Empirical Software Engineering (2026) 31:23
https://doi.org/10.1007/s10664-025-10766-3

On detection latencies of network intrusion detectors – discussion and application

Tommaso Puccetti[1] · **Andrea Ceccarelli**[1]

Received: 4 March 2025 / Accepted: 3 November 2025 / Published online: 29 November 2025

Abstract
The ever-evolving landscape of attacks and the growing complexity of ICT systems make crafting anomaly-based intrusion and error detectors difficult: they must accurately detect attacks and promptly perform detections. Although improving and comparing the detection capability is the focus of most research works, the timeliness of the detection is less considered and often insufficiently evaluated or discussed. In this paper, we argue the relevance of measuring the temporal latency of attacks, and we propose an evaluation approach for detectors to ensure a trade-off between correct and in-time detection. Briefly, the approach relates the false positive rate to the temporal latency of attacks, ultimately leading to guidelines for configuring a detector. We discuss and apply the strategy to compose datasets for intrusion detection that can support the computation of our metrics. We exercise our approach by evaluating different intrusion and error detectors in three industrial cases: i) an embedded railway on-board system that optimizes public mobility, ii) an edge device for the Industrial Internet of Things, and iii) an IoT network that monitors an industrial facility. Results show that considering latency in addition to traditional metrics like the false positive rate, precision, and recall gives an additional fundamental perspective on the actual performance of the detector and should be considered when assessing and configuring intrusion detectors.

Keywords Attack latency · Anomaly detection · Intrusion detection · Metrics · Evaluation

Communicated by: Lei Ma, Roberto Pietrantuono.

✉ Tommaso Puccetti
tommaso.puccetti@unifi.it

Andrea Ceccarelli
andrea.ceccarelli@unifi.it

[1] Department of Mathematics and Informatics, University of Florence, Viale Morgagni 65, 50142 Florence, Italy

1 Introduction

Complex information and communication systems play a central role in daily operations, industrial processes, and even safety–critical contexts where a maximum frequency of failure per hour must be guaranteed, and attacks may have catastrophic consequences, also impacting the well-being of people and the environment (Lyu et al. 2019). They are connected through the network, either to be accessible by the end-user or to orchestrate distributed components to provide an emergent behavior (Kopetz et al. 2015). The complexity of these systems makes them susceptible to errors that may arise from diverse sources and exposes them to attacks.

Anomaly detection is a widely researched initiative among the multiple approaches to tolerate attacks and failures. It involves deploying error or intrusion detection systems (Cruz et al. 2015; Cotroneo et al. 2017; Zhao et al. 2019; Palmieri 2019; Al and Dener 2021) that monitor system indicators (often called *features*) during the execution and utilize the values collected to suspect errors or attacks. The foundational assumption underlying the construction of these anomaly detectors is that mistakes and attacks produce discernible anomalies in the trend of the values collected from monitored features; luckily, this assumption is often realistic (Hossin and Sulaiman 2015).

A promising solution to build the detector is training a machine learning model relying on a large set of features encoded as structured data. The detection of anomalies is then formulated as a binary classification problem in which we predict a label for any input data points provided to the detector (0 for normal, 1 for attack). The training process allows the model to approximate the binary function that associates a label for each data point fed to the detector. It is performed either in a supervised or unsupervised way, depending on the availability of labels in the training set.

Exploiting the temporal order between data points is a distinguishing feature of anomaly-based intrusion detectors (IDs). Incorporating the time dimension enables the detection of contextual, group, and punctual anomalies (Lindemann et al. 2021). Contextual anomalies are data instances anomalous in a specific context but not otherwise, while group anomalies are collections of related data instances, anomalous to the entire data set (Chandola et al. 2009). Suppose classification relies solely on the information conveyed by individual data points. In that case, the detection capabilities are limited to punctual anomalies, whose values are different from the modeled distribution. Models can leverage time series data to make decisions based on system evolution: they may better describe the anomalous behavior triggered by an attack or a system error. The literature is aware of this, with numerous works that exploit the temporal order between data points (Yuan et al. 2021; Alharthi et al. 2021; Dayaratne et al. 2022; Wang et al. 2022b).

What we expect from an ID, independent of the model, is to minimize the risk of false alarms to avoid unnecessary responses, while being able to detect attacks or errors as soon as possible. In the literature, the correctness of the detection is a well-considered factor for evaluating the efficacy of ID. It is measured in terms of how many data points in a sequence are classified correctly, using state-of-the-art classification metrics such as f1-score, precision, recall, and accuracy (Hossin and Sulaiman 2015).

However, it is easy to observe that achieving high classification performance alone, without considering the time an attack or an error has been latent in the system, offers a partial

understanding. Unfortunately, the time required to identify an occurring attack or failure, i.e., the attack latency, is rarely measured. Two main factors substantiate this lack of attention: the training set and the metrics used. First, while training sets include time-stamped data points, the way they are typically built is insufficient to model the evolution of attacks through time. They normally aggregate raw network data in flows, resulting in data points describing an entire communication session between two machines. The problem is reduced to detecting anomalous data flows, preventing a fine-grained analysis. This way, the transition between normal and anomalous behavior of the systems is not reproduced, preventing the detector from learning this dynamic. Second, widely accepted metrics like accuracy or f1-score count the number of correctly classified data points. These metrics do not account for multiple anomalous data points belonging to the same attack. Consequently, detecting all the anomalous data points is not necessary; identifying just one is sufficient to start proper reaction strategies (Wang et al. 2022a). Additionally, detecting a malicious network flow only after it has terminated enables the attacker to achieve its goal and hinders timely detection and response.

For these reasons, accuracy, f1-score, or similar metrics might seem satisfactory to benchmark and assess detector performances, but they may prevent a more sensitive configuration of the detector based on the operational requirements. To overcome these limitations, we propose and apply an approach for evaluating IDs that considers latency and related metrics in addition to state-of-the-art classification metrics. This proves a complete understanding of a detector, with the ability to define a trade-off between the correct detection and time to detect. This implies proposing different dataset structures to overcome the limitation of the flow-based approach.

Very practically, we assume that a system has operational requirements defined in terms of an acceptable false positive rate. A high false positive rate means a suspicious detector that raises many unnecessary alarms, but also a higher detection capability.

Then, we proceed to evaluate our detectors as follows. Given an anomalous sequence, we set multiple false positive rate thresholds and compute the corresponding attack latency, which is the distance between the first anomalous data point and the first anomalous data point correctly classified. This allows us to identify the best configuration based on the system's operational requirements to defend.

We exercise the proposed approach and metric for evaluating IDs by executing three systems: SPaCe (Puccetti et al. 2024), a cyber-physical system that orchestrates and optimizes public mobility; Arancino (Zoppi et al. 2023b), a smart board for distributed monitoring; and DoS/DDoS-MQTT-IoT (Alatram et al. 2023), a sensor network for industrial applications. Experiments show that the proposed approach offers a novel perspective on the intrusion detection problem and facilitates efficient calibration of IDs for real deployment scenarios.

The code to reproduce our results, together with the detailed results from the experiments not included in the paper for brevity, is publicly available.

The rest of the paper is organized as follows: Section 2 provides an overview of the background and related works concerning anomaly detection. In Section 3, we introduce our evaluation approach and discuss the motivation behind it. Section 4 outlines the steps of our experimental campaign, while Section 5 presents the corresponding results. We conclude our discussion in Section 6, where we summarize the practical takeaway from our work.

2 Background and related work

We review basic notions of anomaly-based intrusion detection (Section 2.1) and details on state-of-the-art datasets (Section 2.2). We report on recent and notorious machine learning solutions applied to such a purpose (Section 2.3) and state-of-the-art metrics commonly used for the evaluation (Section 2.4).

2.1 Anomalies

Anomalies are patterns in data that do not conform to a well-defined notion of normal behavior. They might be induced for various reasons, such as malicious activity or system malfunctions (Lyu et al. 2019). In the case of IDs, anomalies can be observed by monitoring system features. Values collected from features are assembled in a univariate or multivariate ordered collection of data points.

It has been repeatedly proven that anomaly detection is a viable approach to detecting ongoing attacks (Cruz et al. 2015; Cotroneo et al. 2017; Al and Dener 2021) as an alternative to signature-based detectors. The latter performs satisfactorily when aiming to observe known effects, for example, to recognize the signature of a known attack (Zhao et al. 2019; Palmieri 2019).

However, signature-based detectors are weak against zero-day attacks or unprecedented events (Bilge and Dumitraş 2012). Especially in the case of zero days, these attacks exploit either new vulnerabilities or known vulnerabilities in novel ways and cannot be matched to known signatures. This problem cannot be neglected: zero-day attacks against a target system are a credible possibility, as confirmed by multiple reports in recent years (Bilge and Dumitraş 2012; Meakins 2019).

In anomaly-based detection, the problem of identifying attacks is reduced to finding anomalies. However, detectors are still far from perfect (Zoppi et al. 2021), and more research is needed.

Usually, anomaly-based IDs rely on machine learning algorithms, where the anomaly detectors learn a model using a training set composed of both normal data and anomalous data. The training can be *supervised*, leveraging the availability of labels explicitly indicating anomalous and normal data, or unsupervised, where labels are not available.

The data describes the system's behavior as monitored during its operation and encoded as a time series, a sequence of data points recorded at successive intervals. This series captures the system's evolution using either a single (univariate) or multiple (multivariate) features.

One approach to anomaly-based detection is time series classification (TSC), where each instance in the training set represents an entire time series. The goal is to determine whether a time series exhibits anomalous behavior (Baydogan et al. 2013; Ji et al. 2022). Alternatively, point-based classification focuses on classifying individual data points within a time series (Cotroneo et al. 2017; Puccetti et al. 2023).

In this paper, we focus on point-based classification of multivariate time series.

2.2 Network intrusion detection datasets

One of the most researched approaches to Intrusion Detection is Network Intrusion Detection, with data primarily consisting of network packets monitored during system communications. Network traffic is inherently heterogeneous, with each packet potentially differing in structure. As a result, not all features are collected from every packet, leading to a sparse dataset. This introduces an additional layer of complexity compared, for example, to error detection settings, where dataset observations capture the system's state using a homogeneous set of features.

For this reason, a common practice is to extract communication flows from raw monitored network traffic using tools like CIC Flow Meter (Draper-Gil et al. 2016) or NFStream (Aouini and Pekar 2022). This approach aggregates packet information from the same session, preventing the sparsity of the dataset and reducing the volume of training data. In practice, this approach consists of identifying the packets exchanged between two communicating machines from the opening of a connection to its closing. The packets of each flow are analyzed to extract a set of features (e.g., protocol used, number of packets exchanged, flow duration) that can be shared between each flow. The final dataset is a multivariate time series in which each data point conveys information about a flow between A and B, starting at time t. Given that, the Intrusion Detection problem is to identify flows where an ongoing attack is present.

This approach influences the strategies used for training and testing, as well as how detectors behave when deployed in real-world scenarios.

During training, the detector processes communication flows distilled as single data points (each flow is a data point). Necessarily, during testing, the detector performance is evaluated without considering the time the attacker has been operating against the system.

The practical implication is that, in the hypothetical case of a real deployment, a detector trained on flows would wait until the end of the communication to determine whether the flow is anomalous. This means that the attacker has completed his activities, possibly also meaning the exploit was complete. Additionally, an attacker may simply keep the communication flow active to avoid the detector operating.

In this paper, we discuss how attack latency provides valuable insight into the detector's realistic capabilities and enables performance adjustments for a trained model.

We propose an approach to build datasets for ID that report network packets organized in sequences of normal operation and anomalous operation.

Unlike the flow-based approach, our work requires datasets such as (Puccetti et al. 2023), where all communication packets are recorded, and sequences of normal data points alternate with sequences where the attacker is active. This enables us to calculate the time required to detect an attack sequence targeting the system.

Because such structured datasets are unavailable, we manipulate the DoS/DDoS-MQTT-IoT dataset from (Alatram et al. 2023) to create a sequence-based dataset that, instead of grouping traffic into flows, combines attack and normal packets. This produces attack sequences of data points that we can analyze for latency. In Section 4.3, we detail the process used to construct the dataset from raw network data, and highlight how it differs from a traditional flow-based approach.

Additionally, we use an error detection dataset, which traditionally presents labeled sequences.

2.3 Anomaly detection solutions

Models can be classified into two categories, based on their ability to exploit the temporal context from the training set (Chandola et al. 2009).

Unordered data points Solutions belonging to this category do not benefit from the temporal dimension, i.e., there is no difference between shuffled data points or data points collected in a (temporal) sequence. The most researched approach is to use a machine learning-based detector that relies on the features from a single data point x_t and that is focused on detecting punctual anomalies. In this case, features related to the temporal or sequential order of data points are not exploited; on the contrary, they should be dropped when training to avoid acting as a label (Puccetti et al. 2023).

Amongst the most noticeable works, Bhattacharya et al. (Bhattacharya et al. 2020) propose an ID based on XGBoost and principal component analysis (PCA). XGBoost is a supervised learning algorithm that combines the prediction of an ensemble of regression trees (Chen and Guestrin 2016). Other XGBoost-based solutions are proposed by (Verma et al. 2018; Xu et al. 2021). Random Forest is another popular ensemble-based solution that relies on supervised learning and bagging (Breiman 2001). A random forest is a classifier that aggregates the decisions of a decision tree classifier ensemble. Similarly, Extremely Randomized Trees (Geurts et al. 2006) construct trees over every observation in the dataset but with different subsets of features.

Isolation Forest (Liu et al. 2008) is an unsupervised approach viable for intrusion detection that consists of explicitly isolating anomalies (Peng et al. 2018). The anomalies are detected from those instances that have short average path lengths on the isolation trees. Other unsupervised approaches are K-Means (Hamerly and Elkan 2003; Ji et al. 2022), FastABOD (Kriegel et al. 2008), and LOF (Breunig et al. 2000). All these approaches are frequently used to evaluate anomaly detectors (Muda et al. 2011; Huang et al. 2013; Peng et al. 2018; Verma et al. 2018; Zoppi et al. 2021; Puccetti et al. 2023; Taheri Sarteshnizi et al. 2024).

In this paper, we will use XGBoost, Extremely Randomized Trees, and Isolation Forest.

Exploit ordered data points Solutions in this category are intended to exploit the temporal dimension or the sequence of data points, deriving knowledge from the variation of feature values. There are two main approaches.

One approach is to craft new features that synthesize the information conveyed by consecutive data points. This way, each data point x_t encodes information about the previously observed values, allowing for contextual and collective anomaly detection. Approaches to crafting temporal features are mostly explored in the time series classification (TSC) literature, but can be adapted to multivariate point-based classification.

As reported by Ji et Al. (Ji et al. 2022), some of the approaches adopt statistical features in the time domain, such as the mean and variance (Baydogan et al. 2013). Similarly, other methods exploit structural features of time series, such as trend, seasonality, and periodicity (Wang et al. 2006; Wu et al. 2021). Also, some researchers transform the features in the

frequency domain (Batal and Hauskrecht 2009; Chaovalit et al. 2011). Distances between time series have been adopted as features for TSC (Kate 2016; Ji et al. 2022).

The approach of synthesizing network flows from raw network captures, reported in Section 2.2, is a way to craft temporal features. Each of the flows is described using common features, e.g., number of packets per second and the duration of the flow.

Another approach is to use algorithms such as LSTM (Long-Short Term Memory (Gers et al. 2000)) that employ recurrent neural networks (RNN (Hopfield 1982)) specifically crafted to consider the evolution of feature values. In this case, the approach is to predict the state x_t of the data point at time t (i.e., the values of the data point features observed at time t) using previous observations. This approach implies the definition of a threshold value that is compared to the difference between the predicted data point and the ground truth to decide if the data point is anomalous (Karim et al. 2018). An example of this approach is the model.[1] Amongst noticeable examples, Qin et al. (Qin et al. 2021) propose an LSTM-based anomaly detector to detect the abnormal behavior of the controller area network (CAN) bus under tampering attacks. A similar approach is proposed in (Au Yeung et al. 2020) and applied to financial time series.

Many approaches rely on LSTM to extract temporal features and on autoencoders to estimate the input time sequence. In this case, the loss between the ground truth sequence and the estimated one is compared to a threshold value: if the loss exceeds the threshold, the sequence is considered anomalous (Lindemann et al. 2020). The model1, referred to as LSTMCD, combines convolutional layers (C) used for initial feature extraction, LSTM layers for capturing temporal dependencies in sequence data, and dense layers for final classification (D).

