, an anomalous behavior is reported by the IDS. The features are related to summary statistics aggregated over packet flows identified by source MAC address, and source and destination IP address, TCP socket or UDP socket. A hierarchical clustering algorithm is adopted to determine a suitable mapping between each flow feature and one of the autoencoders in the ensemble. Kitsune's detection accuracy, however, varies widely (between 24% and 98%).

The main problem of these solutions is that they either showed very high inference times (making them impractical on modern embedded devices) or used high-level statistical network traffic features [7,20,21]. Our work instead presents an analysis of the expected inference time on a real edge computing platform both with and without GPU acceleration. The achieved results showed that our system can satisfy the typical timing requirements of the cyber–physical systems while analyzing the full packet features.

2.2.3. Dataset poisoning

Dataset poisoning is a well-known issue that can be exploited by adversaries to make any kind of classifier produce false negatives. Even though such a problem has been known in the literature for decades [35,36], state-of-the-art IDS are generally not robust against data contamination. Nkashama et al. [13] investigated this issue for six modern deep learning algorithms for intrusion detection and showed that even a small injection of malicious packets (5%) among the benign ones used for training could destroy the detection capabilities of the considered solutions. To the best of our knowledge, none of the state-of-the-art solutions consider dataset poisoning in their evaluation. The experiments presented in this work also consider the effect of dataset poisoning, and show that the proposed approach is robust against this kind of threat.

2.2.4. Explainability

Several studies have been conducted to explore the application of explainable AI techniques for IDS. On this front, Roy et al. [22] proposed a knowledge graph with ontologies to explain the detected anomalies and applied the proposed approach to the CIC-IDS2017 dataset. Similarly, Mane et al. [23] combined several explainability techniques (such as LIME, and SHAP) to generate both local and global explanations with respect to the features of the NSL KDD dataset. However, these approaches only provide attempts at the explanation for the high-level features of flow-based datasets [37]. In contrast, our explainability method considers the contents of the full packet and allows to identify which specific parts of the packet are considered malicious. Overall, as summarized in Table 1, to the best of our knowledge, this work is the first to perform network intrusion detection by analyzing full packet data in an unsupervised learning approach with very high detection accuracy (99%) while also considering timing performance, dataset poisoning and detection explainability.

3. The proposed real-time network IDS method

This section presents the proposed detection method. Before describing the model architecture – the core component of our detection method – we first discuss the considered scenario and the data preprocessing that guided our model design and training.

3.1. Use cases

In this paper, we considered two possible use case scenarios: an automotive setting and a network of sensors. In the first case, we considered the new generation of vehicles, composed of a set of devices interconnected by an Automotive Ethernet infrastructure. In the second case, we considered a set of sensors (and actuators) interconnected through a wireless infrastructure. In both scenarios, each of such devices (in particular if performing critical tasks) is also equipped with our IDS to protect it from malicious attacks.

In this context, the IDS is fine-tuned directly on the machine on which it is to be deployed using network traffic acquired live from the real network.

3.2. Threat model

The adversary considered in this work possesses advanced technical knowledge and skills, has access to the network traffic and structure, and can perform attacks from multiple locations in the network, posing a significant risk to the system's security. The attackers also know the neural network adopted in the IDS and the set of input features received by the neural network to perform the detection. Therefore, the adversary can properly select attacks that are more difficult to detect when considering the restricted set of features provided as input to the neural network.

Additionally, unlike most of the state-of-the-art approaches, we assume that the adversary could partially manipulate the dataset used to train the IDS by injecting some malicious packets. The ability to be resilient to attacks by such a powerful adversary is crucial, as it allows to perform training in a realistic deployment setting, where it is not possible to have full control over the network traffic.

3.3. Data preprocessing

As discussed above, our work uses the EDGE-IIOTSET dataset as it covers a broad range of device types and the most recent set of attacks.

Since packets are provided as byte sequences, before feeding them to the classifier, we must encode them in an interpretable format. Because the packet length is highly variable and we want the model to learn the important features of each packet, we encode each byte as a floating-point value from 0 to 1, by dividing the 8-bit unsigned integer representation of each byte by 255. Moreover, the 1D CNN needs the input tensor to be of fixed size, so we added some padding at the end of each packet to make it fit the maximum packet size, i.e., 1514 bytes, based on the standard Maximum Transmission Unit (MTU) for Ethernet. Fig. 1 shows how the input tensor is constructed starting from an input packet.

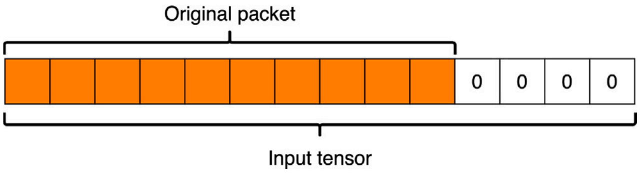


Fig. 1. Figure showing how a sample input tensor to the network is constructed starting from an input packet. The original packet (in orange) is padded to reach the size of the desired input tensor to the network.

3.4. Model architecture

Autoencoders are a class of unsupervised neural networks composed of two main components: an encoder and a decoder. The encoder reduces the dimensionality of the incoming data and produces a compressed representation (encoding) of the input. The decoder performs the opposite operation by reconstructing the original data with little or no error starting from the encoding. This way, the autoencoder learns how to automatically extract relevant features from the input and how to use such features to reconstruct the original input.

Key idea. Traditional approaches make use of binary classifiers trained with labeled benign and malicious packets; however, their main drawback is the low accuracy in the classification of new attacks, thus requiring continuous retraining of the network to keep it up to date. Our approach aims at overcoming this limitation by *training the autoencoder with only benign packets*. In this way, the autoencoder will have low reconstruction error on benign packets and high error on malicious (anomalous) ones. Thus, by analyzing the reconstruction loss of the packets being analyzed by the system, it is possible to detect malicious packets (including zero-day attacks) without the need for retraining. Specifically, if X is the input sequence of length N and Y is the reconstructed sequence, the reconstruction loss is defined as:

$$loss(X, Y) = \sum_{i=1}^{N} |x_i - y_i|, \tag{1}$$

where x_i and y_i are the ith elements of the X and Y sequences, respectively. Since in the considered dataset all packets are unencrypted, we fed the whole Ethernet frame (X) to the neural network.

In this work, we developed a 1D Convolutional Neural Network Autoencoder (1D CNN AE). 1D CNNs are widely used when dealing with sequences and time series, due to their advantages over conventional (2D) CNNs. In this kind of networks, the convolutional filter slides along a single dimension to produce an output. Compared to traditional fully connected autoencoders, a 1D CNN reduces the number of parameters and thus the memory footprint and computational complexity of the network while improving its accuracy.

A convolutional autoencoder is composed of many convolutional (or transposed-convolutional) and pooling (or unpooling) layers stacked one on top of the other (see Fig. 2). To guarantee a correct reconstruction, the kernels and channel sizes of the decoder stage are symmetric to those of the encoder. When designing a 1D-CNN autoencoder we must *carefully select the hyperparameters* (i.e., kernel sizes, stride, padding, etc.) to ensure that the reconstructed packet has the same size as that of the input. Moreover, we must also ensure that the resulting

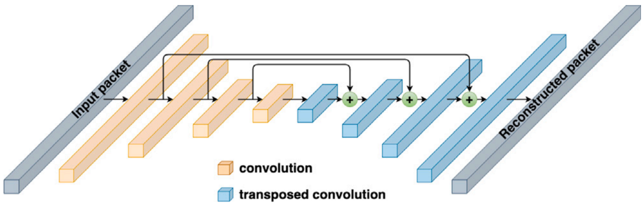


Fig. 2. Structure of the proposed 1D CNN Autoencoder.

