Future Generation Computer Systems 175 (2026) 108090

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Collaborative anomaly detection in log data: Comparative analysis and evaluation framework

André García Gómez [a,b,*], Max Landauer [a], Markus Wurzenberger [a], Florian Skopik [a], Edgar Weippl [b]

[a] AIT Austrian Institute of Technology GmbH, Vienna, 1210, Austria
[b] University of Vienna, 1010, Vienna, Austria

ARTICLE INFO

Keywords:
Machine learning
CIDS
IDS
Anomaly detection
AI
Log analysis

ABSTRACT

Log Anomaly Collaborative Intrusion Detection Systems (CIDS) are designed to detect suspicious activities and security breaches by analyzing log files using anomaly detection techniques while leveraging collaboration between multiple entities (e.g., different systems, organizations, or network nodes). Unlike traditional Intrusion Detection Systems (IDS) that require centralized algorithm updates and data aggregation, CIDS enable decentralized updates without extensive data exchange, improving efficacy, scalability, and compliance with regulatory constraints. Additionally, inter-detector communication helps to reduce the number of false positives. These systems are particularly useful in distributed environments, where individual system have limited visibility into potential threats. This paper reviews the current landscape of Log Anomaly CIDS and introduces an open-source framework designed to create benchmark datasets for evaluating system performance. We categorize log anomaly detectors into three categories: Sequential-wise, Embedding-wise, and Graph-wise. Furthermore, our open framework facilitates rigorous evaluation against different challenges identifying weaknesses in existing methods like Deeplog and enhancing model robustness.

1. Introduction

Information and communication systems are increasingly growing in size and complexity while becoming more essential in our daily lives. As mentioned in [1], cyber attacks on these systems pose a substantial threat to society, consistently endangering them. Automating certain tasks, such as cybersecurity, is vital to keep up with this trend. This subject has been discussed before; 25 years ago, IBM released a manifesto [2] advocating for the development of self-managing systems that can autonomously configure, repair, and secure themselves. The core idea is that systems are evolving to a level of complexity that human maintenance is not feasible, thereby emphasizing the necessity of creating automated tools for these tasks. Over the past two decades, publications have addressed this topic. Early approaches to Collaborative Intrusion Detection Systems (CIDS) can trace back to 2003. For instance, [3] investigated CIDS to enhance intrusion detection across distributed systems. In 2012, [4] motivated their research in creating parsers capable of automatically processing logs to aid in the maintenance of large systems. Another example is from 2015, when LogCluster [5] was developed to autonomously detect system failures through log data analysis and was implemented in Microsoft online services. The developers underscored the importance of such a tool, as these services generated petabytes of logs each day, an amount too vast for manual human analysis. In alignment with this thought process, our work will focus on current methods employed to automatically analyze logs to identify anomalous behavior or intrusion attacks in decentralized systems known as Log Anomaly CIDS [6].

It is important to note that not all CIDS utilize logs [7] for anomaly detection, nor do all CIDS examine anomalous behavior. A more detailed discussion will be provided in the paper. While our primary focus is on Log Anomaly CIDS, we will also explore other Anomaly CIDS and comparable methods to obtain a broader understanding of the current state of the art and emerging trends. This paper revolves around the following research questions (RQ).

RQ1: What methods and baselines are used in the literature for Anomaly CIDS? We will explore different methods to detect anomalies in CIDS from recent years. As noted previously, not all Anomaly CIDS rely solely on logs; some utilize network traffic packets or diverse time series outputs from multiple sensors. To broaden our investigation, we will also consider Intrusion Detection Systems (IDS), which function

* Corresponding author.
E-mail addresses: andre.garcia-gomez@ait.ac.at (A. García Gómez), max.landauer@ait.ac.at (M. Landauer), markus.wurzenberger@ait.ac.at (M. Wurzenberger), florian.skopik@ait.ac.at (F. Skopik), edgar.weippl@univie.ac.at (E. Weippl).

https://doi.org/10.1016/j.future.2025.108090
Received 2 March 2025; Received in revised form 14 August 2025; Accepted 17 August 2025
Available online 24 August 2025
0167-739X/© 2025 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