In this paper, we will use LSTM and LSTMCD. Additionally, we craft temporal features by computing the mean, variance, and linear regression slope of numeric attributes over different time intervals to investigate their impact on the performance of algorithms for unordered data points (as discussed in RQ5, Section 5.2).

This paper examines the role of attack latency in both algorithms designed for unordered data and those incorporating ordered data or temporal features, to explore how attack latency can be exploited to highlight performance differences between these models and to assess their relative effectiveness in detecting different categories of attacks.

2.4 Metrics and evaluation

Anomaly Detectors are typically evaluated based on state-of-the-art metrics for classification. This approach is applied regardless of the system, the type of anomalies targeted, and the possibility of taking advantage of temporal features. The most used approach measures how many data points are correctly classified in terms of True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN).

Different metrics can be derived from the above. Accuracy measures the correct predictions in a classification task and is defined as:

$$A = \frac{TP + TN}{TP + TN + FP + FN}$$

[1] https://kaggle.com/code/robertlangdonvinci/time-series-classifier-keras-conv1d-lstm-dense

Accuracy is an excellent summarizing metric, but it is misleading in the case of unbalanced datasets when the number of normal data points differs significantly from the number of anomalous data points. In this case, it should be replaced or complemented by other metrics.

Other notorious metrics focus on the detection capability and do not include TNs. Recall (also called coverage) measures the fraction of anomalies that are detected, and it is measured as:

$$R = \frac{TP}{TP + FN}$$

Precision measures the fraction of detections that are anomalies, and it is measured as:

$$P = \frac{TP}{TP + FP}$$

Precision and recall can be used to plot the precision-recall curve, which calculates the trade-off between precision and recall for different detection thresholds.

Still related to precision and recall, the f1-score is the harmonic mean of the precision and recall:

$$F1 = \frac{2 \times (P \times R)}{P + R}$$

The ROC curve plots the probability that the model will correctly rank a randomly chosen positive instance higher than a random negative one (Haque and Mineno 2018), representing the trade-off between recall and false positive rate, where the false positive rate is:

$$FPR = \frac{\mathrm{FP}}{\mathrm{FP} \ + \ \mathrm{TN}}$$

We review in Table 1 the metrics utilized for evaluating the performance of detectors in recent conference papers focusing on anomaly detection. Our review includes publications since 2021 from the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), the IEEE International Symposium on Software Reliability Engineering (ISSRE), and the USENIX Security Symposium. We delve into papers that explore datasets collected from various system features, obtained through diverse system monitoring tools. The review works are well-aligned with the most famous metrics previously reviewed. In addition, attention is given to the time required to analyze the input and prompt a decision (Striki et al. 2009; Tsikoudis et al. 2016; Lapolli et al. 2019; Chen et al. 2021; Jha et al. 2022; Li and Jung 2023). The related works aim to maximize the number of data points correctly classified. Aside from Table 1, a significant exception is proposed by Wang et al. (Wang et al. 2022a), where the concept of correct detection is bound to the concept of sequence of data points: a sequence of anomalous data points is correctly detected if at least one of the data points is identified as anomalous. A time-related approach is shown by T. Markovic et al. (Markovic et al. 2023), by measuring the size of the input windows required

Table 1 Analysis of papers on anomaly detection at security and dependability conferences. Only papers that deal with features represented as tabular data points are considered. Papers dealing with anomalies in images or text are left out

Paper	Venue	Scope	Metrics
(Jha et al. 2022)	DSN	ED	P, R, F1, Lead Detection Time
(Wang et al. 2022a, b)	DSN	ED	P, R, F1
(Dayaratne et al. 2022)	DSN	ID	P, R,F1, FPR
(Alharthi et al. 2021)	DSN	ED	P, R, F1, MCC
(Yuan et al. 2021)	DSN	ID	P, R, TPR, FPR
(Xu et al. 2021)	DSN	ED	P, R, F1
(Zhao et al. 2019)	DSN	ID	A
(Wang et al. 2022a, b)	ISSRE	ED	P, R, F1
(Zhang et al. 2022)	ISSRE	ED	P, R, F1
(Jia et al. 2021)	ISSRE	ED	P, R
(Zhang et al. 2021)	ISSRE	ED	P, R, F1,ROC
(Alsaheel et al. 2021)	USENIX	ID	P, R F1, ROC
(Chen et al. 2021)	USENIX	ED	R, avg. time
(Downing et al. 2021)	USENIX	ID	P, R, FPR, ROC
(Izhikevich et al. 2021)	USENIX	ED	A, proc. time
(Fu et al. 2021)	USENIX	ED	P, R, FPR
(Tang et al. 2021)	USENIX	ID	TPR, FPR

to detect an attack. This can be interpreted as an estimation of latency, even if it is bound to the investigation of the window size.

Unlike prior work, this paper introduces an evaluation methodology and metrics specifically focused on attack latency. By doing so, we provide an additional perspective on detector capabilities that incorporates timely detection and enables calibration for real-world deployment.

3 Measuring detection latency

In this section, we present our approach to measuring detection latency, define the metrics used, and provide a discussion supported by Fig. 1.

We define a time series as:

$$X = \{\langle x_t \rightarrow x_{t+1}\rangle | x_t \in X, t \in [0, T]\}$$

where x_t is the data point collected at time t, and x_{t+1} is the next data point after x_t, collected at time $t+1$ (Li and Jung 2023). An anomaly detection dataset based on time series can exploit the temporal dimension by having multiple sequences of data points.

We define a sequence as the concatenation of contiguous data points collected while the system is operating in the absence of attacks (we call it normal operations) and data points collected after an attack is injected (we call it anomalous operations). More formally, we define a sequence of length N and an injection instant t_i as the concatenation of two time series *S1* and *S2*:

$$\begin{array}{l} S1 = \{\langle s_t \rightarrow s_{t+1}\rangle | s_t \epsilon S, t\epsilon [0, T_{i-l}]\} \\ S2 = \{\langle s_t \rightarrow s_{t+1}\rangle | s_t \epsilon S, t\epsilon [T_i, T_N]\} \end{array},$$

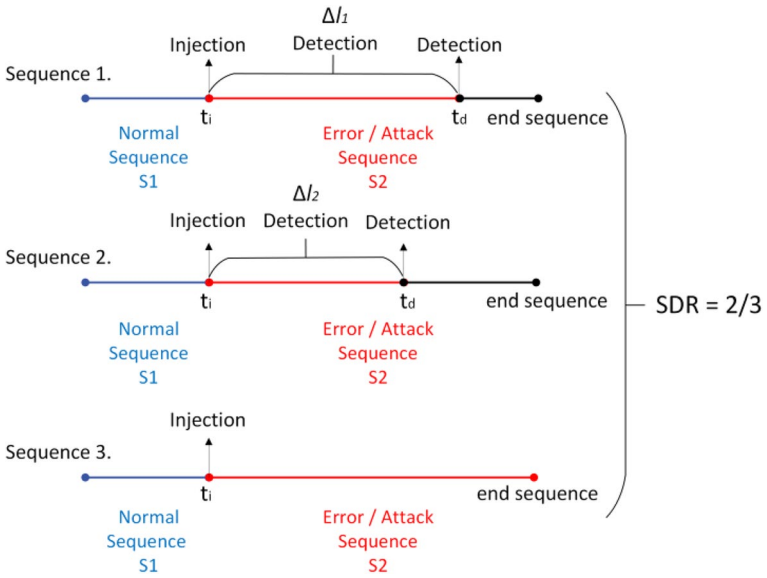


Fig. 1 Overview of our approach to perform latency measurements

where data points $[t_0; t_{i-1}]$ are collected during normal operation of the system, and data points $[t_i; t_N]$ are collected during anomalous operations.

The detection of an attack in a sequence *S2* means correctly classifying a data point x_t that belongs to the anomalous operations.

To present the metrics we will use in this paper, we need to formalize the definition of attack latency presented in Section 2.3 and from (Puccetti et al. 2024).

Given: i) a sequence S, and an attack or error injected at time t_i, which marks the beginning of the anomalous operations, ii) s_i the data point of S collected at time t_i, and iii) x_d the value at time t_d when detection is performed ($t_d \geq t_i$), we define latency as:

$$\Delta l = t_d - t_i$$

In practice, we can measure latency as a time interval, or as the number of data points, labeled as 1, that occurred between the two data points s_i and s_d.

In case the attack is not detected, the latency cannot be measured: this is the situation in which the attack succeeds.

For this reason, we compute the sequence detection rate SDR, expressed as the number of sequences where an attack is detected over the total number of sequences. In other words, SDR = *detected sequences*/*total sequences*. We can measure the average latency ΔL over the N sequences, that is:

$$\Delta L = \sum_{i=0}^{N} \Delta l_i / N$$

In Fig. 1, we represent 3 sequences, 2 of which are detected sequences, achieving SDR = 0.67. The two detected sequences allow measuring the average latency $\Delta L = (\Delta l_1 + \Delta l_2)/2$.

Let us discuss some practical implications. A short latency can be easily achieved at the cost of a high FPR, which can be obtained by setting a low detection threshold (used by the anomaly detectors to discern whether a data point is normal or anomalous). This would make the system unusable because of too many and too frequent false alarms. The natural implication is that the measure of latency needs to be weighed against the FPR.

In an analogous way to how the precision-recall curve and the ROC are computed, we measure the latency at different FPR thresholds: this presents the tradeoff between latency and FPR. We expect that by increasing the FPR, the detection latency will be reduced because the detector will be more suspicious and tend to increase the data points classified as anomalous. The relationship between FPR and latency can be used to set the best trade-off based on the system requirements and the application domain of the detector.

The opposite (computing the FPR given a target latency) can also be measured, but it is not effective in explaining the performance of a detector. We justify our statement with the help of Fig. 2. Let us suppose that we set a target latency $\Delta l_3 = t_3 - t_0$. According to Fig. 2, all data points with confidence lower than 0.2 should be classified as normal, and those with confidence higher than 0.2 should be classified as anomalous. Using this information, the FPR can be computed easily. However, with such a confidence threshold, the first anomalous data point correctly detected corresponds to t_1, so the actual latency would be $\Delta l_1 = t_1 - t_0$, which is smaller than Δl_3. This means a lower latency Δl_1, a higher confidence threshold of 0.3, and correspondingly, a lower FPR better describe our detector.

We remark that these metrics have practical value only if the dataset in use is composed of sequences of normal and anomalous operations, as previously introduced. They are not useful when the dataset does not have these characteristics. For example, shuffling the data points is often necessary, due to the characteristics of the datasets (Muda et al. 2011; Peng et al. 2018; Verma et al. 2018; Puccetti et al. 2023): the realistic alternation of normal and anomalous behavior is compromised, and the measurement of latency would be unrealistic.

In the following, we show through an experimental campaign how we use this trade-off to study and evaluate a detector.

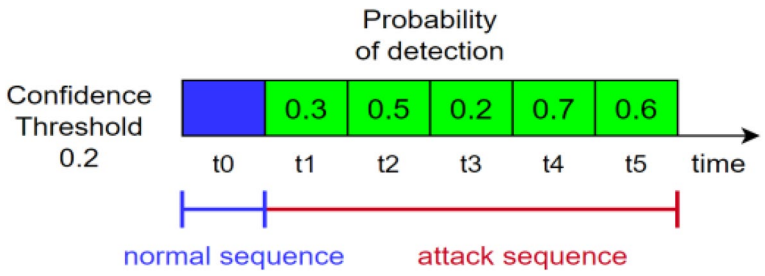


Fig. 2 Relationship between latency values and confidence

4 Experimental campaign

This section outlines the experimental setup used to evaluate our proposed approach. Section 4.1 provides an overview of the key steps involved in the evaluation process. The subsequent subsections offer detailed explanations of each step: dataset selection and its characteristics (Section 4.2), the methodology for creating an additional dataset (Section 4.3), selection and description of the detectors (Section 4.4), the process for crafting temporal features (Section 4.5), training procedures (Section 4.6), and training and testing along with the evaluation metrics used (Section 4.7).

4.1 Experimental steps

In this section, we introduce the steps of our experimental campaign, schematized in Fig. 3.

Step1. Public datasets selection We use two publicly available datasets for the evaluation: ROSPaCe (Puccetti et al. 2024) and Arancino (Zoppi et al. 2023b). The first is a dataset for intrusion detection composed by monitoring an embedded system under normal behavior and attack. The second is specifically designed for error detection and represents an embedded system in an Internet of Things (IoT) setting. Both datasets are time series-based and comprise a set of sequences (see Section 2.2) of variable length. The datasets are detailed in Section 4.2.

Step2. Additional dataset creation We compose an additional semi-synthetic dataset, *DoS MQTT*, collecting and processing the raw data of the DoS/DDoS-MQTT-IoT dataset (Alatram et al. 2023), as detailed in Section 4.3. The raw data are in *pcap* (packet capture) format, a common file type used to store network traffic data by capturing the packets transmitted over a network. Each row in the pcap file represents a network packet captured at a specific timestamp t, including both header and payload information.

Step3. Detectors selection: We select 5 representative detectors from the state-of-the-art solutions reviewed in Section 2.3, organized into algorithms for unordered data points and algorithms for ordered data points. The algorithms are detailed in Section 4.4.

Step4. Temporal features crafting We enhance the performance of unordered-point algorithms selected in step3 by generating temporal statistical features (mean, variance, slope) over multiple time intervals, enriching the dataset without losing granularity. Using fea-

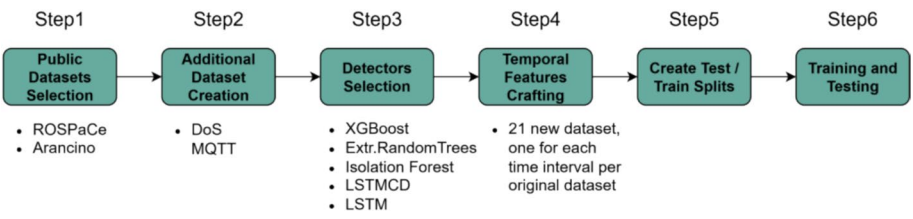


Fig. 3 Steps of the experimental campaign

ture selection techniques, we retain the most informative features per dataset. This process results in 21 new datasets, one for each time interval per original dataset. The methodology is detailed in Section 4.5.

Step5. Create test/Train splits We split the datasets into train and test sets.

Step6. Training and testing Initially, we train and test the selected detectors, evaluating them using both standard state-of-the-art metrics and our proposed metrics. We then retrain and evaluate the best-performing model for ordered data points using the dataset enriched with temporal features. A summary of the evaluation metrics is provided in Section 4.7, and the results are discussed in Section 5.

4.2 Datasets selection

ROSPaCe (Puccetti et al. 2024) is a dataset for intrusion detection composed by performing penetration testing on SPaCe, an embedded cyber-physical system built over Robot Operating System 2 (ROS2). The SPaCe system is designed to orchestrate and optimize public mobility, improving the user experience and preventing security violations. It is implemented as an embedded cyber-physical system that relies on data collected by sensors to manage and determine the operational conditions of the vehicles in real time. The system analyzes the collected data to establish the occupation of the vehicles and notifies users and administrators of the system to optimize the availability of the service. It also provides detection capabilities to spot any dangerous situations, like damage to vehicle equipment.

Features are monitored from three architectural layers: the Linux operating system, the network, and the ROS2 services. It collects a total of 6 attacks, listed in Table 2. Three attacks are provided by the tools *nmap* and Metasploit: i) the two Denial of Service (DoS) flooding attacks *mflood*, and *nflood*, respectively exercised by Metasploit and nmap, and ii) the discovery attack *ndisco* that explores the network architecture. Three attacks instead exploit ROS2 vulnerabilities: i) *ros_rec* is a discovery attack that extrapolates information on the running nodes, ii) *rosc* is a DoS attack that causes incorrect handling of ROS2 messages, and iii) *ros_ref* redirects the node to a malicious server.

The dataset is organized as a time series in which sequences of normal (attack-free) operations and sequences when attacks are carried out, in addition to the normal operations, are alternated. The goal of this strategy is to reproduce multiple scenarios of an attacker trying to penetrate the system. Noteworthy, this allows measuring the time to detect an attacker and the number of malicious activities performed before detection. ROSPaCe includes 30,247,050 data points (78% attacks, 22% normal), and 482 columns excluding the label, for a total of 40.5 GB. We use the lightweight version of the ROSPaCe dataset that is reduced to the 61 most informative features (30 features from the Linux operating system, 5 from ROS2 services, and 25 from the network). The total number of sequences is 1173.