Table 2
IEEE 802.3 Ethernet frame structure.

Name	MAC Dst	MAC Src	Ethertype	Payload	CRC
Length (Bytes)	6	6	2	46–1500	4

encoding compresses enough input information to allow the network to generalize, as well as to correctly reconstruct the packets.

To make the output size equal to the input size, we first need to compute the output size of each convolutional (and transposed convolutional) layer.

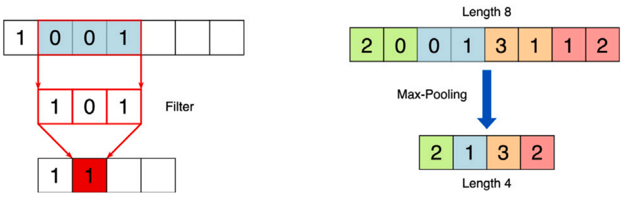
$$L_{out} = \left\lfloor \frac{L_{in} + 2 \cdot pad - dil \cdot (kernel\_size - 1) - 1}{stride} + 1 \right\rfloor \tag{2}$$

Eq. (2) provides the output size L_{out} of a convolutional layer where L_{in} is the input size, pad is the padding size, dil the dilation, $stride$ the stride, and $kernel\_size$ the kernel field size. Repeating this process for each layer we can make sure that the input and output size of the network satisfy the requirement.

By analyzing the Ethernet frame structure (Table 2), we observe that each byte typically carries little or no useful information on its own. Since most of the information is typically grouped into 6–8 consecutive bytes, we start by selecting a kernel size for the first convolutional layer of at least 8. In machine learning, Receptive Field (RF) is defined as the size of the region in the input that produces the feature. For the first layer, the RF is given by the kernel size and should be close to the size of the low-level features that we want such a layer to learn to extract, so we started exploring the kernel size of the first layer in the vicinity of 8 and we found that the optimal performance can be achieved when the kernel size of the first layer is 7. This means that each group of 7 consecutive bytes will be encoded in one byte (see Fig. 3(a)). Similar reasoning is repeated also for subsequent layers that, instead, are in charge of detecting higher-level features based on the ones extracted by the first layer. This allows subsequent convolutional layers to gradually reduce the kernel size to gradually extract higher-level features. To achieve a high compression of the original input, 1D CNN autoencoders typically use pooling layers (see Fig. 3(b)) to downsample the data. However, from the mathematical point of view, a pooling (or unpooling) layer can be replaced by a convolutional (or transposed convolution) layer with stride equal to the pooling size [38]. Thus, since one convolutional layer followed by a pooling layer can be replaced by a convolutional layer with stride, we can reduce the number of operations to be performed and improve the inference time.

The decoder part of the autoencoder should ideally perform the opposite operation of the encoder. Based on this assumption we designed the decoder part using the same kernel size and number of channels of the corresponding layer in the encoder part. Furthermore, the architecture utilizes the concept of *skip connections* (or *residual connections*) across the autoencoder bottleneck. These additional connections forward the feature maps from a layer in the encoder to a later layer in the decoder, with the aim of facilitating packet reconstruction. Skip connections have been utilized in numerous neural architectures in various domains, e.g., in the well-known U-Net architecture and its variants [39–41].

Based on the above observations, we designed from scratch a novel ANN architecture that can achieve both high detection accuracy and low computational overhead. Achieving such objectives required a careful selection of the architecture and its hyper-parameters. This was achieved with an extensive design space exploration of the possible number of layers and their size, guided by the idea of minimizing the size of the network.



(a) 1D Convolutional layer (b) 1D Pooling layer

Fig. 3. Diagram showing the behavior of the 1D Convolution and the 1D Pooling operations, applied to a sample input vector.

3.5. Training

The proposed approach relies on training the autoencoders with only benign packets to enhance the generalization capability against novel malicious attacks. To do so, we divided the available benign packets into three groups: training set, validation set, and test set.

During the training phase, the network was fed with only benign samples and the autoencoder weights were updated to learn how to extract the relevant features needed for reconstructing such samples. More specifically, in the training phase, the model optimizer adopted the reconstruction loss shown in Eq. (1) to update the network weights. At the end of each training epoch, the network performance was evaluated by measuring its reconstruction loss on the validation set. This is a key step for evaluating whether the network is learning to generalize rather than simply memorizing the training samples.

Recent research has shown the importance of adding random noise to the inputs, in the training phase, to both improve the capabilities of the neural network to learn important features [42] and to improve the model robustness to dataset poisoning [43]. Based on these findings, a Gaussian perturbation was introduced in all training packets to help the network learn relevant features rather than an identity function. This random perturbation is applied as a random change in the float-encoded value of each byte of the packet by adding/subtracting a float value between 0 and 1. The effect of this perturbation on the original packet is a multiple-bit random flip.

In the training phase, a hyperparameter exploration (kernel size, number of layers, amount of noise, etc.) was also carried out to improve the reconstruction capability of the network on benign packets.

The anomalous packets are detected based on a threshold on the loss value. Therefore, the distribution of the reconstruction loss was analyzed on the trained model in order to tune such a threshold. In general, a small threshold value makes the detection more sensitive to malicious packets, but it can also misclassify some sporadic benign packet. On the contrary, a higher threshold value reduces the misclassification of benign packets, but at the same time it reduces the sensitivity of the detection of malicious packets. In a real-world deployment, the threshold should be tuned based on the analysis of the reconstruction loss distribution based on the specific application requirements. In this work, we followed a fully-unsupervised approach, selecting the threshold based on the 99th percentile of the loss on normal packets used during training.

4. Implementation and evaluation

Prototype. The proposed 1D CNN autoencoder architecture has been implemented and trained using the PyTorch framework on an Nvidia

Table 3
Per-packet detection capabilities of proposed network compared with the Kitsune solution.

Architecture	TP	TPR	FP	FPR	F1-score
1D CNN AE	9,863,877	**99.97%**	1144	0.10%	**0.9997**
Kitsune IDS	3,982,940	40.37%	1143	0.10%	0.5751

DGX server. To evaluate the performance of the model in a real-world setting, the network has also been implemented in C++ and tested on an NVIDIA Jetson AGX Orin Developer Kit using the Libtorch library, setting the weights with the values obtained from the training phase, and in a specialized GPU-accelerated implementation using the NVIDIA TensorRT framework for optimized inference performance.

Evaluation. The objective of the experimental study carried out on the models is twofold and is aimed at evaluating (i) the detection accuracy of malicious packets, including those of novel unseen attacks, also in the presence of dataset poisoning; and (ii) the inference time on embedded devices that are typically employed in cyber–physical applications. More specifically, the experimental results report: **(1)** the detection accuracy of the proposed network architecture; **(2)** the inference time of the model on a real embedded platform; **(3)** the effect of batching on latency and throughput performance; and **(4)** the robustness of the system against dataset poisoning. Finally, a heatmap-based mechanism is presented and discussed to help the user interpret the decisions made by the model.