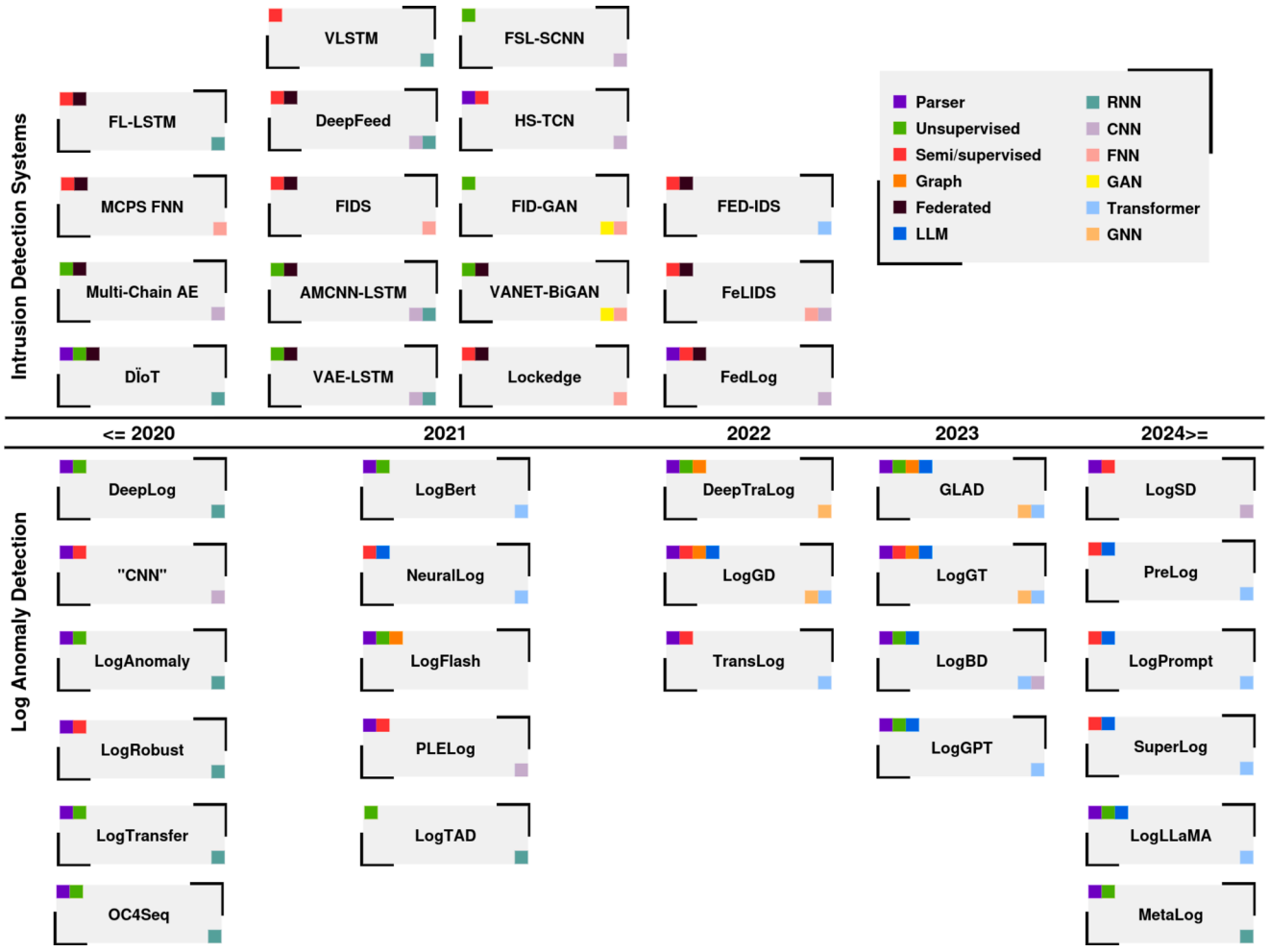


Fig. 1. The literature review distinguishes between publications that introduce new Intrusion Detection Systems (IDS) and Collaborative Intrusion Detection Systems (CIDS) methods on the upper section and those identified via Log Anomaly Detection (LAD) on the lower section. Each method is color-coded to indicate the presence of specific keywords and the type of machine learning architecture employed. All the publications are sorted in chronological order.

similarly to CIDS but focus on centralized systems. Furthermore, we will examine Log Anomaly Detection (LAD) approaches, which involve algorithms that scan logs to identify unusual behavior in a similar way as Log Anomaly CIDS. Fig. 1 illustrates all the publications referenced in this study, all the methods considered are machine learning based. We compile the articles by employing the snowball technique on Google Scholar, utilizing the previously specified keywords.

RQ2: How can we categorize the different Log Anomaly CIDS and other Log Anomaly approaches? We will organize our classification based on all the logging methods collected in RQ1. Other studies, such as [8] or [9], often categorize methods according to the architecture of the machine learning model. Although this aspect has significance, we believe that it is not the most crucial element. Since machine learning models are data-driven algorithms that depend on an optimization process, our categorization will emphasize data pre-processing techniques and the objective functions employed during training.