Arancino (Zoppi et al. 2023b) is an embedded system for Internet of Things applications that is composed of two main parts: a microprocessor and a microcontroller. The dataset, presented in (Zoppi et al. 2023b), provides data about how the Arancino device behaves in the case of anomalies due to different injected errors. The system is monitored while executing normal operations and while injecting 9 errors related to unusual resource consumption. Referring to the terminology in Table 2, *disk* and *mem* constitute an unusual usage and

Table 2 ROSPaCe, Arancino, and DoS MQTT datasets. Columns indicate i) attacks and errors injected, ii) the number of data points, iii) the number of sequences (seq. num.), along with the corresponding average (avg. dur), iv) minimum (min.dur.), and maximum durations (max. dur.) in seconds, and their respective standard deviations (std.dur.), v) the average (avg. len.), minimum (min. len.), and maximum sequence lengths (max. len) in terms of number of data points, along with the associated standard deviations (std. len)

	Attack/error	Data points	Sequence number	Average duration	min. duration	max. duration	std. durat	Average length	min. length	max. length	std. length
ROSPaCe	mflood	4,899,066	294	60.709	59.000	160.344	8.233	43,303	812	171,799	22,562
	ndisco	166,786	362	21.890	19.665	26.20	1.154	1377	1180	8248	232.80
	nflood	2,664,621	20	29.848	29.658	29.960	0.079	398,655	467,044	40,534	16,319
	rosc	1946	101	1.2415	1	40.232	6.817	51	2	150	20.477
	ros rec	133,901	214	12.725	9.397	60.26	11.54	1907	1295	4873	251.54
	ros_ref	1321	182	7.258	1	40.2	8.698	24	1	99	7.157
Arancino	disk	9396	619	14.18	14	17	0.71	15.18	15	18	0.71
	mem	9279	581	14.97	14	17	1.4	15.97	15	18	1.4035
	redr	9174	604	14.18	14	17	0.73	15.18	15	18	0.73
	deadl	8952	596	14.02	14	17	0.24	15.02	15	18	0.24
	cpus	8805	582	14.13	14	17	0.6	15.13	15	18	0.6
	httpr	8760	585	13.97	14	17	0.27	14.97	15	18	0.27
	proc1	8694	579	14.01	14	17	0.21	15.01	15	18	0.21
	redw	8547	572	14.94	14	17	0.41	13.94	12	15	0.41
	proc2	8343	554	14.05	14	17	0.42	15.05	15	18	0.42
DoS MQTT	bf1	1,846,728	40	45.76	10	79.72	5.95	41,169	2278	78,461	20
	delay	663,777	36	45.59	11.37	79.50	6.5	12,292	1510	33,247	18
	syn	310,823	44	44.57	10.21	79.47	5.16	6759	1626	15,343	22
	sub	1,217,708	50	44.54	9.97	78.75	5.94	22,684	1520	48,079	25
	will	1,794,108	56	45.84	10.63	79.91	5.84	29,727	5309	54,934	28

access to the hard drive and the RAM, *redr, and redw* are erroneous read and write operations towards the Redis database, *deadl* is the misuse of semaphores in a multithreading application, *cpus* is the unusual utilization of the CPU, *httpr* is anomalous network traffic generated by a faulty application, and *proc1* and *proc2* are two delayed processes of the Raspbian operating system.

The injection of the error remains active for 5 s, then the system waits for a total of 10 s of cooldown, in which memory is freed, and no errors are injected. It assumes that at the end of 5 s, the system fails; that is, we are required to detect errors before this period. The dataset is composed of 3 separate labeled datasets obtained by running the monitoring methodology in 3 different network environments (Zoppi et al. 2023b). The dataset counts 154,000 observations and 276 features other than the timestamp and a label, i.e., normal or any of the 8 errors. The total number of sequences is 5273. After preliminary runs, we decided to augment the Arancino dataset to facilitate the detectability of its anomalous operations within each sequence. We replicate 3 times each anomalous data point x_i and place the copied data point between the original x_i and the subsequent data point x_{i+1}. Finally, we rearrange the timestamp to include the new data points, obtaining consecutive data points at 1-s intervals as in the original dataset. The structure of the datasets is summarized in Table 2.

4.3 Additional dataset creation

We compose **DoS MQTT**, a semi-synthetic (Stojanović et al. 2020; Skopik et al. 2014) dataset for intrusion detection as introduced in Section 2.2. We leverage raw network traffic monitored during the communication of a cyber-physical system. For this purpose, we utilize the DoS/DDoS-MQTT-IoT dataset (Alatram et al. 2023), which provides several hours of raw network traffic captured in *pcap* files. Normal and attack traffic are collected during dedicated simulations and stored in distinct files.

The dataset is collected by monitoring a cyber-physical system consisting of Raspberry Pi sensors overseeing an industrial facility. Communication between devices is implemented using the MQTT protocol, which follows a publish-subscribe paradigm and connects 22 machines among publishers, subscribers, and the broker.

We select 5 attacks from those included in the dataset (Alatram et al. 2023):

CONNECT Flooding Attack (*bf1*): A large volume of CONNECT packets is sent to the target machine to overwhelm it during authentication requests.
CONNECT Flooding Attack (*delay*): This forces a certain period of delay between the TCP three-way handshake and the first packet regarding the MQTT, which is a CONNECT packet.
TCP SYN Flooding Attack (syn): exploit the TCP protocol's state retention mechanism. This attack exploits the three-way handshake technique of the TCP protocol, leading to the maximum number of simultaneous TCP half-connections.
Invalid Subscription Flooding Attack (*sub*): with valid credentials, an attacker can either overwhelm the server (broker) using invalid subscriptions or publish requests to the subscriber, resulting in the consumption of the broker's CPU resources.
CONNECT Flooding with WILL Payload Attack (*will*): by sending a CONNECT packet for authentication requests, an attacker can consume the broker's CPU resources by increasing the CONNECT packet size through piggybacking a WILL payload.

The flow-based approach typically refines the raw network data to obtain labeled network flows. Differently, we report all communication packets, and we identify attack sequences to enable the computation of attack latency. The dataset construction follows these steps:

- *Selection of Normal and Attack Traffic*: we select pcap files containing normal traffic from two distinct days to form the training and test sets, respectively. For each attack, we randomly select packets from the files to include in the training and test sets.
- *Extraction of attack* sequences: for each attack file, we define two randomly timed intervals and extract the corresponding data points to generate two attack sequences.
- *Traffic Conversion*: normal and attack traffic are converted to *csv* format while preserving all features originally present in the pcap files. We compute the correlation matrix and eliminate features with a correlation of 99% or higher. Then, we apply the Laplacian unsupervised scoring method to identify features that best characterize normal traffic, reducing the total number to 100. Details on feature selection are reported in our repository.
- *Merging Attack Sequences into Normal Traffic*: we select a random insertion point within the normal traffic to merge the first attack sequence. The timestamps of the attack sequence are adjusted to ensure they follow the insertion point while preserving the original relative timing. Subsequent attack sequences are inserted after the previous one with random time intervals between each sequence. For the training, we use an interval ranging from 1 to 10 min, while for the test, it is from 1 to 5 min. Normal data points are labeled as 0, while attack data points are labeled as 1. We maintain indexes to track the position of each attack sequence within the normal data.

In Fig. 4, we illustrate how our processing method and resulting dataset differ from the traditional flow-based approach. To keep the example simple, we select two pcap files: one containing normal traffic and the other containing attack traffic.

We begin by applying a flow-based approach using the NFStream tool (Aouini and Pekar 2022), which extracts flow-level features from the raw network traffic. For both the attack and normal pcap files, we extract and report two representative flows, along with a subset of the features generated by NFStream. The resulting csv files are then merged into a single dataset, with each entry labeled as attack or normal based on its origin.

Next, we apply our methodology to the same.pcap files, emphasizing the finer granularity of our approach by using consistent colors to highlight packets that belong to the flows previously extracted with the flow-based method. We first convert the normal traffic from the Normal pcap into a csv table, where each row represents a single packet (with blue and green entries representing normal traffic). We then identify two distinct packet sequences in the Attack pcap, each corresponding to an attack flow, separated by approximately 16 min and 30 s and having different durations. For clarity, we display only the first three packets from each sequence. These two attack sequences are then merged into the normal traffic stream. We insert the first packet of the first attack sequence (marked in red) at timestamp 04:28:20.456, which is 1.13 min after the last packet in the preceding segment of normal traffic (in blue). The timestamps of the remaining packets in the first attack sequence are adjusted to follow this insertion point, while preserving their original inter-packet timing.

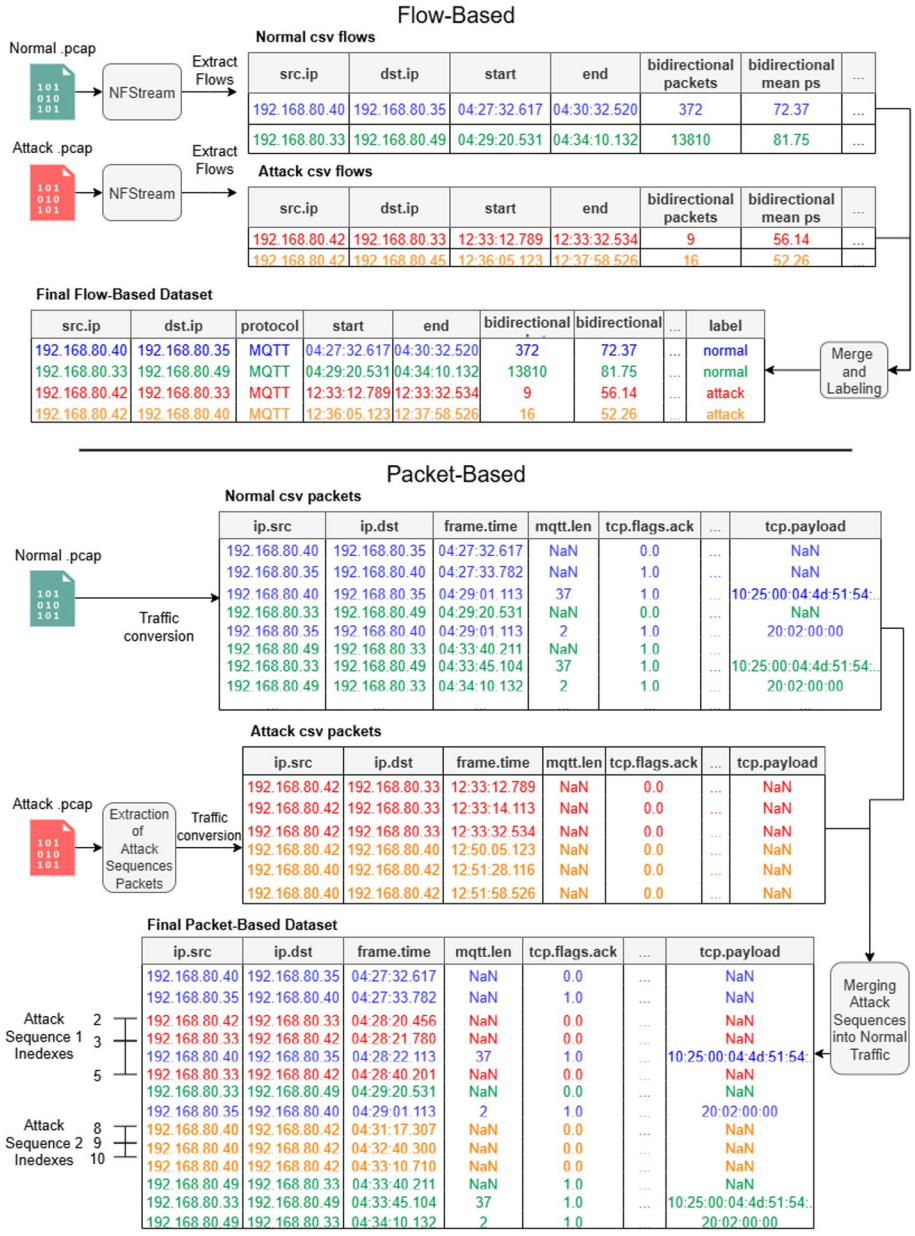


Fig. 4 Comparison between the traditional flow-based processing (top) and our packet-level sequence-based approach (bottom). Two.pcap files, one normal and one attack, are processed using NFStream to extract representative flows. The Packet-Based method highlights finer-grained packet sequences, shows how attack packets are injected into normal traffic with preserved timing, and enables precise indexing for latency analysis

The same procedure is applied to the second attack sequence. Finally, we extract the indices of all packets from both attack sequences, enabling straightforward access and facilitating latency computation. We apply the same procedure to the second attack sequence. Finally, we extract the indices of the packets from both attack sequences to facilitate direct access and to enable latency computation as introduced in Section 3.

The final dataset consists of 52% of normal data points and 48% of attack data. The distribution of labels is reported in Table 2, along with the information about the attack sequences.

4.4 Detectors selection

We validate our approach by running 5 different machine-learning algorithms on the three datasets.

Algorithms for unordered data points We use XGBoost, Extremely Randomized Trees, and Isolation Forest to detect punctual anomalies. XGBoost is a supervised decision-tree-based ensemble machine learning algorithm that uses a gradient boosting algorithm (Chen and Guestrin 2016; Wang and Lu 2020). We configure XGBoost with the following parameters: 10 gradient boosted trees, maximum tree depth for base learners set to 20, and learning rate of the boosting algorithm set to 1. We use the binary logistic loss metric.

Extremely Randomized Trees (*ExtraTreeClassifier* in the following) (Geurts et al. 2006) is a random forest model based on ensembles. As in random forests, a random subset of candidate features is used, but thresholds are drawn at random for each candidate feature, and the best of these randomly generated thresholds is picked as the splitting rule. The training is performed in a supervised fashion. We use the implementation available at (Pedregosa 2011) and we configure the model to run with 100 estimators, which is the number of trees in the forest.

Isolation Forest (Liu et al. 2008) is an unsupervised algorithm constructed as an ensemble of Isolation Trees, where each tree organizes data points as nodes. The underlying assumption is that anomalies represent rare events with feature values significantly deviating from those of expected data points. Consequently, anomalies are more likely to be isolated closer to the root of the tree rather than the leaves. This logic implies that a data point can be effectively isolated and subsequently classified based on its distance from the root of the trees within the forest. We use the implementation available at (Pedregosa 2011). We set in the case of ROSPaCe a contamination parameter, indicating the assumed proportion of outliers in the dataset, to 0.25, which resembles the percentage of normal data points in the training set.

Algorithms that exploit ordered data points We train an LSTM model composed of a convolutional neural network and two LSTM layers that process the input data points sequentially. Stacking multiple convolutional layers, the model learns hierarchical representations of the input, taking advantage of the order in which data points are provided. The lower layers capture low-level temporal patterns, while higher layers learn more abstract and complex patterns. The output of the convolutional network is passed as input to the LSTM

model, which can capture temporal dependencies from time-ordered data points. We set *return_sequence = True* for the first LSTM layer to return the full output sequence, while we set it as False for the second, to output only the label prediction. The implementation can be found in (Wijesooriya 2024). The first layer is composed of 64 units, while the second is composed of 32. We use SGD as an optimizer with a learning rate configured to 0.001, decay equal to 10^{-8}, and momentum set to 0.9.

Additionally, we train an improved model that combines machine learning techniques and LSTM technology for sequence classification tasks (LSTMCD).[2] The model architecture consists of convolutional layers, followed by LSTM (Long Short-Term Memory) layers, and dense layers. The idea is to use the convolutional network to capture local patterns in the input sequences, identifying features that are important for classification. We use two 1D convolutional layers, each one followed respectively by a batch normalization and a ReLU activation. The first convolutional layer has 64 filters with a kernel size of 5, and the second has 64 filters with a kernel size of 3. Each of the LSTM layers is configured with 64 units. The output of the LSTM layers is fed to the two dense layers to improve the understanding of the sequences. We configure these layers to have 100 units, while the subsequent dense layer restricts the dimension to 50 units. The output is fed to the final dense layer of two units, which use the softmax function to predict the final binary classification outcome.

To execute LSTM and LSTMCD, it is required to preprocess the train, test, and validation sets to obtain an input composed of 10 subsequent data points, obtained by selecting for each point x_t the points between x_t and x_{t+9}.