4.1. Detection accuracy

The trained network was tested with the threshold value obtained in the tuning phase. Tests were performed using all malicious packets of each attack type to assess the generalization capability of the network against new attack types.

The model has been evaluated using the ground truth labels of the samples to measure conventional performance metrics, such as the numbers of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). Here, TP represents the number of malicious packets that are correctly detected, and FP represents the number of normal packets that are (falsely) classified as malicious. TN (FN) can be defined similarly, but for normal (malicious) packets that are classified as normal packets.

Since the different classes of malicious packets and the benign packets contain different numbers of samples, the True Positive Rate (TPR), the False Positive Rate (FPR), and the F1-score have also been computed, as follows:

$$TPR = \frac{TP}{TP+FN}, \qquad FPR = \frac{FP}{TN+FP}, \qquad F1 = \frac{2 \cdot TP}{2 \cdot TP+FP+FN}$$

Results. Table 3 reports the performance of the proposed architecture in terms of TP, TPR, FP, FPR, and F1-score over all the malicious packet classes. The selected dataset contains 9,865,868 malicious packets and 1,143,948 normal packets. Results show that the proposed network has excellent detection capabilities, with an F1-score very close to 1 (the optimal score).

Table 4 shows the performance of the proposed network under different classes of attacks that the network was not trained on.

The results show that the proposed 1D CNN AE achieved an overall detection accuracy close to 100%. With the ability to consider the full packet at the same time, the 1D CNN could easily extract packet-wide features; as a result, it even detected the attack classes that are more similar to normal packets.

4.2. Importance of skip-connections

To better understand the implications of using the skip connections in our autoencoder architecture we compared our detection performances with and without such architectural feature. To do so, we tested the proposed autoencoder architecture with and without adding such connections using the same hyperparameters for training. Then, we evaluated the two solutions using the same metrics presented in Section 4.1.

Results. Table 5 presents the results of such a comparison. We can observe that the introduction of the skip-connections further improves the network performance by 0.1% while maintaining the same FPR (0.10%). Although this improvement may appear small, in cybersecurity, every single percentage point is critical for ensuring the highest level of system security.

4.3. Inference time analysis

After assessing the accuracy performance of the proposed architecture, the inference time of the model was assessed on an NVIDIA Jetson AGX Orin Developer Kit using the Libtorch library, version 1.12.0 [44]. Each measure was repeated one million times, considering the worst-case packet lengths taken from the dataset. Then, such measures were evaluated using three aggregated metrics: maximum inference time, median inference time (50th percentile), and mean inference time. To assess the performance in realistic scenarios, the measurements were performed considering two different power configurations of the platform: 30 W and EDP (full power) [45].

Results. Table 6 shows the inference time of the proposed architecture. Thanks to the high computational parallelization capability of the convolutional layers, this network can achieve lower inference times for each packet compared with existing fully connected and recurrent neural networks (e.g., [34]).

The inference time obtained above is crucial for reducing the overall end-to-end latency of the packets and effectively applying the proposed IDS in practical settings where the typical packet latency is in the order of milliseconds or tens of milliseconds. In these regards, note that the detection process applied by the IDS can be performed in parallel with the propagation of the packet through the networking stack, as a way of monitoring the security of the system against external attackers. With this configuration, the detection latency is not added to the end-to-end packet latency; rather, it only affects the capability of the system to react to an incoming attack in timely fashion. The response times observed in the evaluation can be deemed appropriate for most use cases in CPS development, since they allow initiating a reaction to the detected attack (including raising a security alarm or blocking a network connection from the source of the anomalous packet) within milliseconds from packet reception.

Memory usage of the IDS including the final model implemented in C++ using the libtorch library is approximately 350 MB.

4.4. Optimizing the GPU inference time

To enhance the performance of the networking stack, modern network interfaces implement specialized mechanisms designed to reduce the overhead caused by network interrupts when receiving packets, by providing interrupts for a group of packets rather than for a single packet at a time. For instance, the New API (NAPI) interface in the Linux kernel implements this interrupt mitigation approach via polling, where the interface periodically checks for arriving packets instead of relying solely on interrupts.

When such a mechanism is adopted in packet reception and a group of packets (instead of a single one) is made available to higher-level components, it is possible to leverage the inherent parallel processing capabilities of GPUs to perform anomaly detection on multiple packets at the same time to maximize the packet processing throughput of the IDS. In this case, a set of packets, referred to as a *batch*, can be processed at the same time by a neural network implementation optimized for GPU inference, in an approach known as *batching*. In this approach, the

Table 4
Detection performances of the 1D CNN AE and two techniques from the literature, including the state-of-the-art Kitsune IDS (Mirsky et al. [9]), on different attack classes of the considered dataset. The *Norm.* and *Mal.* columns show how many packets of the considered attack classes were classified as respectively normal and malicious by each of the considered IDS. The *Prec.* column shows the true positive rate.

Attack	1D CNN AE			Ferrag et al. [29][a]			Kitsune IDS [9]		
	Norm.	Mal.	Prec.	Norm.	Mal.	Prec.	Norm.	Mal.	Prec.
Backdoor	0	24,914	100%	–	–	0.23%	25	24,889	99.90%
DDoS HTTP	0	229,142	100%	–	–	99%	5339	223,803	97.67%
DDoS ICMP	1838	2,912,518	99.93%	–	–	99%	1,851,198	1,063,158	36.48%
DDoS TCP	0	2,020,152	100%	–	–	99%	1,182,597	837,555	41.46%
DDoS UDP	48	3,215,684	99.99%	–	–	99%	2,821,162	394,570	12.27%
MITM	0	1229	100%	–	–	66.21%	262	967	78.68%
OS Fing.	0	1176	100%	–	–	23.45%	22	1154	98.13%
Password	0	1,053,893	100%	–	–	0.22%	421	1,053,472	99.96%
Port Scan	105	23,224	99.54%	–	–	0.23%	21,659	1670	7.16%
Ransom	0	11,030	100%	–	–	0.23%	50	10,980	99.55%
SQL Inject.	0	51,228	100%	–	–	0.23%	26	51,202	99.95%
Uploading	0	37,644	100%	–	–	0.23%	26	37,618	99.93%
Vuln. Scan	0	265,828	100%	–	–	0.23%	106	265,721	99.96%
XSS	0	16,215	100%	–	–	0.23%	35	16,181	99.79%
TOTAL	1991	9,863,877	**99.97%**	–	–	**0.24%**	5,882,928	3,982,940	**40.37%**

[a] The IDS of the dataset authors was trained using normal and DDoS packets flows (not on single packets), so the comparison cannot be done in terms of packets, but just as percentages.

Table 5
Comparison of the detection accuracy of the proposed architecture with and without the shortcuts of the encoder.