We report in our repository the information on the complexity of the selected models, for all the models.

4.5 Temporal features crafting

We create temporal statistical features from the dataset's numerical features to improve the best algorithm for unordered data points. We do not apply this solution to algorithms for ordered data points, as they are originally structured to extract information from previously seen data.

For this purpose, we adopt the technique from the TSC field reported in Section 2.3. The approach in (Wang et al. 2006; Wu et al. 2021) is not applicable as our datasets have an insufficient duration to show patterns as seasonality. The frequency (Batal and Hauskrecht 2009; Chaovalit et al. 2011) and distance-based (Kate 2016; Ji et al. 2022) approaches can be adapted, but we believe it mainly helps compare univariate time series by their distance or waveform. CIC Flow Meter (Draper-Gil et al. 2016) is not applicable because we aim to preserve granularity and compute attack latency.

The approach from (Baydogan et al. 2013) can instead be easily adapted to multivariate time series. It consists of dividing a univariate time series into subsequences using time intervals and computing the mean, variance, and linear regression slope for each of them. The aggregated information is used to describe the original time series in a reduced form.

[2] https://kaggle.com/code/robertlangdonvinci/time-series-classifier-keras-conv1d-lstm-dense

We define different time intervals, such as 5, 10, 30 s, and 1, 5, 10, and 30 min. We group data points into the same set if their timestamps fall within that interval. For each set, we compute the mean, variance, and linear regression slope.

Different from (Baydogan et al. 2013), we do not eliminate data points. We merge the newly created features with the original dataset, adding features to each data point, and ensuring that the number of data points remains unchanged. This way, the dataset is enriched without losing granularity. The approach is depicted in Fig. 5.

We use the correlation matrix to eliminate redundant features. Then, we use the Laplacian unsupervised score to determine the features that better describe the system behavior, and we select the best n ones, with n varying depending on the dataset. We ensure that each feature reflects the system's evolution over time, e.g., *RAM usage* and *TCP Window Size*.

In the case of the Arancino dataset, all the features are numerical, so we directly select the best 20. In the case of ROSPaCe, we select the best 10 numerical features, keeping the 10 categorical features in the dataset. The same approach is applied to the DoS MQTT dataset.

We generate a new dataset for each of the 7 intervals we chose. This leads to 7 new datasets for each original one, for a total of 21 new datasets.

4.6 Create test/Train splits

In the case of Arancino and ROSPaCe, we shuffle the order of sequences to create a test and a training set. Note that we do not alter the order of data points within each sequence. We split the two datasets into train, validation, and test sets, using 60% of sequences for the training, 30% for the test, and 10% for validation.

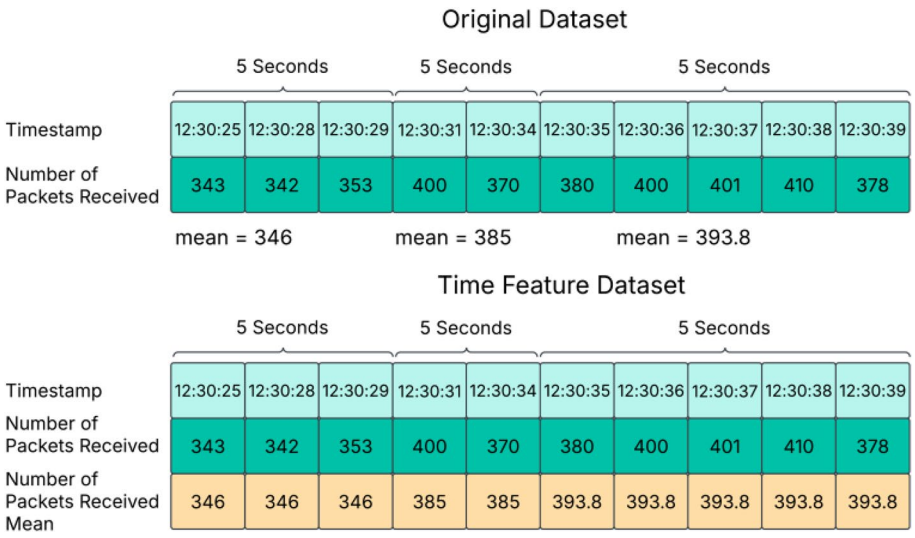


Fig. 5 Illustration of the temporal statistical features extraction process using a two-feature dataset (Timestamp and Number of Packets Received). For every 5-s interval, the mean, variance, and linear regression slope of the Number of Packets Received are calculated and assigned to all data points within that interval, forming the dataset

In the case of the DoS MQTT dataset, we preserve the original order of sequences since the randomization of attack sequences is performed when creating the dataset, as detailed in Section 4.3.

4.7 Training and testing

We applied a grid search strategy to identify the optimal hyperparameter configuration for each model. All models were trained and evaluated on the same predefined train–test splits to ensure consistent and fair comparison. The same evaluation metrics, random seed, and computational environment were used throughout, and the number of hyperparameter combinations was kept consistent across models.

For XGBoost, we tuned n_estimators [10, 100, 200, 300], max_depth [10, 20, 30, 40], and learning_rate [0.001, 0.01, 0.1, 1]. The best configuration was: ROSPaCe and Arancino, n_estimators=10, max_depth=20, learning_rate=1.0; DDoS MQTT, n_estimators=100, max_depth=20, learning_rate=0.001. For Extra Trees, tuning the same parameter ranges yielded the best performance with n_estimators=100 and max_depth=10. For Isolation Forest, the best result was obtained on ROSPaCe with n_estimators=300 and contamination=0.25. For LSTM and LSTMCD, we used a learning rate scheduler that reduced the rate by 0.5 down to 0.0001 based on validation loss, avoiding extensive grid search. Patience values [1, 3, 5, 10] were tested and had no consistent impact on performance but helped shorten training time. For the LSTM model, we experimented with different dropout values [0.2, 0.3, 0.5] However, increasing the dropout rate consistently led to poorer results across all datasets.

We summarize in Table 3 the metrics we use for the evaluation of the selected models. In addition, we compute the ROC Curve and the precision-recall Curve, which calculate the trade-off for different detection thresholds, respectively, between false positive rate and recall, and between precision and recall. Results are reported in Section 5.

5 Results

In this section, we present and analyze the results of our experimental evaluation. Section 5.1 examines the performance of the detectors using standard state-of-the-art evaluation metrics. Section 5.2 evaluates the detectors using our proposed latency-oriented metrics, discusses the relationship between FPR and attack detection latency, and provides further insights into their practical effectiveness in real-time scenarios. Finally, Section 5.3 summarizes the key findings from previous sections.

Table 3 The metrics we use for performance evaluation in our approach

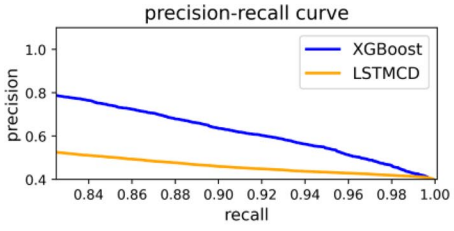
Metrics	Formula
accuracy A	A=(TP+TN)/(TP+TN+FP+FN)
false positive rate FPR	FPR=FP/(FP+TN)
f1-score F1	F1=2×((P×R)/(P+R))
recall R	R=TP/(TP+FN)
precision P	P=TP/(TP+FP)
latency Δl	$\Delta l = t_d - t_i$
average latency ΔL	$\Delta L = \sum_{i=0}^{N} \Delta l_i / N$
sequences detection rate SDR	S=detected/total

5.1 Results with state-of-the-art metrics

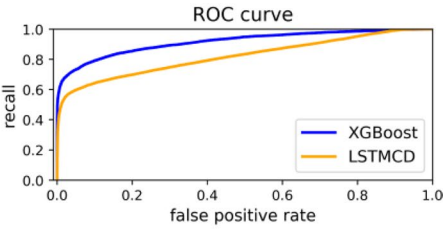
First, we compute the ROC and precision-recall curves to have a preliminary understanding of the classification capabilities of the detector in different settings. In Fig. 6, we show results for LSTMCD and XGBoost trained on both ROSPaCe and Arancino, and the XGBoost and ExtraTreeClassifier for the Dos MQTT dataset. This shows that there is some detection capability, especially for XGBoost, that has good results on all three datasets. The other algorithms are not reported for brevity, but they have similar results.

RQ1: Are attacks easily detectable? How difficult is it to train an accurate detector? Next, we study the amount of misclassification and whether it can be considered acceptable for deployment. We rely on Table 4 and Table 5.

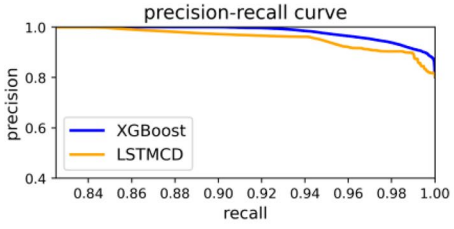
We set an upper limit of 0.01 to the FPR threshold for all the detectors. This is 1 false positive out of 100 data points. With such a maximum FPR, the value of precision easily gets close to 1, except for cases labeled with *NaN* in the table, where no correct detections occurred. Given this consideration, precision is not shown in Table 5. We adopt this specific



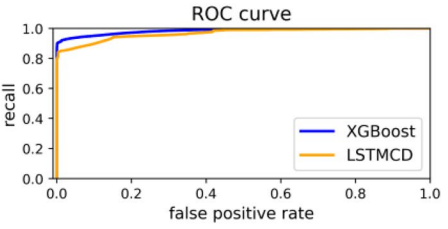
A) Precision-recall curves on Arancino.



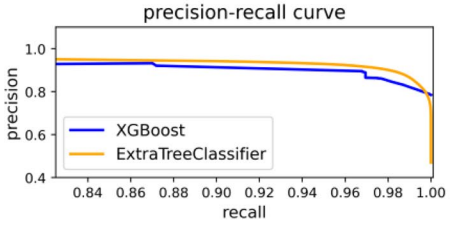
B) ROC curves on Arancino.



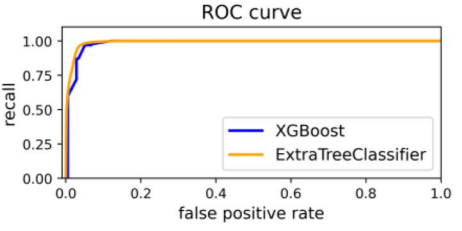
A) Precision-recall curves on ROSPaCe.



B) ROC curves on ROSPaCe.



A) Precision-recall curves on DoS MQTT.



B) ROC curves on DoS MQTT.

Fig. 6 Precision-recall (left) and ROC Curves (right) for XGBoost and LSTMCD on Arancino (top) and ROSPaCe (middle), and for XGBoost and ExtraTreeClassifier on DoS MQTT (bottom)

Table 4 Summary of results across ROSPaCe, Arancino, and DoS MQTT, including normal data points. NaN means no correct detections at all, while the dash means the algorithm was random guessing. FPR threshold is 0.01

Dataset	Detectors														
	XGBoost			Extr.Random.Trees			Isolation Forest			LSTMCD			LSTM		
	A	R	F1	A	R	F1	A	R	F1	A	R	F1	A	R	F1
ROSPaCe	0.927	0.991	0.952	0.61	0.52	0.683	0.57	0.477	0.644	0.879	0.911	0.953	0.20	0.001	0.003
Arancino	0.851	0.634	0.769	0.809	0.524	0.681	—	—	—	0.794	0.487	0.648	—	—	—
DoS MQTT	0.875	0.597	0.748	0.889	0.66	0.787	—	—	—	—	—	—	—	—	—

Table 5 Results obtained on ROSPaCe, Arancino and DoS MQTT. NaN means no correct detections at all, while the dash means the algorithm was random guessing. FPR threshold is set to 0.01

Attack/error		XGBoost		Extr.Random. Trees		Isolation Forest		LSTMCD		LSTM	
		R	F1	R	F1	R	F1	R	F1	R	F1
ROSPaCe	mflood	0.89	0.94	0.865	0.927	0.86	0.924	0.872	0.932	0.001	0.003
	ndisco	0.356	0.53	0.72	0.84	NaN	NaN	0.75	0.86	NaN	NaN
	nflood	0.999	0.999	0.09	0.15	0.018	0.037	0.856	0.922	0.03	0.07
	rosc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	ros_rec	0.007	0.014	0.002	0.004	NaN	NaN	NaN	NaN	NaN	NaN
	ros_ref	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Arancino	disk	0.986	0.993	0.956	0.977	—	—	0.979	0.989	—	—
	mem	0.773	0.872	0.781	0.877	—	—	0.862	0.926	—	—
	redr	0.982	0.991	0.962	0.98	—	—	0.98	0.99	—	—
	deadl	0.313	0.477	0.241	0.388	—	—	0.2	0.34	—	—
	cpus	0.861	0.925	0.578	0.733	—	—	0.54	0.70	—	—
	httpr	0.507	0.673	0.256	0.407	—	—	0.031	0.061	—	—
	proc1	0.129	0.229	0.15	0.26	—	—	0.000	0.001	—	—
	redw	0.959	0.979	0.573	0.729	—	—	0.662	0.796	—	—
	proc2	0.132	0.233	0.133	0.235	—	—	0.002	0.004	—	—
DoS MQTT	bf1	0.244	0.401	0.661	0.796	—	—	—	—	—	—
	delay	0.644	0.784	0.676	0.806	—	—	—	—	—	—
	syn	0.175	0.298	0.557	0.716	—	—	—	—	—	—
	sub	0.852	0.929	0.556	0.715	—	—	—	—	—	—
	will	0.779	0.875	0.729	0.843	—	—	—	—	—	—

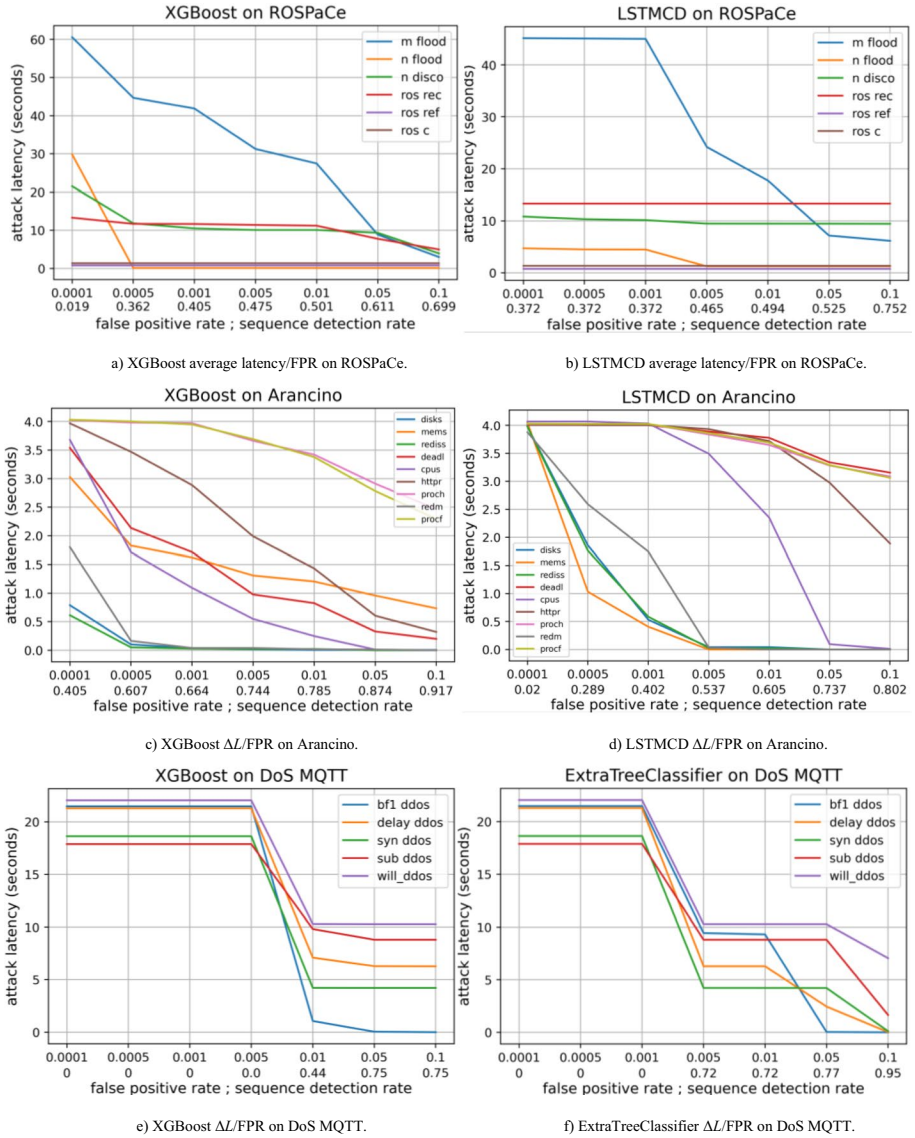
false positive rate (FPR) as it is used in prior works (Zoppi et al. 2023b; Puccetti et al. 2024; Puccetti and Ceccarelli 2024a) on the considered datasets, and it provides a meaningful basis for interpreting detection latency. Nevertheless, results across a range of FPR values are reported in Fig. 7 and Fig. 9, and in our repository for completeness. There are no established standards or industry guidelines for selecting an ideal FPR. In this paper, we bring attention to this gap and encourage further discussion on the topic.

The results for Isolation Forest and LSTM on the Arancino and DoS MQTT datasets were close to random guessing, with the same outcome observed for LSTMCD on the latter. For this reason, the results are not in Table 5.

As reported in Table 4, XGBoost shows the best performance for the Arancino and the ROSPaCe datasets, while ExtraTreeClassifier shows the best performance on the DoS MQTT dataset. In general, the algorithms for unordered data points surpass algorithms for ordered data points.

The ROSPaCe attacks are best detected by XGBoost and LSTMCD. XGBoost achieves an accuracy of 0.927, a recall of 0.991, and an f1-score of 0.952. LSTMCD shows a slightly better performance in the f1-score with 0.953 while losing the comparison in terms of accuracy and recall, respectively 0.879 and 0.911. The LSTM model is the worst in terms of performance: recall and f1-score are near 0.

For the Arancino dataset, again, the detectability is overall low. The best is XGBoost with an accuracy of 0.851, a recall of 0.634, and f1-score of 0.769. Extremely Randomized Trees show similar results with an accuracy of 0.809, a recall of 0.524, and f1-score of 0.681, while LSTMCD is much behind.



a) XGBoost average latency/FPR on ROSPaCe.

b) LSTMCD average latency/FPR on ROSPaCe.

c) XGBoost ΔL/FPR on Arancino.

d) LSTMCD ΔL/FPR on Arancino.

e) XGBoost ΔL/FPR on DoS MQTT.

f) ExtraTreeClassifier ΔL/FPR on DoS MQTT.

Fig. 7 Trade-off between average latency and FPR for XGBoost and LSTMCD on ROSPaCe (top) and Arancino (middle), and for XGBoost and ExtraTreeClassifier (bottom) on DoS MQTT. FPR varies from 0.0001 to 0.1. SDR is also shown on the x-axis for completeness. Better viewed in color

For the DoS MQTT dataset, the ExtraTreeClassifier is the best model with an overall accuracy of 0.889, a recall of 0.66, and an f1-score of 0.787.

In Table 5, we report results for each attack. We omit accuracy since the evaluation is restricted to attack instances; under this condition, accuracy is equivalent to recall, and therefore, only recall is reported.

All three datasets are difficult, meaning that the anomalous behavior of the system is not easy to observe with the available features. This is particularly evident in the case of the ROSPaCe dataset, where the extracted features fail to effectively capture the anomalies induced by the rosc and ros_ref attacks, thereby preventing successful detection. Also, the better performances of XGBoost and ExtraTreeClassifier suggest that the observable anomalies in all datasets are punctual rather than contextual or collective, or that the short interval of 10 data points set for LSTM and LSTMCD is not enough to capture efficiently the evolution of the system to detect anomalous patterns.

5.2 Latency-related metrics

RQ2: Which algorithm is more effective at detecting attack sequences? In Table 6, we report the results obtained with the proposed evaluation approach. We measure the sequence detection rate and the average latency.

XGBoost shows the best performance in terms of SDR for all three datasets considered. In the case of ROSPaCe, XGBoost, and LSTMCD show similar performances with an SDR of 0.50 and 0.494, which means there is at least one correct detection in half of the sequences.

ExtraTreeClassifier and LSTM perform worse than XGBoost, with an SDR of 0.388 and 0.34, respectively: there is at least a correct detection in a third of the sequences. As

Table 6 Sequence detection rate (SDR) and average latency (ΔL) measured in seconds on ROSPaCe, Arancino, and DoS MQTT. The FPR threshold is set to 0.01

Attack/error		XGBoost		Extr. Ran. Trees		Isol. Forest		LSTMCD		LSTM	
		SDR	Δ*L*	SDR	Δ*L*	SDR	Δ*L*	SDR	Δ*L*	SDR	Δ*L*
ROSPaCe	mflood	0.69	12.87	0.357	8.71	0.738	2.596	0.783	6.03	0.277	39.39
	ndisco	1.0	10.07	1.0	10	0.0	NaN	1.0	1.24	NaN	NaN
	nflood	1.0	0.094	0.666	7.19	0.111	1.80	1.0	9.46	0.97	11.63
	rosc	0.0	NaN	0.0	NaN	0.0	NaN	0.0	NaN	NaN	NaN
	ros_rec	0.13	4.311	0.013	7.17	0.0	NaN	0.0	NaN	NaN	NaN
	ros_ref	0.0	NaN	0.0	NaN	0.0	NaN	0.0	NaN	NaN	NaN
	all	**0.50**	**6.8**	**0.388**	**8.26**	**0.178**	**2.198**	**0.494**	**5.57**	**0.34**	**25.51**
Arancino	disk	1.0	0.005	0.978	0.01	—	—	0.989	0.0	—	—
	mem	0.988	1.16	0.988	1.10	—	—	1.0	0.0	—	—
	redr	0.994	0.0	0.977	0.017	—	—	0.994	0.0	—	—
	deadl	0.844	0.236	0.543	0.0	—	—	0.20	2.799	—	—
	cpus	0.97	0.127	0.623	0.198	—	—	0.50	0.661	—	—
	httpr	0.769	0.664	0.295	0.40	—	—	0.18	2.479	—	—
	proc1	0.21	1.254	0.15	0.0	—	—	0.27	2.652	—	—
	redw	1.0	0.017	0.83	1.03	—	—	1.0	0.0	—	—
	proc2	0.22	1.183	0.139	0.292	—	—	0.238	2.594	—	—
	all	**0.785**	**0.51**	**0.62**	**0.33**	—	—	**0.605**	**1.24**	—	—
DoS MQTT	bf1	1.0	1.066	0.812	9.308	—	—	—	—	—	—
	delay	0.833	7.098	0.833	6.287	—	—	—	—	—	—
	syn	0.8	4.221	0.8	4.223	—	—	—	—	—	—
	sub	0.5	9.819	0.545	8.804	—	—	—	—	—	—
	will	0.666	10.295	0.666	10.281	—	—	—	—	—	—
	all	**0.74**	**6.49**	**0.72**	**7.78**	—	—	—	—	—	—

expected, Isolation Forest yields the worst results since it is an unsupervised algorithm, which generally performs worse than supervised methods in the absence of unknowns (Zoppi et al. 2023a). A similar trend is measured on Arancino, but the gap between XGBoost and LSTMCD is more evident.

In the case of DoS MQTT, the best performance is from XGBoost with an SDR of 0.74. ExtraTreeClassifier shows a similar performance with an SDR of 0.72. The results for each attack are very similar, with the exception that XGBoost can perfectly detect the bf1 attack, while ExtraTreeClassifier detects only near 80% of the sequences.

Again, the success of algorithms for unordered data points suggests a prevalence of punctual anomalies or the insufficiency of the time interval set for LSTMCD and LSTM.

RQ3: How does the detectability of attacks change when evaluated using standard classification metrics versus sequence detection rate? We now compare the results of Table 4 and Table 5, ranked according to f1-score, and those of Table 6, ranked according to SDR.

In the case of Arancino, the ranking is the same: XGBoost shows the best overall performance, followed by ExtraTreeClassifier and LSTMCD. However, the gap between XGBoost and the other models in Table 4 and Table 5 is much more evident than in Table 6. In fact, in the case of Table 5, XGBoost detects many anomalous data points in the same sequence. If we are interested in the first detection of an anomaly, clearly Table 5 presents misleading outcomes. For example, ROSPaCe includes some sequences where the anomalous operations are composed of many data points, especially in the case of DoS attacks such as *nflood* (398,655 data points for each sequence of anomalous operations, see Table 2 and (Puccetti et al. 2024). XGBoost correctly detects many of these data points, so its overall f1-score is very good. The most evident discrepancy between the two evaluation methods can be observed in the *bf1* attack of the DoS MQTT dataset. XGBoost obtains a recall of 0.244 and an f1-score of 0.40, which is disappointingly low. The perspective about the detectability of such attacks is drastically changed if considering the SDR: XGBoost detects all the *bf1* sequences. Similar considerations can be applied to the *deadl* error of the Arancino dataset. XGBoost obtains a recall of 0.313 and an F1-score of 0.477, but the detectability of such error is more optimistic with an SDR of 84.4%. Another interesting observation for Arancino is considering LSTM, which generally shows poor classification performance in Table 5. The *nflood* attack has a very low recall, but SDR = 1.0. In this case, the model can recognize very few attack data points, but in all the sequences. This may suggest that there is a specific operation or packet sent by the attacker that always reveals the execution of an *ndisco* attack sequence, and the anomalies are detectable in only a part of the sequence.

RQ4: How long is the attack being executed before detection? We comment on the attack latency with the support of Table 6.

For the ROSPaCe dataset, we mainly consider the XGBoost and LSTMCD models as they show the best SDR. Further, they are closely aligned, giving us the possibility to comment fairly on the differences in latency. LSTMCD, on average, detects attacks earlier than XGBoost, with a ΔL on all the attacks of 5.57 s against 6.8 s. XGBoost performs especially well on the nflood attack with $\Delta L = 0.094$, given an average duration of the attack of 29.658 s, which means it is detected almost instantly. With the same SDR, LSTMCD takes

by far more time with $\Delta L = 9.46$ s. However, LSTMCD performs better on the ndisco attack with $\Delta L = 1.24$ s against the 10.07 s of XGBoost. LSTMCD is also better at detecting the mflood attack, taking half of the time ($\Delta L = 6.03$ against 12.87 of XGBoost).

For the Arancino dataset, we focus again on XGBoost and LSTMCD, as both show competitive SDRs. On average, XGBoost detects attacks slightly faster, with an average latency of 0.51 s, compared to 1.24 s for LSTMCD. For high-impact anomalies like disk, redr, and redw, both models detect almost instantly, with LSTMCD achieving zero latency, and XGBoost only slightly behind. LSTMCD also detects the mem anomaly instantly ($\Delta L = 0.0$), outperforming XGBoost ($\Delta L = 1.16$), despite both having near-perfect SDRs. However, XGBoost consistently detects more complex anomalies faster. For instance, on deadl, it triggers detection at 0.236 s, while LSTMCD takes 2.799 s. The same pattern holds for cpus (0.127 vs. 0.661), httpr (0.664 vs. 2.479), and both proc1 and proc2, where LSTMCD shows more than double the latency of XGBoost. Although LSTMCD is slightly faster on a few straightforward anomalies, XGBoost offers lower latency overall, particularly for subtler, system-level issues, while maintaining higher or comparable SDRs.

For the DoS MQTT dataset, we compare XGBoost and ExtraTreeClassifier, the only two models with complete and comparable results. In terms of detection latency, XGBoost performs slightly better overall, with an average latency of 6.49 s, compared to 7.78 s for ExtraTreeClassifier, while also maintaining a marginally higher SDR (0.74 vs. 0.72). XGBoost is particularly effective on the bf1 attack, detecting it with a latency of 1.066 s and achieving perfect accuracy (SDR = 1.0), while ExtraTreeClassifier takes much longer (9.308 s) and detects fewer sequences (SDR = 0.812). On delay and syn, both models achieve identical SDRs (0.833 and 0.8, respectively), but XGBoost detects delay slightly later (7.098 vs. 6.287) and performs equally on syn with almost identical latencies. For more subtle application-level anomalies like sub and will, both models show higher latencies (close to or over 10 s), but XGBoost still retains a small edge on detection timing. These results suggest that XGBoost is more responsive for high-impact, easily identifiable attacks, particularly bf1, while ExtraTreeClassifier performs comparably but with slightly higher latency across most attacks.

In conclusion, average attack latency varies significantly depending on the attack and error considered. This can be attributed to the variable effect of attacks and errors on the target system: feature values may be altered differently, and their visible effects manifest at different time intervals since injection. No single model consistently outperforms the other in all scenarios. This variability underscores the importance of evaluating detection systems not only by accuracy but also by considering timely detection capability. Our latency-based evaluation methodology, which relies on the packet-structured organization of the datasets, enables fine-grained insights into model performance. This perspective reveals nuances in model behavior that traditional classification metrics alone cannot capture.

RQ5: What is the trade-off between FPR and average latency? We report in Fig. 7 how the average latency ΔL varies for different FPRs. For brevity, we consider the best models for each dataset. For the ROSPaCe and Arancino datasets, we consider only LSTMCD and XGBoost, whereas for the DoS MQTT dataset, we consider XGBoost and ExtraTreeClassifier.

As expected, increasing the FPR reduces the average latency because the detector is more suspicious. When a sequence is not detected, we use its duration as an attack or error latency. This implies it is too late to perform the detection, with the attack latency coinciding with its peak value. This methodology avoids spikes in the graph. When the FPR rises, more sequences may have at least one detection; such detections may be in a later part of the sequence, so the ΔL may increase. In other words, while the number of anomalous sequences detected does not increase significantly, they may consistently add to average latency if detected. We avoid these cases that make the FPR/Latency trade-off unclear and in contrast with the general concept of the ROC or precision-recall curves, which always show monotonic trends.

LSTMCD and XGBoost on ROSPaCe exhibit closely aligned ΔL across all attacks, with LSTMCD registering a ΔL generally shorter than XGBoost. This suggests that, on average, LSTMCD can detect attacks earlier than XGBoost for a given FPR.

In Arancino, LSTMCD and XGBoost have similar behavior, but the ΔL of XGBoost is shorter. This is probably because of two reasons: i) Arancino includes anomalies that can be classified as punctual, and they are placed in the initial part of the anomalous operations; ii) since LSTMCD uses a window of data points, it requires processing a larger piece of data than XGBoost before an anomaly is classified.

In DoS MQTT, XGBoost, and ExtraTreeClassifier have similar results for all the attacks except for the *bf1,* which shows a detection latency of 1.066 against the 9.308 of ExtraTreeClassifier (see Table 6). However, looking at Fig. 7, the ExtraTreeClassifier can detect attacks for a significantly lower FPR threshold. XGBoost cannot detect any sequence when the FPR is smaller than 0.01, while ExtraTreeClassifier detects 72% of the sequence with an FPR of 0.005. As a result, the two detectors show similar performance, but the ExtraTreeClassifier should be preferred if the system has a stricter FPR requirement.

To resume, Fig. 7 shows a relationship between FPR, ΔL, and SDR. The FPR can be tuned to set the desired average latency ΔL and SDR, following the system's operational requirements. For example, in ROSPaCe, LSTMCD should be elected if the main desideratum is to detect flooding attacks with low FPR, or when a short detection time is fundamental. XGBoost should be preferred if the objective is to have a high SDR, i.e., increase the chance of detecting an attack, even later. To validate this result, we statistically demonstrate that attack latency remains the same or decreases as the FPR rises.

For each dataset, we applied a Spearman rank correlation test to evaluate the relationship between FPR and latency. The null hypothesis (H_0) states that there is no monotonic association between the two variables, while the alternative hypothesis (H_1) states that such an association exists. We calculate the Spearman correlation coefficient by pairing FPR threshold values with their corresponding latency values in ordered vectors, where FPR acts as the reference label for latency. These vectors are then analyzed using the Spearman function to derive the correlation score, and the associated p-value.

To refine our analysis, we filter out cases that inherently confirm our hypothesis, specifically: *i*) when detection occurs at the first data point in the sequence for all FPR thresholds, *ii*) when no detection occurs at any FPR threshold,

iii) when detection happens at the same timestamp for all FPR thresholds.