Attack	With skip connections			Without skip connections		
	Norm.	Mal.	Prec.	Norm.	Mal.	Prec.
Backdoor	0	24,914	100%	10	24,904	99,95%
DDoS HTTP	0	229,142	100%	906	228,236	99,60%
DDoS ICMP	1838	2,912,518	99.93%	127	2,914,229	99,99%
DDoS TCP	0	2,020,152	100%	0	2,020,152	100%
DDoS UDP	48	3,215,684	99.99%	14	3,215,718	99,99%
MITM	0	1229	100%	0	1229	100%
OS Fing.	0	1176	100%	0	1176	100%
Password	0	1,053,893	100%	334	1,053,559	99,96%
Port Scan	105	23,224	99.54%	105	23,224	99,54
Ransom	0	11,030	100%	7	11,023	100%
SQL Inject.	0	51,228	100%	0	51,228	100%
Uploading	0	37,644	100%	458	37,186	98,78%
Vuln. Scan	0	265,828	100%	160	265,668	99,93%
XSS	0	16,215	100%	564	15,651	96,89%
TOTAL	1991	9,863,877	**99.97%**	2685	9,863,183	**99,96%**

Table 6
Inference time measured for the proposed architecture and Kitsune IDS under different platform configurations.

	Power	Max	Median	Mean
1D CNN AE	30 W	30 ms	15 ms	15 ms
	EDP	26 ms	15 ms	15 ms
Kitsune	30 W	143 ms	1.23 ms	6.78 ms
	EDP	96 ms	0.96 ms	4.78 ms

number of packets processed in a single inference of the neural network is referred to as *batch size*.

With an increase in the batch size, provided that sufficient computational resources are available on the GPU, the achieved packet processing throughput grows at the cost of increased latency for each specific packet (since collecting the result for single packets has to wait for the whole batch to complete). However, integrating the proposed batching mechanism with the already present NAPI mechanism allows for reducing this extra latency, since it is already paid for by the NAPI mechanism itself.

Moreover, the target application may be characterized by different requirements in terms of processing performance, available system resources, and required security responsiveness. For instance, the IDS mechanism might be implemented as a packet filter at the networking level, which processes packets before they are delivered to the higher levels in the stack, or as a monitoring system characterized by lower resource utilization, which however might detect anomalous packets after they were already delivered to the next layer. By exploiting a batching strategy on the GPU, both packet processing throughput in the IDS and GPU resource utilization can be maximized or adapted to the requirements of the application by exploring the design trade-offs related to packet processing throughput, packet latency, and resource utilization.

Table 7
Latency, throughput, and GPU utilization of the 1D CNN AE GPU implementation for different batch sizes.

Batch size	Avg. latency	Max. latency	Throughput
1	0.142 ms	0.146 ms	44.0 Mbps
2	0.142 ms	0.155 ms	176 Mbps
5	0.132 ms	0.144 ms	473 Mbps
10	0.189 ms	0.193 ms	664 Mbps
20	0.293 ms	0.297 ms	837 Mbps
30	0.448 ms	0.589 ms	634 Mbps
80	1.05 ms	1.06 ms	850 Mbps
100	1.30 ms	1.32 ms	951 Mbps
160	1.97 ms	1.98 ms	990 Mbps
200	2.42 ms	2.43 ms	1006 Mbps
350	4.08 ms	4.10 ms	1040 Mbps

The following experiments investigate the quantitative effects on the system performance of processing packets in batches when considering GPU implementations of the 1D CNN AE architecture proposed in this work. In particular, they report the average throughput and latency achieved with varying batch sizes.

Results. The performance results of the 1D CNN AE are reported in Table 7. With a batch size of 1 (i.e., when the batching mechanism is not active), the 1D CNN AE achieves an average throughput of 44 Mbps on the GPU with an average added latency of 0.142 ms for the packets in each batch, which is almost negligible compared with the typical expected end-to-end transmission latency. However, the processing capabilities of the GPU are not properly leveraged with a batch size of 1. Batch sizes larger than 1 give significant performance gains in terms of throughput. Specifically, when the batch size is increased to 20, a throughput of more than 800 Mbps is reached, with only a slight loss in terms of average latency (0.293 ms). By further increasing the batch size to 200, it is possible to completely use the parallelization capabilities of the GPU. In this setting, the throughput reaches 1 Gbps and the latency is increased to 4.08 ms per batch. Beyond this value for the batch size, no significant gain in throughput performance is observed, which is expected since the GPU is already at its maximum capacity.

Following this evaluation, it is possible to identify a reasonable compromise between latency and throughput with a batch size of 20 for the evaluated 1D CNN AE GPU implementation. However, different batch sizes can be selected to satisfy the application-specific requirements.

The above design space exploration enables an optimization of the batch size parameter depending on the requirements of the specific application to obtain the desired latency and throughput performance. Additionally, we can envision the use of adaptive batch size selection strategies to minimize the expected latency while maximizing the throughput based on the current incoming packet processing workload.

Overall, due to its highly parallelizable architecture, the 1D CNN AE proved to be very suitable for GPU-accelerated inference, providing significant throughput improvements compared with CPU inference. Such a new result further highlights the strength of the proposed 1D CNN architecture, reverting the findings reported in previous works [12,17,34], which gave an advantage of LSTM-based IDSs, with respect to CNN-based ones, due to their higher level of accuracy.

Note, however, that such comparative studies did not consider a thorough evaluation of the achieved timing performance, which is crucial to deploying the required IDS functionality in actual application scenarios. The experimental results reported in this paper, instead, show that the 1D CNN AE architecture dominates the LSTM-based solutions from a timing perspective, especially when GPU acceleration is available, while achieving comparable detection accuracy.

Memory usage of the GPU-accelerated implementation of the IDS, which leverages the NVIDIA TensorRT framework for GPU acceleration, is approximately 3 GB. In contrast, the edge computing platform considered in the experiments (NVIDIA Jetson AGX Orin Developer Kit) features 32 GB of main memory.

4.5. Poisoning robustness

Ideally, a model should be trained using only benign samples. However, in a real-world setting this assumption may not hold, because it would require having full control over the deployment environment. Nkashama et al. [13] investigated this issue for several existing IDS approaches and showed that even a small injection of malicious packets (5%) into the benign ones used for training could abruptly reduce the detection capabilities of the considered solutions.

This section presents an analysis of the robustness of the proposed solution against the poisoning of the training dataset. Based on the results found by Nkashama [13], the original training set was poisoned by substituting 5% of benign packets in the training data with malicious ones. This substitution has been performed by randomly selecting malicious packets from the dataset and using them to replace an equal number of randomly selected normal packets in the training and evaluation sets. This type of training simulates erroneous or intentional replacement of a portion of the training packets with malicious packets during training that are still labeled as normal packets, which may potentially reduce the resulting accuracy of the IDS. Then, the 1D CNN AE was trained as before, but using the poisoned data as training and evaluation sets. Finally, the performance achieved by such a trained model was assessed on the original normal and malicious packets of the test set.

The performance of the proposed network was analyzed using Receiver Operating Characteristic (ROC) curves (Fig. 4), which plot the TPR as a function of the FPR for various threshold values. The goodness of a ROC curve is evaluated by the area under the curve (AUC), which captures how much the model can distinguish between two classes of packets (i.e., benign and malicious samples). In an ideal situation, when detection is perfect, this curve reaches the top-left edge of the graph and thus the AUC value is 1. In general, the higher the AUC, the better the model is at detecting an anomaly.

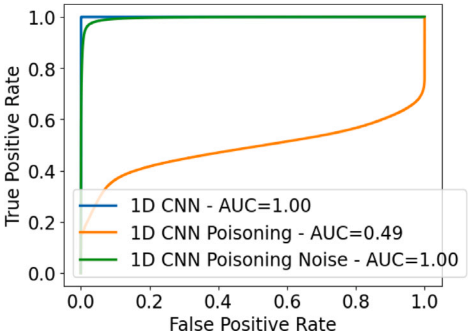


Fig. 4. ROC curve for the proposed 1D CNN AE when trained with the normal and the 5% poisoned datasets.

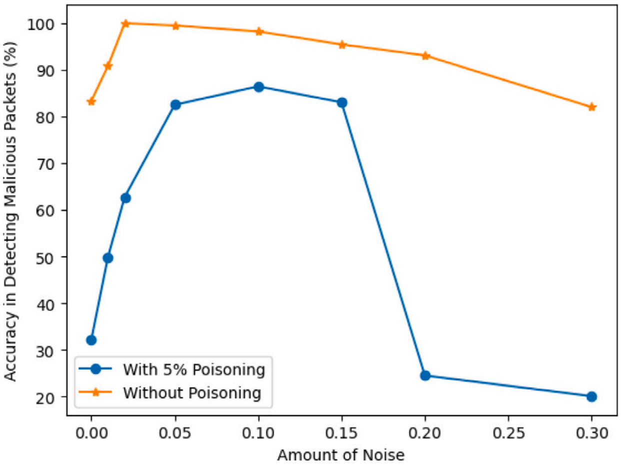


Fig. 5. Analysis of the effect of introducing random noise to the input in training phase on the detection capabilities of malicious packets.

To investigate the importance of introducing random noise on the inputs used for training, the proposed architecture was tested both with and without such a disturbance. Moreover, to provide a baseline, the results were compared with the one achieved when training the network without poisoning. Fig. 4 shows the ROC curves of the 1D CNN AE when trained using the normal and poisoned datasets. Note that, without using noise in the training phase, but with a poisoned dataset, the detection capabilities of the network are almost completely sabotaged. However, by introducing a small perturbation to the input bytes of the packet, the robustness of the network to such an attack is greatly improved, reaching a performance that is very close to the one achieved without poisoning.

We performed an extensive analysis to measure of the effect of introducing such a random perturbation on the input. This was achieved by repeating the training of the proposed architecture on the poisoned and on the normal dataset while varying the amount of random noise applied to the input. For each noise configuration, the network was then tested in terms of its detection capabilities against malicious packets. Results shown in Fig. 5 show that the addition of noise should be at least about 0.02 to be effective in improving the performance of the network. At the same time, if the noise amount is too high then the network stops learning properly and its detection capabilities start to degrade. This is particularly noticeable when the training is performed on the poisoned dataset, where the effect of adding 0.20 of random noise to the input destroys the detection capabilities of the autoencoder. Given these results, for the purpose of deploying the autoencoder in a real setting the optimal amount of noise that should be introduced to guarantee the maximum detection accuracy if the dataset is poisoned is about 0.10.

Such a type of robustness is highly beneficial in practice, as it can still ensure good detection capability even when the training set is not clean. It also broadens the deployment capabilities, since one should be able to train his IDS directly on the target deployment environment using data recorded during run time. This is possible because, even if an attacker can inject some malicious packets in the training data, if the amount of malicious traffic is below a certain fraction of the total traffic (e.g., 5% based on our tests), this will not significantly impact the IDS performance. This threshold assumption is a representative value of a realistic scenario, where the attacker typically tries to minimize the amount of malicious traffic to avoid being detected by traditional IDSs.

4.6. Comparison with existing work

To position the proposed solution with respect to the existing literature, the experiments were repeated on the solutions proposed by the dataset's authors [29] and by Mirsky et al. [9], who presented a leading state-of-the-art IDS (Kitsune).

In addition to introducing the EDGE-IIOTSET dataset, Ferrag et al. [29] applied the dataset to train and evaluate both a binary and a multiclass classifier, using as input a set of features manually extracted from the packet flow. Due to the nature of the proposed solutions, the performance comparison was carried out with the binary classifier only. Based on the information provided in [29], the same model used by the authors (referred to as *reference architecture*) was recreated and trained using the same hyperparameters. This reference architecture needs to be trained using both normal and malicious packets. However, since we would like to evaluate also the generalization capabilities of the different approaches to new unseen attacks, we performed this training using the benign samples and only a subset of the attack classes (i.e., by considering DDoS attacks only). Then, the evaluation was performed considering also the other unseen attack classes. This was necessary to evaluate how well the reference architecture could detect new types of attacks.

As presented in Section 2.2, the Kitsune IDS is a multi-stage solution where the first stage extracts and selects the features of each packet to be fed to the autoencoders in the second stage. To perform the comparison with this model, the implementation provided by the authors was retrained on the common dataset, setting the threshold at the 99th percentile of the benign samples loss (like with the proposed model).

The evaluation results comparing the proposed solution with the ones by Ferrag et al. [29] and Mirsky et al. [9] are discussed below.

Results. In Table 4 we reported the results of our experiments using the existing solutions by Ferrag et al. [29] and Mirsky et al. [9] when applied to the EDGE-IIOTSET dataset. The solution by Ferrag et al. [29] can achieve excellent performance (99% accuracy) in detecting the attacks it was trained on. However, it is unable to detect the unseen ones. Table 4 reports the TPR achieved by the reference architecture on all attack flows when trained using normal and DDoS malicious flows (the same test has been repeated using different malicious flows for the training, achieving similar results). Since the work by Ferrag et al. [29] uses per-flow instead of per-packet detection, it is not meaningful to compare the inference times it achieves with respect to the timing performance of the proposed approach. As the table shows, for unseen attacks, its detection accuracy is close to 0%, classifying almost all malicious packets as normal packets. Such results confirm that the proposed unsupervised approach is much more robust than an supervised one since it can correctly identify new types of attacks as well as known attacks (cf. Section 4.1).

Regarding the Kitsune IDS, the global detection accuracy on the considered dataset is close to (but still lower than) the one achieved by the proposed solution in some specific attack classes such as XSS and Backdoor attacks. However, Table 4 clearly shows that, although the Kitsune IDS performs similarly to the proposed solution on some attack classes, it is almost unable to detect other attack classes (e.g., DDoS attacks) whereas the proposed IDS performs very well in all classes.

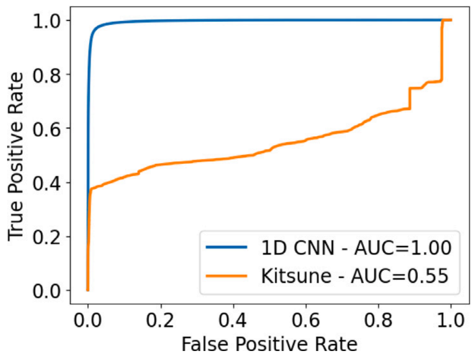


Fig. 6. ROC curve comparing the effect of poisoning with the proposed approach trained with noise and with the Kitsune IDS [9].