These cases show non-negative correlation coefficients, introducing noise in our analysis. However, they already validate our hypothesis that latency may remain constant when FPR rises. We discard them and compute the Spearman correlation on the remaining sequences.

The distribution of correlation coefficients, shown in Fig. 8, is consistently negative, with p-values below 0.05 in 83% of cases, providing evidence to reject the null hypothesis H_0 and indicating a statistically significant monotonic association. In the 17% of cases where latency remains stable across most FPR thresholds, the p-value is around 0.13. The presence of these cases continues to support the overall trend, as they are similar to case iii), and where latency consistently remains the same or decreases as the false positive rate (FPR) increases. A p-value below 0.05 suggests we can reject the null hypothesis H_0, indicating a statistically significant monotonic relationship. We report results for each sequence and each dataset in our repository. For example, we show the distribution of correlation scores for ROSPaCe and XGBoost in Fig. 8.

RQ6: Do time features improve algorithms for unordered data points? We analyze the results presented in Table 7 and Table 8. We highlight in red the metric values that perform worse compared to those in Table 6 (no temporal features), and in green those that show improvements.

In general, the SDR achieved by XGBoost on ROSPaCe is higher when using shorter intervals. The best results are obtained with a 10-s interval, where the SDR reaches 0.541, and a 5-s interval, where it reaches 0.524. In contrast, longer intervals show lower SDR values, with the best obtained within 30 s, with an SDR of 0.485. Notably, no clear downward trend is observed as the time interval increases. For instance, the SDR for 5 s is lower than for 10 s, and the FPR for a 30-min window is higher than for both 30 s and 10 min. This variability may be due to the structure of ROSPaCe, which integrates different sessions that may contain varying time gaps in the data.

Compared to Table 6, using time features with a 10-s interval improves the performance of XGBoost and LSTMCD, achieving an SDR of 0.541, compared to 0.50 and 0.494, respectively. This improvement is primarily due to better detection of *m flood* and *nflood* attacks, which involve sending a huge volume of packets over random time durations. The variations in the mean, variance, and linear regression slope of ROSPaCe features make these attacks more easily detectable.

The best performance of a short time window is confirmed in the case of Arancino, with the best results obtained by using 5 s as duration. The SDR in this case is 0.812, while the

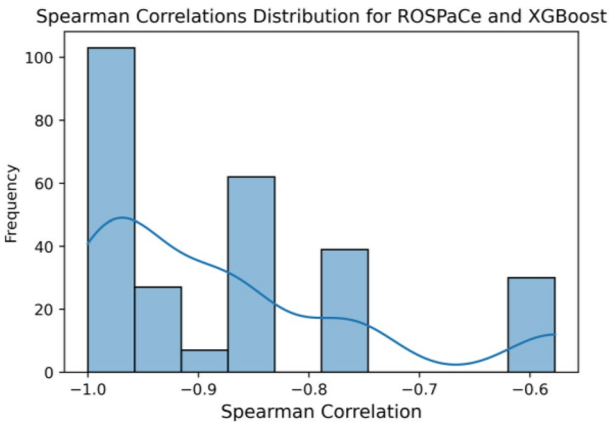


Fig. 8 Spearman correlation score distribution for dataset ROSPaCe and model XGBoost

Table 7 Sequence detection rate (SDR) and average latency (ΔL) measured in seconds on ROSPaCe, Arancino, and DoS MQTT with time features. Each XGBoost model is trained on a dataset that includes statistical features such as the mean, variance, and linear regression slope of data points calculated over different time intervals: 5 s, 10 s, 30 s, 10 min, and 30 min. FPR threshold set to 0.01

Attack/error		XGBoost 5 s		XGBoost 10 s		XGBoost 30 s		XGBoost 10 min		XGBoost 30 min	
		SDR	ΔL	SDR	ΔL	SDR	ΔL	SDR	ΔL	SDR	ΔL
ROSPaCe	mflood	0.857	19.00	0.904	15.704	0.738	24.935	0.630	26.842	0.678	25.836
	ndisco	1.0	7.2	1.0	9.731	1.0	10.085	1.0	10.076	1.0	10.076
	nflood	1.0	2.845	1.0	5.52	0.777	15.682	0.333	19.952	0.888	5.741
	rosc	0.0	NaN	0.0	NaN	0.0	NaN	0.0	NaN	0.0	NaN
	ros_rec	0.054	12.36	0.081	11.875	0.013	13.187	0.027	13.319	0.027	13.14
	ros_ref	0.0	NaN	0.0	NaN	0.0	NaN	0.0	NaN	0.0	NaN
	all	**0.524**	**10.35**	**0.541**	**10.50**	**0.485**	**15.97**	**0.447**	**17.54**	**0.473**	**13.69**
Arancino	disk	1.0	0.31	1.0	0.090	0.989	0.294	0.989	0.197	0.989	0.213
	mem	1.0	2.92	1.0	3.88	0.982	4.845	0.994	4.725	0.994	4.485
	redr	1.0	0.188	0.994	0.127	0.994	0.155	0.994	0.194	0.988	0.172
	deadl	0.833	3.263	0.698	4.586	0.489	7.15	0.596	5.645	0.639	5.107
	cpus	0.988	1.847	0.882	3.435	0.511	7.988	0.6	6.247	0.647	5.717
	httpr	0.923	3.852	0.686	6.881	0.242	11.414	0.39	9.65	0.485	8.254
	proc1	0.265	12.246	0.024	13.849	0.030	13.656	0.012	13.885	0.024	13.771
	redw	1.0	1.022	1.0	1.056	0.994	1.152	1.0	0.813	1.0	0.762
	proc2	0.25	12.302	0.023	13.86	0.011	13.889	0.005	13.988	0.023	13.848
	all	**0.812**	**4.21**	**0.709**	**5.30**	**0.592**	**6.727**	**0.63**	**6.14**	**0.652**	**5.81**
DoS MQTT	bfl	0.562	14.11	0.75	9.434	0.75	9.980	0.0	NaN	0.0	NaN
	delay	0.444	14.994	0.833	6.276	0.833	6.276	0.0	NaN	0.0	NaN
	syn	0.4	11.985	0.8	4.216	0.8	4.216	0.0	NaN	0.0	NaN
	sub	0.362	12.98	0.545	8.804	0.545	8.804	0.0	NaN	0.0	NaN
	will	0.666	11.032	0.666	10.28	0.666	10.28	0.0	NaN	0.0	NaN
	all	**0.49**	**13.02**	**0.71**	**7.8**	**0.71**	**7.8**	**0.0**	**NaN**	**NaN**	**NaN**

second-best obtained with 10 s is consistently worse with an SDR of 0.709. The performance of the original model with SDR 0.785 is improved. This may be because the average duration of the sequences is 14 s.

In the case of DoS MQTT, the best result is again with a 5-s interval with an SDR of 0.71. The SDR is the same for 10 s and 30 s. There is no observable improvement from the original model that reports an SDR equal to 0.74. Altogether, there is no improvement from aggregating features, while reducing the number of original features reduces detectability.

We comment on the results for the ExtraTreeClassifier trained with time features in Table 8. As for XGBoost, the best results are obtained with a short interval. The best results for ROSPaCe come from using the 5-s time interval with an SDR of 0.529. The 10-s time interval is closely aligned with an SDR of 0.526. Performance drops drastically when augmenting the duration, with no detectability when using 30 s. There is an improvement compared to the 0.50 SDR of the original model.

In the case of Arancino, the best result is with a time interval of 5 s with an SDR of 0.619. However, in this case, no improvement is observable from the original model, with the other time interval showing very poor detectability. This can be explained by the short duration of the Arancino sequences.

Table 8 Sequence detection rate (SDR) and average latency (ΔL) measured in seconds on ROSPaCe, Arancino, and DoS MQTT with time features. Each ExtraTreeClassifier model (ExtraTree) is trained on a dataset that includes statistical features such as the mean, variance, and linear regression slope of data points calculated over different time intervals: 5 s, 10 s, 30 s, 10 min, and 30 min. The FPR threshold is set to 0.01

Attack/error		ExtraTree 5 s		ExtraTree 10 s		ExtraTree 30 s		ExtraTree 10 min		ExtraTree 30 min	
		SDR	ΔL	SDR	ΔL	SDR	ΔL	SDR	ΔL	SDR	ΔL
ROSPaCe	mflood	0.940	11.703	0.928	11.73	0.0	NaN	0.488	33.54	0.5	35.100
	ndisco	1.00	8.943	1.0	10.016	0.0	NaN	0.97	10.708	1.0	10.006
	nflood	0.888	6.244	0.888	10.354	0.0	NaN	0.111	26.553	0.555	14.463
	rosc	0.0	NaN	0.0	NaN	0.0	NaN	0.0	NaN	0.0	NaN
	ros_rec	0.0	NaN	0.0	NaN	0.0	NaN	0.0	NaN	0.0	NaN
	ros_ref	0.0	NaN	0.0	NaN	0.0	NaN	0.0	NaN	0.0	NaN
	all	**0.529**	**8.96**	**0.526**	**10.7**	**0.0**	**NaN**	**0.399**	**23.6**	**0.385**	**19.85**
Arancino	disk	0.978	1.053	0.673	6.128	0.288	10.62	0.844	2.545	0.866	2.192
	mem	0.982	4.72	0.565	11.554	0.28	13.834	0.868	8.251	0.96	6.662
	redr	0.994	0.616	0.727	5.288	0.333	10.327	0.872	2.088	0.9	1.7
	deadl	0.456	8.537	0.053	13.327	0.021	13.811	0.129	12.225	0.22	10.9
	cpus	0.752	5.511	0.052	13.652	0.023	13.9	0.147	12.435	0.235	11.46
	httpr	0.272	11.461	0.0	NaN	0.0	13.96	0.017	13.769	0.047	13.49
	proc1	0.06	13.68	0.006	13.951	0.012	13.867	0.0	NaN	0.006	NaN
	redw	0.96	5.028	0.19	12.451	0.016	13.830	0.31	11.48	0.367	10.892
	proc2	0.04	13.819	0.0	NaN	0.023	13.813	0.005	13.97	0.0	14.052
	all	**0.619**	**7.15**	**0.259**	**10.90**	**0.113**	**13.10**	**0.363**	**9.58**	**0.408**	**8.91**
DoS MQTT	bf1	1.0	0.669	1.0	0.981	1.0	0.981	0.312	17.656	0.312	17.656
	delay	0.944	4.707	0.833	6.276	0.833	6.276	0.666	11.499	0.388	15.15
	syn	0.8	4.216	0.8	4.216	0.8	4.216	0.7	8.547	0.7	8.387
	sub	0.545	8.804	0.545	8.804	0.545	8.804	0.545	9.377	0.545	9.954
	will	0.666	10.280	0.666	10.28	0.666	10.28	0.666	10.359	0.666	10.286
	all	**0.77**	**5.73**	**0.75**	**6.11**	**0.75**	**6.11**	**0.59**	**11.48**	**0.49**	**12.41**

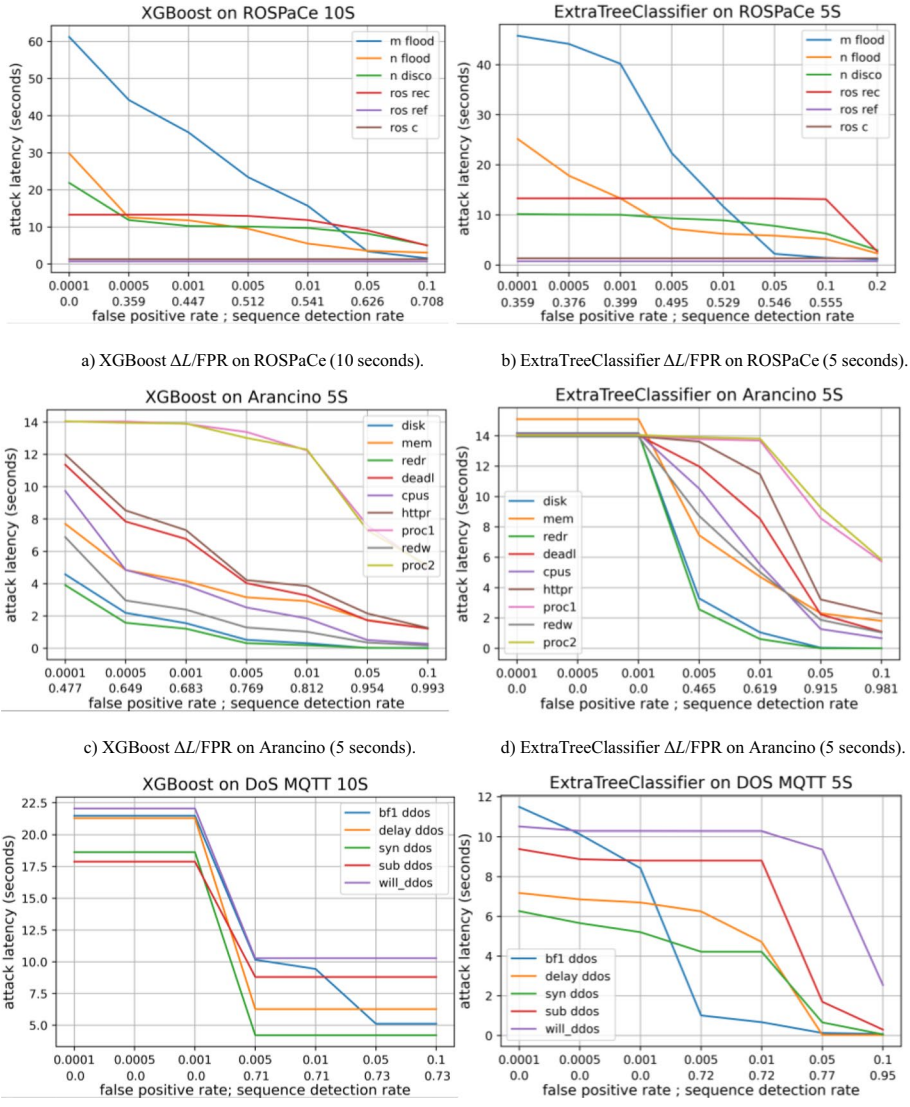
In the case of DoS MQTT, the best results are with 5 s, 10 s, and 30 s, with, respectively, SDR equal to 0.77, 0.75, and 0.75. Again, a longer interval causes a drastic reduction in SDR. In this case, there is an improvement in the SDR compared to the original model, which has an SDR of 0.72.

We comment on the plot in Fig. 9 to compare XGBoost and ExtraTreeClassifier at different FPR thresholds.

On the ROSPaCe dataset, when the FPR requirement is below 0.001, the ExtraTreeClassifier should be preferred to XGBoost. As reported in Fig. 9, the latter shows no sequence detectability for FPR of 0.0001 and lower detectability for FPR equal to 0.0005 with SDR of 0.359 against 0.376 of the ExtraTreeClassifier. When FPR arises, XGBoost is consistently better than ExtraTreeClassifier with comparable latencies.

On the Arancino datasets, XGBoost is always the preferred solution as the SDR is generally consistently better.

On the DoS MQTT dataset, the SDR is 0 for both models when the FPR is below 0.005. From SDR equal to 0.005, ExtraTreeClassifier is always preferred: it slightly beats XGBoost with an SDR of 0.72 for FPR of 0.005 and 0.001 against the 0.71 of XGBoost, where the



a) XGBoost ΔL/FPR on ROSPaCe (10 seconds).

b) ExtraTreeClassifier ΔL/FPR on ROSPaCe (5 seconds).

c) XGBoost ΔL/FPR on Arancino (5 seconds).

d) ExtraTreeClassifier ΔL/FPR on Arancino (5 seconds).

e) XGBoost ΔL/FPR on DoS MQTT (10 seconds).

f) ExtraTreeClassifier ΔL/FPR on DoS MQTT (5 seconds).

Fig. 9 Trade-off between average latency and FPR for XGBoost and ExtraTreeClassifier on ROSPaCe (top), Arancino (middle), and DoS MQTT (bottom). Each figure specifies the time interval in seconds applied to compute time features. FPR varies from 0.0001 to 0.1. SDR is also shown on the x-axis for completeness. Better viewed in color

gap between the two is larger when augmenting FPR tolerance. The Better performance of the ExtraTreeClassifier is also reflected in latency, with average latency always comparable or lower when FPR arises.

Lastly, we compare the models with time features in Fig. 9 with their original counterparts in Fig. 7.