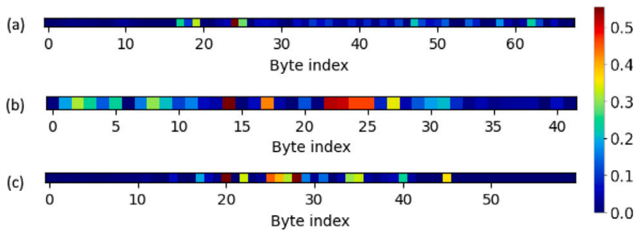


Fig. 7. Heatmaps of some sample packets reconstructed by the proposed 1D CNN AE model. Normal bytes are colored in blue, while anomalous ones are in red. The color bar illustrates the full color mapping used to represent the reconstruction loss. (a) Normal packet; (b) Ransomware packet; (c) DDoS ICMP. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Concerning the inference time (Table 6), the Kitsune IDS achieves a better mean inference time compared to the proposed solution, but its maximum inference time is much higher. This is because the computation time for the feature extraction of the Kitsune IDS highly depends on the packet length. Since, in the considered dataset, most of the packets are short (around 60–80 bytes), the mean inference time is highly affected by the low inference time achieved for such packets. However, when fed with longer packets (e.g., 1500 bytes), then their inference time increases almost linearly by about two orders of magnitude. Instead, due to limitations that are currently imposed by the machine learning libraries, the proposed approach is independent of the input size since it always considers the inference time required by the longest packet that can be fed into the network. However, it is worth noting that when considering the longest packets, the proposed solution outperforms the Kitsune IDS by one order of magnitude (see Max inference time).

Finally, we compared the poisoning robustness of the proposed method with that of the Kitsune IDS [9]. We did not consider the solution by Ferrag et al. [29] due to the different nature of the data taken as input (flows rather than single packets). To perform a fair evaluation we trained the Kitsune IDS using the poisoned dataset presented in Section 4.5 and then we tested it on the normal and malicious datasets. Results presented in Fig. 6 confirm the findings of Nkashama et al. [13], and highlight how the Kitsune state-of-the-art solution is not robust to poisoning (dropping to 0.55 AUC), whereas the proposed approach is (retaining approximately 1.0 AUC).

4.7. Explainability issues

Explainable AI is about understanding how a machine learning model obtained a particular solution. Modern AI-based IDSs can analyze large amounts of data, recognize patterns, and identify suspicious activity more accurately and efficiently than traditional IDSs.

However, this comes with a challenge: the lack of transparency and interpretability of AI algorithms, which can make it difficult to understand how a system arrived at a particular decision or identified a potential threat. Because of this, explainable AI (XAI) methods have lately been studied to help make AI-based systems more transparent and interpretable. In the context of IDSs, XAI can provide insights into how the system makes decisions and identify the features that trigger an alert or flag an activity as suspicious.

This section presents a heatmap-based mechanism that can be used to understand why a given packet is classified as an anomaly. The proposed detection method makes use of the reconstruction loss and a threshold to detect anomalous packets. As explained in Section 3.4, this loss is computed based on the reconstruction error of each byte of the packet (Eq. (1)). Thus, if considering the reconstruction error of the single bytes, it is possible to understand which ones have a higher impact on the threshold value and thus which are the most anomalous ones. The higher the loss value of a specific byte, the more likely that byte represents an anomaly with respect to the expected network traffic. On the other hand, the closer to zero the reconstruction error of a specific byte, the more likely the byte is considered normal by the system. Plotting such values as a heatmap produces a visual explanation of a specific detection decision.

As illustrated in Fig. 7, normal packets have a small or negligible reconstruction error on the majority of their bytes except for those that are more subject to change (e.g., the sequence number). Instead, in malicious packets, the reconstruction error spreads widely across the whole packet. In particular, when dealing with ransomware packets, the IDS considers as anomalous mainly the payload and the IP addresses, whereas, when reconstructing a DDoS ICMP packet, the addresses and the ethertype are considered to be correct (as in a normal traffic lots of ICMP packets are regularly exchanged), thus the error comes mainly from the payload. This is a further demonstration of how the proposed detection method was able to correctly learn to recognize the specific features that are characteristic of the expected traffic, and distinguish them from the anomalous ones.

4.8. Portability

Evaluating the performance of the proposed approach across multiple datasets is crucial to evaluate the generalization capabilities of the IDS across multiple environments. Therefore, we considered the TON_IoT dataset [28] to perform additional testing. TON_IoT is another IoT-related dataset which is widely used in the literature on IDSs. We trained and tested the proposed autoencoder architecture on this dataset with the same hyperparameters used in the previous experiments.

Unfortunately, it was not possible to evaluate the performance of the Kitsune architecture on such a dataset due to an incompatibility between the expected input format of Kitsune and the packet format provided by the raw captures in TON_IoT, with the result that the original Kitsune implementation [9] failed to extract the features during the learning phase. On the other hand, we note that the proposed autoencoder approach does not make specific assumptions on the packet format, thus enabling out-of-the-box compatibility with different packet formats.

The results of the performance evaluation for the proposed autoencoder approach on the TON_IoT dataset are reported in Table 8. It is worth noticing that, even though results achieved on the per-packet detection seem lower than the ones achieved on the EDGE-IIOTSET dataset, this is partially due to the different nature of the two datasets. More in detail, differently from EDGE-IIOTSET, the raw network captures in the TON-IoT dataset are organized in pcap files that are categorized under either normal captures or malicious captures depending on the presence of an ongoing attack during the capture of such traffic. As a result, pcap files that are overall categorized under malicious captures also contain packets that should be considered as normal. This choice made by the dataset authors is reasonable since the raw pcap files are then processed to generate csv files containing features for each flow of packets, and each of such sets of features is then labeled as either normal or malicious based on detailed logging information.

Table 8
Performance of the proposed architecture on the TON_IoT dataset.

TP	TPR	FP	FPR	F1-score
15,560,042	**88%**	19,547	1,0%	**0,9341**

Therefore, given that the pcap files containing malicious traffic also contain normal packets (about 10% of the traffic), when performing the per-packet evaluation with the proposed approach some normal packets may correctly be classified as non-malicious while the corresponding packet flow is categorized as malicious within the dataset.

5. Discussion

The experimental studies carried out in this work show that the developed 1D CNN AE architecture can learn the features of the packets it was trained on and thus properly distinguish between normal and malicious packets.

The presented inference time analysis provided an estimation of the inference time of such a detection algorithm on a real platform. This kind of analysis is crucial to correctly deploy IDS in a real setting, but it was largely ignored in prior IDS research. The experimental results showed that, in terms of inference time, the 1D CNN AE presented in this work is more efficient than existing approaches based on recurrent neural networks. Moreover, it is worth noticing that the overhead introduced by the proposed solution is comparable with the processing latency, which is normally experienced by network packets, thus it is applicable in real-world cyber–physical systems. The proposed architecture was trained and tested on the EDGE-IIOTSET and TON_IoT datasets. As with other IDS solutions, applying this solution in an environment with different expected normal traffic requires retraining it to learn the new features. In short, the proposed network not only exhibits excellent detection capabilities, but also provides reasonable inference times. It is worth noting that inference timing results reported here were achieved without exploiting network quantization or other optimizations, since they are currently not supported by the underlying library. We expect that leveraging such optimizations will further improve the already good timing performance. We plan to investigate such optimizations in a future work.

The network proposed in this work has an accuracy comparable to that of the state-of-the-art supervised approach when tested on already-known malicious attacks. More importantly, when dealing with novel attacks, our solution can maintain the same accuracy as the one observed with the known attacks, whereas the evaluated approach by Ferrag et al. [29] exhibited poor detection performance. The achieved results confirm the importance of unsupervised learning for IDSs, given that it does not require continuous retraining to detect novel threats but only needs to update the model if the network traffic pattern changes. Moreover, when compared with the state-of-the-art unsupervised approach (Mirsky et al. [9]), the proposed architecture experimentally demonstrated a better capability in identifying malicious packets, as well as a lower worst-case inference time.

The exploitation of the parallel processing capabilities of GPUs allowed further improving the timing performance of the proposed architecture. Experiments showed that the 1D CNN reached about 800 Mbps of throughput while preserving a very low latency. Also, a proper selection of the batch size at inference time allowed to further improve both latency and throughput performance for the specific application, while optimizing the utilization of the GPU resources available.

Previous works in the literature [12,17] reported a superior accuracy of LSTM-based approaches over CNN-based approaches. Differently, the experimental results presented in this work highlighted the importance of properly accounting for timing performance to determine the applicability of an IDS technique in real settings. This is especially critical for systems to be deployed in the context of cyber–physical applications.

This work also showed that, introducing random noise in the input samples used for training, the proposed method not only can detect novel unseen malicious packets, but is also robust to the contamination of the training set. The robustness of the IDS against poisoning attacks is a crucial property, as it provides security guarantees not only when training the IDS on public untrusted datasets, but also when deploying the IDS in a real environment. Despite its importance, data contamination has largely been ignored by state-of-art IDS solutions. The analysis presented in this work also demonstrated that the proposed architecture is much more robust to dataset poisoning than state-of-the-art baseline IDS solutions [13].

Finally, the explainability method presented in this work can help the user to understand why a given packet is detected (or not) as malicious and which bytes are corrupted. This not only makes the proposed IDS more transparent and understandable, but also highlights potential vulnerabilities in the detection process. The achieved results show that when reconstructing anomalous packets, the proposed architecture correctly identifies the malicious features of each specific attack, even though they had never been shown during the training phase. In a real-world setting, this explainability mechanism can be used as part of the reporting process to provide valuable insights about what is happening in the monitored infrastructure to the incident response team. Thanks to the generated heatmap an analyst could more easily identify the malicious patterns in the anomalous packets, thus simplifying the understanding of the attack that has been attempted.

6. Conclusions

This paper presented a novel packet-level anomaly-based intrusion detection system for networked cyber–physical systems, based on a 1D CNN autoencoder fully trained by an unsupervised learning paradigm. The unsupervised approach made the proposed IDS capable of detecting different types of malicious attacks without showing the corresponding packets in the training phase. This makes the proposed IDS able to detect new types of attacks without retraining the model.

The proposed autoencoder architecture was carefully designed to achieve high accuracy, reduced inference time, and enhanced poisoning robustness. Experimental results showed that the proposed technique substantially outperforms state-of-the-art supervised learning-based solutions in terms of detecting novel attacks. Moreover, when compared with leading unsupervised learning techniques, the proposed solution showed better performance both in terms of detection accuracy and worst-case inference time. The throughput was further improved by introducing a novel way to exploit the parallel architecture of GPUs by using batching at inference time. An experimental evaluation carried out on a real embedded platform further demonstrated the applicability of the proposed technique on CPS devices.

Finally, this work presented a method to help the user interpret the outputs produced by the model in detecting anomalous packets, making the system more transparent and understandable. This explainability method was also used to confirm the ability of the proposed approach to learn the 'right' characteristics of the genuine packets, and to distinguish them from malicious ones for a wide range of attacks. In future work, we plan to investigate optimization strategies to further improve the inference time of the proposed approach.

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported in part by the Italian Ministry of University and Research (MUR), under the SPHERE project funded within the PRIN-2017 framework (grant no. 20172NNB4T_001) and in part by the Project SERICS under the Ministry of University and Research (MUR) National Recovery and Resilience Plan funded by the European Union–NextGenerationEU under Grant PE00000014.

References

[1] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, F. Ahmad, Network intrusion detection system: A systematic study of machine learning and deep learning approaches, Trans. Emerg. Telecommun. Technol. 32 (1) (2021) e4150.

[2] J. Lansky, S. Ali, M. Mohammadi, M.K. Majeed, S.H.T. Karim, S. Rashidi, M. Hosseinzadeh, A.M. Rahmani, Deep learning-based intrusion detection systems: a systematic review, IEEE Access 9 (2021) 101574–101599.

[3] Y. Chen, Y. Li, X.-Q. Cheng, L. Guo, Survey and taxonomy of feature selection algorithms in intrusion detection system, in: H. Lipmaa, M. Yung, D. Lin (Eds.), Information Security and Cryptology, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 153–167.

[4] A.H. Sung, S. Mukkamala, The feature selection and intrusion detection problems, in: M.J. Maher (Ed.), Advances in Computer Science - ASIAN 2004. Higher-Level Decision Making, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 468–482.

[5] N. Gao, L. Gao, Q. Gao, H. Wang, An intrusion detection model based on deep belief networks, in: 2014 Second International Conference on Advanced Cloud and Big Data, IEEE, 2014, pp. 247–252.

[6] M. Yousefi-Azar, V. Varadharajan, L. Hamey, U. Tupakula, Autoencoder-based feature learning for cyber security applications, in: 2017 International Joint Conference on Neural Networks, IJCNN, IEEE, 2017, pp. 3854–3861.

[7] G. Kathareios, A. Anghel, A. Mate, R. Clauberg, M. Gusat, Catch it if you can: Real-time network anomaly detection with low false alarm rates, in: ICMLA, 2017.

[8] J. Dromard, G. Roudière, P. Owezarski, Online and scalable unsupervised network anomaly detection method, IEEE Trans. Netw. Serv. Manag. 14 (1) (2017) 34–47.

[9] Y. Mirsky, T. Doitshman, Y. Elovici, A. Shabtai, Kitsune: An ensemble of autoencoders for online network intrusion detection, 2018, arXiv:1802.09089.

[10] E. Tekiner, A. Acar, A.S. Uluagac, A lightweight IoT cryptojacking detection mechanism in heterogeneous smart home networks, in: Proceedings 2022 Network and Distributed System Security Symposium, 2022.

[11] H. Jmila, M.I. Khedher, Adversarial machine learning for network intrusion detection: A comparative study, Comput. Netw. 214 (2022) 109073, http://dx.doi.org/10.1016/j.comnet.2022.109073, URL https://www.sciencedirect.com/science/article/pii/S1389128622002146.

[12] R.K. Malaiya, D. Kwon, J. Kim, S.C. Suh, H. Kim, I. Kim, An empirical evaluation of deep learning for network anomaly detection, in: 2018 International Conference on Computing, Networking and Communications, ICNC, IEEE, 2018, pp. 893–898.