On the ROSPaCe dataset, XGBoost with time features generally achieves a higher SDR compared to the original model, while latency remains comparable. However, in the case of certain flood attacks, the original model detects them with zero latency, whereas the time-feature model does not.

For the Arancino dataset, XGBoost with time features also outperforms the original model in terms of SDR. However, the original model has significantly lower latency. Given that the difference in SDR is not substantial, the original model should be preferred if low latency is a strict requirement.

On the DoS MQTT dataset, the time-feature model can detect attacks at a lower FPR threshold. When FPR is set to 0.005, it achieves an SDR of 0.71, whereas the original model shows no detectability. Additionally, the time-feature model generally provides better latency.

RQ7: How does the impact of attacks on the system vary depending on the detection model used for defense? Using the ROSPaCe dataset and results in Table 6, we report average detection latencies for XGBoost and LSTMCD across three attacks: mflood, ndisco, and nflood, estimating their effectiveness based on the attack's intended goal.

In ndisco, which scans the top 1000 ports to infer the OS (at 30 ports/sec), XGBoost detects the attack in 10 s, allowing 300 ports to be scanned before detection. In contrast, LSTMCD detects it in 1.24 s, limiting the scan to just 37 ports, and it is significantly more effective in protecting the system from discovery.

In case of the mflood attack, which in our setting sends approximately 3000 packets per second, XGBoost takes 12.87 s, allowing 38,610 packets to be sent before detection.

LSTMCD improves this with a detection time of 6.07 s (18,210 packets). While this can be considered a reasonably long time, the ROSPaCe system did not collapse in either case, but only showed a slowdown, as it is able to sustain the proposed rate of arrival packets for a long time.

In nflood, where 4,500 packets per second are sent, XGBoost detects the attack in just 0.094 s (423 packets), outperforming LSTMCD, which takes 9.46 s (42,570 packets).

Notably, the results in flow-based datasets and the consequences for the system are different. Detection is delayed until flows are closed, making early detection highly dependent on flow duration.

We use flows from the CSE-CIC-IDS2018 (Sharafaldin et al. 2018) dataset as a reference example, as it is a notorious flow-based dataset often used to evaluate intrusion detectors. We consider the DDoS attack (LOIC-UDP), which sends UDP packets to random destination ports, the DoS Hulk, which sends different HTTP requests at random times, and SSH-Bruteforce attacks, which repeat ssh login attempts using various credential pairs (Table 9).

Table 9 Example flows of the CSE-CIC-IDS2018 dataset. The Protocol column indicates the network protocol used for the communication, Flow_Duration is the duration of the flow in microseconds, Tot_Fwd_Pkts is the number of packets sent by the source IP address, and Label describes the attack originating the flow

CSE-CIC-IDS2018				
Protocol	Flow_duration	Tot_Fwd_Pkts	…	Label
17	119999987	116161	…	DDOS attack-LOIC-UDP
6	66019148	66019148	…	DoS attacks-Hulk
6	1394820	8	…	Infiltration
6	401257	22	…	SSH-Bruteforce

The DDoS attack (LOIC-UDP) lasts approximately 120 s, meaning the attack can only be detected after this duration when using flow-based analysis. Similarly, the DoS Hulk attack is detectable only after around 66 s. In these cases, the flow-based approach delays detection until the flow concludes, whereas a packet-based method allows for earlier identification of the attack. For attacks like Infiltration and SSH-Bruteforce, this limitation is less pronounced since their durations are much shorter, approximately 1.3 s and 0.4 s, respectively. These short durations result in a temporal granularity closer to that of packet data.

A more comprehensive assessment that quantifies the actual damage to the system can be conducted, but it would require real-time monitoring during attacks and the ability to evaluate the system's dependability under stress. Unfortunately, this is not feasible in our case, as the IDS dataset contains only network traffic data and lacks information about the system's functional state or the impact of attacks beyond what can be inferred from network behavior alone (e.g., absence of response from the SPACE system in case of the mflood attack).

Although attack latency is not a direct substitute for traditional evaluation metrics such as accuracy, precision, or recall, it provides complementary insights into the performance of an Intrusion Detection System (IDS) when facing specific attacks. As illustrated by various examples, attack latency reflects how quickly the system can detect and respond to threats targeting a particular environment. Timely detection is critical, as it can significantly reduce the potential damage caused by an attack.

6 Summary of key findings

RQ1: Are attacks easily detectable? How difficult is it to train an accurate detector? The datasets are inherently challenging, as the available features often fail to reveal anomalous behavior, particularly in the ROSPaCe datasets, where rosc and ros_ref attacks remain undetected. The superior performance of models for unordered data suggests that anomalies are mostly point-based and that the short time window used for LSTM models may be insufficient to capture temporal patterns effectively.

RQ2: Which algorithm is more effective at detecting attack sequences? Based on SDR, XGBoost, and in general, algorithms for unordered data points are more robust across different types of attacks and systems. Models relying on sequence patterns, like LSTM and LSTMCD, are less effective, likely due to their sensitivity to time-window configuration and the nature of the anomalies, suggesting again that they are punctual rather than sequential.

RQ3: How does the detectability of attacks change when evaluated using standard classification metrics versus sequence detection rate? The comparison between f1-score, and SDR highlights that the latter offers a more realistic assessment of a model's ability to detect attacks in practical scenarios. While XGBoost consistently ranks highest across both metrics, f1-score can exaggerate performance by rewarding repeated detections within the same sequence, especially for long attacks like nflood in ROSPaCe. In cases like bf1 attack in DoS MQTT and the deadl error in Arancino, show that a low f1-score may mask high sequence-level effectiveness. For example, LSTM, despite low f1-score performance, achieves high SDR on some attacks by consistently identifying key points in each sequence. Broadly, SDR focuses on whether the model can detect

at least one anomaly per attack sequence, which aligns better with real-world detection goals and proves more suitable for evaluating early and actionable detection.

RQ4: How long is the attack being executed before detection? No single algorithm is optimal across all attack types, even when considering attack latency. While models designed for unordered data points generally show superior overall performance in terms of sequence detectability, models that incorporate temporal dependencies can offer lower detection latency for certain types of attacks. This highlights an important distinction: models that are less effective overall may still provide faster responses in specific scenarios. Our methodology enables us to evaluate these differences by focusing on latency at the sequence level, offering a complementary perspective to standard accuracy metrics, allowing us to select or tune models based not only on how well they detect, but also how timely, which is often critical in real-time applications.

RQ5: What is the tradeoff between FPR and average latency? As FPR increases, the attack latency remains stable or decreases: more relaxed thresholds allow earlier or more frequent detections. This trend holds consistently across datasets and models. Observing this tradeoff makes it possible to balance SDR, latency, and FPR according to operational constraints, something that traditional evaluation cannot achieve.

RQ6: Do time features improve algorithms for unordered data points? Time-based features in some cases improve the SDR of algorithms designed for unordered data points, particularly when short aggregation intervals (5–10 s) are used. For example, XGBoost and ExtraTreeClassifier achieve the highest SDRs with short time windows, confirming that capturing temporal dynamics through statistical summaries (e.g., mean, variance) can enhance detection. However, longer aggregation intervals consistently degrade performance, and in some cases, eliminate detectability.

The use of time features does not uniformly improve latency; in certain cases, especially in Arancino, the original model without time aggregation achieves faster detection. Overall, time-based feature aggregation is a valuable enhancement when applied with short windows, but must be tuned carefully to balance gains in detectability with acceptable latency and system-specific constraints.

RQ7: How does the impact of attacks on the system vary depending on the detection model used for defense? Detection latency directly impacts how much of an attack can execute before being detected, and this varies depending on the detection model used. In packet-based detection with the ROSPaCe dataset, models like LSTMCD and XGBoost show significant differences in how quickly they respond to specific attacks, affecting the system's exposure.

In contrast, flow-based systems inherently delay detection until the flow ends. This limitation can be critical for persistent attacks. Overall, latency and packet-based datasets provide a practical measure of real-world defensive effectiveness, offering insights beyond traditional metrics by reflecting how quickly a model can limit the impact of an ongoing attack.

7 Concluding remarks

We believe that a realistic assessment of IDs should include an investigation of the detection latency and the ability to detect a sequence of anomalous events. For example, in practice, we are usually interested in detecting one event to trigger response strategies and interrupt an attack path.

To measure this ability of our detector, we believe it is necessary to include the false positive rate, such that different detection latencies and the ability to timely detect anomalies can be related to false alarms. False alarms are usually a fundamental requirement for ID deployment, as an excessive number of false alarms will result in useless detectors.

We have experimentally and statistically demonstrated that a clear relation between false positive rate, detection latency, and the ability to detect a sequence of anomalous events exists, such that increasing the FPR would lead to an equal or shorter latency since the detector becomes more cautious and suspicious. We emphasize the need for the community to establish clearer guidelines for selecting FPR thresholds in intrusion detection, as the absence of standardized practices currently limits comparability and consistency across studies. We provided the means to compute the above metrics through the code in our repository.

However, we know that most public datasets available are unsuitable for computing these metrics (Puccetti et al. 2024). They are rarely structured in sequences of normal and anomalous behavior, and they label as anomalous a data point collected while an attack is injected, without evidence that the corresponding effect of the attack or error manifests in that specific data point and is thus observable by an algorithm.

Hypothetically, the visible effect could manifest later. This lack of representative datasets is particularly marked in ID for different reasons. The flow-based approach in constructing datasets prevents a granular analysis of the network traffic and an evaluation of the timely detection capability.

7.1 Conclusions

Our evaluation approach offers a proxy for assessing the performance of intrusion and error detection models. Considering latency, false positive rate, and sequence detection rate, we can make decisions tailored to specific system requirements and operational priorities. Our approach provides valuable information about the effectiveness of different models, allowing for the selection of the most suitable detection strategy to preserve system security and resilience. Additionally, our approach prevents biases introduced by dataset labeling methods, ensuring a realistic evaluation.

It is easy to apply our approach in other cases, provided that: *i*) at least the test set is labeled, and *ii*) it is in a time-series format, where the normal behavior (system is operational without attacks) alternates with the execution of attacks. We recommend that the evaluation of an anomaly detector before its deployment in a real operational environment follows the steps:

1. Organize the train and test sets in sequences, where each sequence is composed of normal operations followed by anomalous operations. Each sequence describes the system

dynamics when an attack is launched or an error is activated. This step may require building a dedicated dataset.
2. After training the detector, assess its performance using traditional state-of-the-art metrics.
3. Measure the average latency ΔL and SDR at different FPRs. This requires keeping track of i) the initial data point of each sequence, ii) the position of each data point labeled as anomalous, and iii) the index of the first correctly classified anomalous data point in each sequence. Given this information, computing ΔL and SDR is straightforward.
4. Choose the target FPR threshold that allows matching system operational requirements. It is necessary to recompute traditional state-of-the-art metrics once the target FPR is decided.

Last, we remark that the ID domain spans a wide variety of systems and monitoring targets. While our work specifically focuses on network-based ID, the core principles (organizing data into sequences of normal and anomalous operations and evaluating detection latency) are general and remain applicable to other types of monitoring targets. In particular, the concept of measuring attack latency has been largely overlooked in several subfields of ID beyond network traffic analysis. However, transferring the approach to other domains would require adaptation to comply with possible assumptions and requirements, for example, in case of log-based datasets that include natural language, it would be necessary to deploy different models to look back in time and correlate sentences with coarser temporal granularity, while resource-constrained systems may impose limitations on data collection and model complexity.

7.2 Threats to validity

In this section, we outline the factors that may affect the validity of our experiments, structured according to internal, external, and construct validity.

7.2.1 Internal validity

The hyperparameter search for the unordered data points algorithms was limited to a predefined range of values and parameters, which may not cover the entire space of potentially optimal settings. For the LSTM and LSTMCD models, we were unable to explore different batch sizes or sequence lengths due to computational resource constraints. Additionally, we used grid search with a fixed set of train-test splits, rather than more advanced tuning methods such as cross-validation. To maintain fairness across models, a uniform tuning procedure was applied, which may have limited model-specific optimization.

Our internal validity is also influenced by the choice of fixed window sizes for data processing and the lack of a sliding window approach for aggregating the temporal features.

Although sliding windows are more precise, they require significant computational resources. We tried preliminarily to subsample the datasets and apply a sliding windows on small data. Our initial results did not show notable improvement with sliding windows compared to fixed windows, possibly because the 5 and 10 s windows are already sufficiently short to capture the variations between normal and attacks. Most attacks are DoS-based, and the injected errors in Arancino show similar, easily distinguishable deviations from normal

behavior (e.g., higher packet rates or memory use). In contrast, subtler and shorter attacks would benefit from a sliding-window approach with overlap. A more comprehensive evaluation of sliding window configurations remains future work.

However, it is important to note that the rationale behind the model choice and configuration is to demonstrate the proposed approach on different ID models, either suited for unordered and ordered data points or time engineered features, rather than focusing on finding the optimal configuration and aiming for an absolute performance comparison between different ID solutions. Including latency in the evaluation of detectors enables a deeper understanding of the distinctions between these diverse families of detectors, insights that accuracy alone cannot reveal.

7.2.2 External validity

Attack manifestation speeds vary widely across different systems, resulting in latency values that are not directly comparable. Our approach aims to assist practitioners in tuning detectors to their specific requirements. Expanding evaluations to include more systems and scenarios would enhance the generalizability of our results and is a direction for future research.

7.2.3 Construct validity

It's important to note that our intention isn't to replace state-of-the-art metrics, which, although comparable, can sometimes be misleading. Rather, we aim to introduce a supplementary layer of evaluation. By computing the relationship between latency, SDR, and FPR alongside state-of-the-art metrics, we can pragmatically observe and calibrate them to achieve the desired detection latency for the specific system under observation.

7.3 Future works

The code to apply our dataset creation methodology, along with the code to compute latency at different FPR thresholds, can be better organized in a comprehensive framework. The framework would allow for rearranging the existing raw data of state-of-the-art datasets in a structure that allows packet-level analysis and alternates sequences of normal and attack operations. Finally, it would include functions to identify sequences in the datasets and compute our proposed latency-oriented metrics. Altogether, this would be a handy tool to facilitate the step from state-of-the-art ID to a framework that considers timely detection.

Future work will also investigate the use of sliding-window-based feature extraction to better capture temporal continuity and potentially improve anomaly detection performance.

Author contributions **Tommaso Puccetti**: Conceptualization, Implementation, Data Analysis, Writing. **Andrea Ceccarelli**: Conceptualization, Data Analysis, Writing.

Funding Open access funding provided by Università degli Studi di Firenze within the CRUI-CARE Agreement. This paper was partially supported by the MUR PRIN 2022 project FLEGREA—Federated Learning for Generative Emulation of Advanced Persistent Threat, by the project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union – NextGenerationEU, and by POR CREO Regione Toscana 2023 -D17H24004520009 WAU -Warehouse Automation.

Data availability **Tables**: Table 1, Table 2, Table 3 are reproduced from our previous conference paper (Puccetti and Ceccarelli 2024b). Table 4, Table 5 and Table 6 include partial data from (Puccetti and Ceccarelli 2024b) supplemented with additional results from our extended analysis.
Figures: Fig. 1 and Fig. 2 are reproduced from our previous conference paper (Puccetti and Ceccarelli 2024b). Figure 6 and Fig. 7 include partial data from (Puccetti and Ceccarelli 2024b) supplemented with additional results from our extended analysis.
Datasets: The ROSPaCe dataset is from our previous publication (Puccetti et al. 2024). Arancino dataset is from (Zoppi et al. 2023b). The DoS/DDoS-MQTT-IoT raw data files are from (Alatram et al. 2023). The CSE-CIC-IDS2018 is from (Sharafaldin et al. 2018). The data we produce in our experimental campaign are made public to facilitate the reproducibility of our experiments and are available at https://drive.google.com/drive/folders/1vrFOHsn88L1tSrHXysmZ8-js9yYLrNMw?usp=drive_link. (470 Gb).
Codes: Part of the code to run our experiments are reproduced from our previous conference paper (Puccetti and Ceccarelli 2024b) and publicly available at https://doi.org/10.6084/m9.figshare.26808259.v1. The code to run the experimental campaign of this paper is available at https://drive.google.com/drive/folders/1vrFOHsn88L1tSrHXysmZ8-js9yYLrNMw?usp=drive_link. (470 Gb).

Declarations

Ethical approval Not applicable.

Informed consent Not applicable.

Clinical trial number Not applicable.

Conflict of interest 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.