[13] D. Nkashama, A. Soltani, J.-C. Verdier, M. Frappier, P.-M. Tardif, F. Kabanza, Robustness evaluation of deep unsupervised learning algorithms for intrusion detection systems, 2022, arXiv preprint arXiv:2207.03576.

[14] M.Z. Alom, V. Bontupalli, T.M. Taha, Intrusion detection using deep belief networks, in: 2015 National Aerospace and Electronics Conference, NAECON, IEEE, 2015, pp. 339–344.

[15] T. Vaiyapuri, A. Binbusayyis, Application of deep autoencoder as an one-class classifier for unsupervised network intrusion detection: a comparative evaluation, PeerJ Comput. Sci. 6 (2020) e327.

[16] T. Truong-Huu, N. Dheenadhayalan, P. Pratim Kundu, V. Ramnath, J. Liao, S.G. Teo, S. Praveen Kadiyala, An empirical study on unsupervised network anomaly detection using generative adversarial networks, in: SPAI, 2020.

[17] D. Kwon, K. Natarajan, S.C. Suh, H. Kim, J. Kim, An empirical study on network anomaly detection using convolutional neural networks, in: 2018 IEEE 38th International Conference on Distributed Computing Systems, ICDCS, IEEE, 2018, pp. 1595–1598.

[18] B. Andreas, J. Dilruksha, E. McCandless, Flow-based and packet-based intrusion detection using BLSTM, in: SMU Data Science Review, 2020.

[19] M.S. Alam, B.R. Fernando, Y. Jaoudi, C. Yakopcic, R. Hasan, T.M. Taha, G. Subramanyam, Memristor based autoencoder for unsupervised real-time network intrusion and anomaly detection, in: Proceedings of the ICONS, 2019.

[20] F. Carrera, V. Dentamaro, S. Galantucci, A. Iannacone, D. Impedovo, G. Pirlo, Combining unsupervised approaches for near real-time network traffic anomaly detection, Appl. Sci. 12 (3) (2022).

[21] I.J. King, H.H. Huang, Euler: Detecting network lateral movement via scalable temporal link prediction, ACM Trans. Priv. Secur. 26 (3) (2023) http://dx.doi.org/10.1145/3588771.

[22] S. Roy, J. Li, V. Pandey, Y. Bai, An explainable deep neural framework for trustworthy network intrusion detection, in: 2022 10th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2022, pp. 25–30, http://dx.doi.org/10.1109/MobileCloud55333.2022.00011.

[23] S. Mane, D. Rao, Explaining network intrusion detection system using explainable AI framework, 2021, arXiv:2103.07110.

[24] G. Mohi-ud din, NSL-KDD, 2018, http://dx.doi.org/10.21227/425a-3e55.

[25] N. Moustafa, J. Slay, UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set), in: MilCIS, 2015.

[26] I. Sharafaldin, A.H. Lashkari, A.A. Ghorbani, Toward generating a new intrusion detection dataset and intrusion traffic characterization, in: International Conference on Information Systems Security and Privacy, 2018.

[27] M. Al-Hawawreh, E. Sitnikova, N. Aboutorab, X-IIoTID: A connectivity-agnostic and device-agnostic intrusion data set for industrial Internet of Things, IEEE Internet Things J. 9 (5) (2021) 3962–3977.

[28] N. Moustafa, A new distributed architecture for evaluating AI-based security systems at the edge: Network TON_IoT datasets, Sustainable Cities Soc. 72 (2021) 102994.

[29] M.A. Ferrag, O. Friha, D. Hamouda, L. Maglaras, H. Janicke, Edge-IIoTset: A new comprehensive realistic cyber security dataset of IoT and IIoT applications for centralized and federated learning, IEEE Access 10 (2022) 40281–40306.

[30] S. Hettich, S. Bay, The UCI KDD Archive, University of California, Department of Information and Computer Science, Irvine, CA, 1999, p. 152, http://kdd.ics.uci.edu.

[31] M. Lanvin, P.-F. Gimenez, Y. Han, F. Majorczyk, L. Mé, E. Totel, Errors in the CICIDS2017 dataset and the significant differences in detection performances it makes, in: International Conference on Risks and Security of Internet and Systems, Springer, 2022, pp. 18–33.

[32] D.K. Bhattacharyya, J.K. Kalita, Network Anomaly Detection: A Machine Learning Perspective, Crc Press, 2013.

[33] D. Kwon, H. Kim, J. Kim, S.C. Suh, I. Kim, K.J. Kim, A survey of deep learning-based network anomaly detection, Cluster Comput. 22 (2019) 949–961.

[34] N. Borgioli, L. Thi Xuan Phan, F. Aromolo, A. Biondi, G. Buttazzo, Real-time packet-based intrusion detection on edge devices, in: Proceedings of Cyber-Physical Systems and Internet of Things Week 2023, in: CPS-IoT Week ’23, Association for Computing Machinery, New York, NY, USA, 2023, pp. 234–240, http://dx.doi.org/10.1145/3576914.3587551.

[35] J. Newsome, B. Karp, D.X. Song, Paragraph: Thwarting signature learning by training maliciously, in: International Symposium on Recent Advances in Intrusion Detection, 2006.

[36] N. Dalvi, P. Domingos, Mausam, S. Sanghai, D. Verma, Adversarial classification, in: Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’04, Association for Computing Machinery, New York, NY, USA, 2004, pp. 99–108, http://dx.doi.org/10.1145/1014052.1014066.

[37] S. Neupane, J. Ables, W. Anderson, S. Mittal, S. Rahimi, I. Banicescu, M. Seale, Explainable intrusion detection systems (X-IDS): A survey of current methods, challenges, and opportunities, IEEE Access 10 (2022) 112392–112415, http://dx.doi.org/10.1109/ACCESS.2022.3216617.

[38] J.T. Springenberg, A. Dosovitskiy, T. Brox, M. Riedmiller, Striving for simplicity: The all convolutional net, 2015, arXiv:1412.6806.

[39] O. Ronneberger, P. Fischer, T. Brox, U-net: Convolutional networks for biomedical image segmentation, in: Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18, Springer, 2015, pp. 234–241.

[40] J. Santokhi, P. Daga, J. Sarwar, A. Jordan, E. Hewage, Temporal autoencoder with u-net style skip-connections for frame prediction, 2020, arXiv preprint arXiv:2011.12661.

[41] A.-S. Collin, C. De Vleeschouwer, Improved anomaly detection by training an autoencoder with skip connections on images corrupted with stain-shaped noise, in: 2020 25th International Conference on Pattern Recognition, ICPR, IEEE, 2021, pp. 7915–7922.

[42] P. Vincent, H. Larochelle, Y. Bengio, P.-A. Manzagol, Extracting and composing robust features with denoising autoencoders, in: Proceedings of the 25th International Conference on Machine Learning, ICML ’08, Association for Computing Machinery, New York, NY, USA, 2008, pp. 1096–1103, http://dx.doi.org/10.1145/1390156.1390294.

[43] F. Razmi, L. Xiong, Classification auto-encoder based detector against diverse data poisoning attacks, 2022, arXiv:2108.04206.

[44] P. Foundation, Libtorch, 2022, https://pytorch.org/cppdocs/installing.html.

[45] N. Corporation, NVIDIA jetson orin - tuning power, 2022.