References

Al S, Dener M (2021) STL-HDL: a new hybrid network intrusion detection system for imbalanced dataset on big data environment. Comput Secur 110:102435. https://doi.org/10.1016/j.cose.2021.102435

Alatram A, Sikos LF, Johnstone M et al (2023) DoS/DDoS-MQTT-IoT: a dataset for evaluating intrusions in IoT networks using the MQTT protocol. Comput Networks 231:109809. https://doi.org/10.1016/j.comnet.2023.109809

Alharthi K, Jhumka A, Di S et al (2021) Sentiment analysis based error detection for large-scale systems. In: The 51st IEEE/IFIP International Conference on Dependable Systems and Networks, Taipei, Taiwan, Province of China, 21/06/21. pp 237–249

Alsaheel A, Nan Y, Ma S, Yu L, Walkup G, Celik ZB, Zhang X, Xu D (2021) {ATLAS}: A sequence-based learning approach for attack investigation. Presented at the 30th USENIX security symposium (USENIX Security 21)

Aouini Z, Pekar A (2022) NFStream: a flexible network data analysis framework. Comput Netw 204:108719. https://doi.org/10.1016/j.comnet.2021.108719

Au Yeung JFK, Wei Z, Chan KY et al (2020) Jump detection in financial time series using machine learning algorithms. Soft Comput 24:1789–1801. https://doi.org/10.1007/s00500-019-04006-2

Batal I, Hauskrecht M (2009) A supervised time series feature extraction technique using DCT and DWT. In: 2009 International Conference on Machine Learning and Applications. pp 735–739

Baydogan MG, Runger G, Tuv E (2013) A bag-of-features framework to classify time series. IEEE Trans Pattern Anal Mach Intell 35:2796–2802. https://doi.org/10.1109/TPAMI.2013.72
Bhattacharya S, S SRK, Maddikunta PKR et al (2020) A novel PCA-firefly based XGBoost classification model for intrusion detection in networks using GPU. Electronics 9:219. https://doi.org/10.3390/electronics9020219
Bilge L, Dumitraş T (2012) Before we knew it: an empirical study of zero-day attacks in the real world. In: Proceedings of the 2012 ACM conference on Computer and communications security. Association for Computing Machinery, New York, NY, USA, pp 833–844
Breiman L (2001) Random forests. Mach Learn 45:5–32. https://doi.org/10.1023/A:1010933404324
Breunig MM, Kriegel H-P, Ng RT, Sander J (2000) LOF: identifying density-based local outliers. In: Proceedings of the 2000 ACM SIGMOD international conference on Management of data. Association for Computing Machinery, New York, NY, USA, pp 93–104
Chandola V, Banerjee A, Kumar V (2009) Anomaly detection: a survey. ACM Comput Surv 41(3):1–58
Chaovalit P, Gangopadhyay A, Karabatis G, Chen Z (2011) Discrete wavelet transform-based time series analysis and mining. ACM Comput Surv (CSUR) 43(2):1–37. https://doi.org/10.1145/1883612.1883613
Chen T, Guestrin C (2016) XGBoost: A scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Association for Computing Machinery, New York, NY, USA. pp 785–794
Chen L, Wang Y, Cai Q et al (2021) Sharing more and checking less: Leveraging common input keywords to detect bugs in embedded systems. pp 303–319
Cotroneo D, Natella R, Rosiello S (2017) A fault correlation approach to detect performance anomalies in virtual network function chains. In: 2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE). pp 90–100
Cruz T, Barrigas J, Proença J et al (2015) Improving network security monitoring for industrial control systems. In: 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM). pp 878–881
Dayaratne T, Salehi M, Rudolph C, Liebman A (2022) False data injection attack detection for secure distributed demand response in smart grids. In: Proceedings - 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2022. IEEE, Institute of Electrical and Electronics Engineers. pp 367–380
Downing E, Mirsky Y, Park K, Lee W (2021) {DeepReflect}: Discovering malicious functionality through binary reconstruction. Presented at the 30th USENIX security symposium (USENIX Security 21)
Draper-Gil G, Lashkari AH, Mamun MSI, A. Ghorbani A (2016) Characterization of encrypted and VPN traffic using time-related features. In: Proceedings of the 2nd International Conference on Information Systems Security and Privacy. SCITEPRESS - Science and and Technology Publications, Rome, Italy. pp 407–414
Fu C, Zeng Q, Du X (2021) {HAWatcher}: {Semantics-Aware} Anomaly detection for appified smart homes. Presented at the 30th USENIX security symposium (USENIX Security 21)
Gers FA, Schmidhuber J, Cummins F (2000) Learning to forget: continual prediction with LSTM. Neural Comput 12(10):2451–2471. https://doi.org/10.1162/089976600300015015
Geurts P, Ernst D, Wehenkel L (2006) Extremely randomized trees. Mach Learn 63:3–42. https://doi.org/10.1007/s10994-006-6226-1
Hamerly G, Elkan C (2003) Learning the k in k-means. In: Advances in Neural Information Processing Systems. MIT Press
Haque MdA, Mineno H (2018) Contextual outlier detection in sensor data using minimum spanning tree based clustering. In: 2018 International Conference on Computer, Communication, Chemical, Material and Electronic Engineering (IC4ME2). pp 1–4
Hopfield JJ (1982) Neural networks and physical systems with emergent collective computational abilities. Proc Natl Acad Sci USA 79:2554–2558. https://doi.org/10.1073/pnas.79.8.2554
Hossin M, Sulaiman MN (2015) A review on evaluation metrics for data classification evaluations. Int J Data Min Knowl Manag Process 5(2):1. https://doi.org/10.5121/ijdkp.2015.5201
Huang T, Zhu Y, Zhang Q et al (2013) An LOF-based adaptive anomaly detection scheme for cloud computing. In: 2013 IEEE 37th Annual Computer Software and Applications Conference Workshops. pp 206–211
Izhikevich L, Teixeira R, Durumeric Z (2021) {LZR}: Identifying unexpected internet services. Presented at the 30th USENIX security symposium (USENIX Security 21)
Jha S, Cui S, Tsai T et al (2022) Exploiting temporal data diversity for detecting safety-critical faults in AV compute systems. In: 2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp 88–100
Ji C, Du M, Hu Y et al (2022) Time series classification based on temporal features. Appl Soft Comput 128:109494. https://doi.org/10.1016/j.asoc.2022.109494

Jia T, Wu Y, Hou C, Li Y (2021) LogFlash: Real-time streaming anomaly detection and diagnosis from system logs for large-scale software systems. In: 2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE), pp 80–90. https://doi.org/10.1109/ISSRE52982.2021.00021

Karim F, Majumdar S, Darabi H, Chen S (2018) LSTM fully convolutional networks for time series classification. IEEE Access 6:1662–1669. https://doi.org/10.1109/ACCESS.2017.2779939

Kate RJ (2016) Using dynamic time warping distances as features for improved time series classification. Data Min Knowl Discov 30:283–312. https://doi.org/10.1007/s10618-015-0418-x

Kopetz H, Höftberger O, Frömel B et al (2015) Towards an understanding of emergence in systems-of-systems. https://doi.org/10.1109/SYSOSE.2015.7151925

Kriegel H-P, Schubert M, Zimek A (2008) Angle-based outlier detection in high-dimensional data. In: Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. Association for Computing Machinery, New York, NY, USA, pp 444–452

Lapolli ÂC, Adilson Marques J, Gaspary LP (2019) Offloading real-time DDoS attack detection to programmable data planes. In: 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM). pp 19–27

Li G, Jung JJ (2023) Deep learning for anomaly detection in multivariate time series: approaches, applications, and challenges. Inf Fusion 91:93–102. https://doi.org/10.1016/j.inffus.2022.10.008

Lindemann B, Maschler B, Sahlab N, Weyrich M (2021) A survey on anomaly detection for technical systems using LSTM networks. Comput Ind 131:103498. https://doi.org/10.1016/j.compind.2021.103498

Lindemann B, Jazdi N, Weyrich M (2020) Anomaly detection and prediction in discrete manufacturing based on cooperative LSTM networks. In: 2020 IEEE 16th International Conference on Automation Science and Engineering (CASE). pp 1003–1010

Liu FT, Ting KM, Zhou Z-H (2008) Isolation forest. In: 2008 Eighth IEEE International Conference on Data Mining. pp 413–422

Lyu X, Ding Y, Yang S-H (2019) Safety and security risk assessment in cyber-physical systems. IET Cyber-Physical Systems: Theory & Applications 4:221–232. https://doi.org/10.1049/iet-cps.2018.5068

Markovic T, Dehlaghi-Ghadim A, Leon M et al (2023) Time-series anomaly detection and classification with long short-term memory network on industrial manufacturing systems. In: 2023 18th Conference on Computer Science and Intelligence Systems (FedCSIS). pp 171–181

Meakins J (2019) A zero-sum game: the zero-day market in 2018. J Cyber Policy 4:60–71. https://doi.org/10.1080/23738871.2018.1546883

Muda Z, Yassin W, Sulaiman MN, Udzir NI (2011) Intrusion detection based on K-Means clustering and Naïve Bayes classification. In: 2011 7th International Conference on Information Technology in Asia. pp 1–6

Palmieri F (2019) Network anomaly detection based on logistic regression of nonlinear chaotic invariants. J Netw Comput Appl 148:102460. https://doi.org/10.1016/j.jnca.2019.102460

Pedregosa F (2011) Scikit-learn: machine learning in python fabian. J Mach Learn Res 12:2825

Peng K, Leung VCM, Huang Q (2018) Clustering approach based on mini batch kmeans for intrusion detection system over big data. IEEE Access 6:11897–11906. https://doi.org/10.1109/ACCESS.2018.2810267

Puccetti T, Ceccarelli A (2024a) Detection latencies of anomaly detectors - an overlooked perspective? In: 2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE). pp 37–48

Puccetti T, Ceccarelli A (2024b) Detection latencies of anomaly detectors - an overlooked perspective? In: 2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE). pp 37–48

Puccetti T, Ceccarelli A, Zoppi T, Bondavalli A (2023) Detection of adversarial attacks by observing deep features with structured data algorithms. In: Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing. Association for Computing Machinery, New York, NY, USA, pp 125–134

Puccetti T, Nardi S, Cinquilli C et al (2024) ROSPaCe: intrusion detection dataset for a ROS2-based cyber-physical system and IoT networks. Sci Data 11:481. https://doi.org/10.1038/s41597-024-03311-2

Qin H, Yan M, Ji H (2021) Application of controller area network (CAN) bus anomaly detection based on time series prediction. Veh Commun 27:100291. https://doi.org/10.1016/j.vehcom.2020.100291

Sharafaldin I, Habibi Lashkari A, Ghorbani AA (2018) Toward generating a new intrusion detection dataset and intrusion traffic characterization: In: Proceedings of the 4th International Conference on Information Systems Security and Privacy. SCITEPRESS - Science and Technology Publications, Funchal, Madeira, Portugal, pp 108–116

Skopik F, Settanni G, Fiedler R, Friedberg I (2014) Semi-synthetic data set generation for security software evaluation. https://doi.org/10.1109/PST.2014.6890935
Stojanović B, Hofer-Schmitz K, Kleb U (2020) APT datasets and attack modeling for automated detection methods: a review. Comput Secur 92:101734. https://doi.org/10.1016/j.cose.2020.101734
Striki M, Manousakis K, Kindred D et al (2009) Quantifying resiliency and detection latency of intrusion detection structures. In: MILCOM 2009 - 2009 IEEE Military Communications Conference. pp 1–8
Taheri Sarteshnizi I, Bagloee SA, Sarvi M, Nassir N (2024) Traffic anomaly detection: exploiting temporal positioning of flow-density samples. IEEE Trans Intell Transp Syst 25:4166–4180. https://doi.org/10.1109/TITS.2023.3322695
Tang D, Wang X, Tang H, Zhang K (2021) Demon in the variant: Statistical analysis of {DNNs} for robust backdoor contamination detection. Presented at the 30th USENIX security symposium (USENIX Security 21)
Tsikoudis N, Papadogiannakis A, Markatos EP (2016) LEoNIDS: a low-latency and energy-efficient network-level intrusion detection system. IEEE Trans Emerg Top Comput 4:142–155. https://doi.org/10.1109/TETC.2014.2369958
Verma P, Anwar S, Khan S, Mane SB (2018) Network intrusion detection using clustering and gradient boosting. In: 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT). pp 1–7
Wang X, Lu X (2020) A host-based anomaly detection framework using XGBoost and LSTM for IoT devices. Wirel Commun Mob Comput 2020:8838571. https://doi.org/10.1155/2020/8838571
Wang X, Smith K, Hyndman R (2006) Characteristic-based clustering for time series data. Data Min Knowl Discov 13(3):335–364. https://doi.org/10.1007/s10618-005-0039-x
Wang W, Chen P, Xu Y, He Z (2022a) Active-MTSAD: Multivariate time series anomaly detection with active learning. IEEE Computer Society, pp 263–274
Wang X, Yin K, Ouyang Q et al (2022b) Identifying erroneous software changes through self-supervised contrastive learning on time series data. IEEE Computer Society, pp 366–377
Wijesooriya S (2024) DuseTrive/Anomaly-Based-NID-using-LSTM
Wu S, Wang X, Liang M, Wu D (2021) PFC: a novel perceptual features-based framework for time series classification. Entropy 23:1059. https://doi.org/10.3390/e23081059
Xu M, Li X, Wang Y et al (2021) Privacy-preserving multisource transfer learning in intrusion detection system. Trans Emerging Telecommun Technol 32:e3957. https://doi.org/10.1002/ett.3957
Yuan L-P, Choo E, Yu T et al (2021) Time-window based group-behavior supported method for accurate detection of anomalous users. IEEE Computer Society, pp 250–262
Zhang S, Zhao C, Sui Y, Su Y, Sun Y, Zhang Y, Pei D, Wang Y (2021) Robust KPI anomaly detection for large-scale software services with partial labels. In: 2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE), pp 103–114. https://doi.org/10.1109/ISSRE52982.2021.00023
Zhang K, Zhang C, Peng X, Sha C (2022) PUTraceAD: Trace anomaly detection with partial labels based on GNN and PU learning. In: 2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE), pp 239–250. https://doi.org/10.1109/ISSRE55969.2022.00032
Zhao Z, Cerf S, Birke R et al (2019) Robust anomaly detection on unreliable data. In: 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp 630–637
Zoppi T, Ceccarelli A, Bondavalli A (2021) Unsupervised algorithms to detect zero-day attacks: strategy and application. IEEE Access 9:90603–90615. https://doi.org/10.1109/ACCESS.2021.3090957
Zoppi T, Ceccarelli A, Puccetti T, Bondavalli A (2023a) Which algorithm can detect unknown attacks? Comparison of supervised, unsupervised and meta-learning algorithms for intrusion detection. Comput Secur 127:103107. https://doi.org/10.1016/j.cose.2023.103107
Zoppi T, Merlino G, Ceccarelli A et al (2023b) Anomaly detectors for self-aware edge and IoT devices. In: 2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security (QRS). pp 24–35

Tommaso Puccetti received the M.Sc. degree in Computer Science (curriculum: *Resilient and Secure Cyber-Physical Systems*) from the University of Florence, Italy, in 2021. Since 2021, he has been a Research Fellow at the Department of Mathematics and Informatics of the University of Florence, where he is currently pursuing the Ph.D. degree in Computer Science. His research focuses on machine learning-based anomaly detection, with particular interest in network intrusion detection and the security and dependability of cyber-physical systems. He is involved in a research project on the *design and evaluation of secure systems and applications for the railway domain*. His broader research interests include system dependability, verification and validation of safety-critical systems, and the robustness and security of machine learning components integrated into such systems.

Andrea Ceccarelli received the Ph.D. degree in informatics and automation engineering from the University of Florence, Florence, Italy, in 2012. He is currently an Associate Professor of computer science with the University of Florence. His research interests include the design, monitoring, and evaluation of dependable and secure systems, with a preference for experimental approaches, and his scientific activities originated more than 130 papers. He is regularly involved in the TPC of International Conferences in the domain of dependability and reliability engineering, and he has been the TPC CoChair of SAFECOMP, LADC and SRDS. He has been involved in multiple research projects and he is currently leading his unit in the MUR projects BREADCRUMBS and FLEGREA, and the Tuscany project WAU. He is a member of the IFIP WG 10.4 on “Dependable Computing and Fault-Tolerance.