RQ3: To what extent can we improve the reliability of Log Anomaly CIDS? To effectively create and deploy these systems in practical environments, it is crucial to ensure their robustness. We have created an open-source framework intended to produce well-organized datasets, enabling preliminary assessment of these techniques in diverse contexts. Our main objective is to provide tools to allow for a thorough analysis that support the creation and evaluation of more sophisticated and robust algorithms in this domain. Although this study will not definitively resolve RQ3, it will provide tools to address it in future research.

The corresponding code is available for access here [10].

The paper is organized into these sections: In Section 2, a comparison with prior studies on the same subject is presented. Section 3 introduces the concept of a CIDS and examines how multiple nodes can collaborate for intrusion detection in decentralized systems. Section 4 evaluates current state-of-the-art trends to address RQ1. In Section 5, we analyze the log processing in Log Anomaly CIDS and categorize them, focusing on RQ2. Section 6 discusses existing benchmarks and shows how our open-source framework addresses RQ3. Section 7 includes a demonstration of how our framework can be used to compare different models. Section 8 answers the research questions based on insights from previous sections. Finally, Section 9 covers conclusions and future work.

2. Related work

The literature includes several SoK papers and surveys focused on CIDS, such as those on Log Anomaly CIDS. In addition, there are numerous surveys that discuss Log Anomaly methods in more general. We have classified these into distinct categories on the basis of their content.

- *CIDS Taxonomies*: The extensive topic of CIDS allows multiple taxonomies to be formulated depending on which CIDS aspect is being emphasized. The 2015 work by [11] offers a CIDS classification that is broader than what is used in this paper, with less emphasis on specific algorithms, but a stronger focus on commercial variants. In contrast, [8] highlights the role of federated learning in CIDS and

presents various concepts at a more introductory level, lacking comprehensive technical details.

- *IDS Surveys*: Intrusion detection systems (IDSs) are generally divided into two primary categories, as we will discuss later. In the survey by [12], these are further broken down into several subclasses. Furthermore, [13] enumerates different tactics used to avoid IDS detection.
- *SoK Publications*: The work of [6] provides an extensive overview of the entire detection system pipeline. However, their primary concern is about reducing the volume of logs processed by the CIDS, which diverges from the focuses of this study.
- *Log Anomaly Surveys*: The study by [9] thoroughly investigates various deep learning methods to detect log anomalies, although it remains theoretical and does not incorporate IDS practically. Similarly, [14] discusses different strategies, concentrating mainly on the clustering of different logs.

Our research enhances prior studies in Log Anomaly CIDS by offering a more detailed categorization of prevalent algorithms. In addition, we introduce innovative tools, such as our framework, to support the development of these systems. We assert that the perspective offered in this paper is distinctive and has not previously been addressed in the existing literature.

3. Anomaly CIDS

Anomaly CIDS covers an extensive and cross-disciplinary topic. Initially, this section will clarify the concept of an IDS before diving into multiple CIDS and highlighting their advantages over IDS in complex environments. We define Anomaly CIDS as those designed to recognize anomalies, and Log Anomaly CIDS specifically as those that utilize log inputs for anomaly detection. Ultimately, we will explore the prevalent trends within the reviewed literature for this study. This section aims to provide a thorough overview that extends beyond just Log Anomaly CIDS, whereas the subsequent sections will focus on examining the algorithms used for detecting log-based anomalies.

3.1. Intrusion Detection Systems (IDS)

Intrusion Detection Systems (IDSs) were created to facilitate automated protection of systems against threats. Their primary role is to recognize attacks through the examination of data generated by the system. IDSs are integrated into a broader framework for analyzing and evaluating the behavior of the entire infrastructure. Based on the research presented in [6], Fig. 2 illustrates a standard pipeline designed to collect and analyze logs in multiple IDS algorithms. Similar pipelines can be found in LogLens [7] and in CIDS approaches [3]. The primary functions are as follows. First, there is the capture layer, whose fundamental role is to gather logs from the system's various nodes. Next, the reduction layer processes these logs; as highlighted in [5], the volume of logs can be excessive, so the critical function of this layer is to sort and eliminate redundant logs to reduce the burden on the subsequent layer. However, determining which logs are redundant can be complex and poor decisions could negatively influence IDS performance. This issue is beyond the scope of this document; more comprehensive details are provided in [6]. Following this, the infrastructure layer is responsible for the storage of the various logs. Finally, at the peak of the diagram, the detection and investigation layers are found. The detection element is automatically managed by IDS techniques, while the investigation layer offers users a platform to track alerts and oversee the system. An actual example of an open source initiative that follows this architecture is documented in [15].

As mentioned above, the detection layer enables multiple IDSs to collaborate. The literature identifies various types of IDSs. Signature-based methods excel at identifying known threats, but maintaining their currency is both burdensome and labor-intensive. In addition, they fail to detect unknown threats and can be easily bypassed by sophisticated adversaries such as advanced persistent threats (APTs). On the other hand,

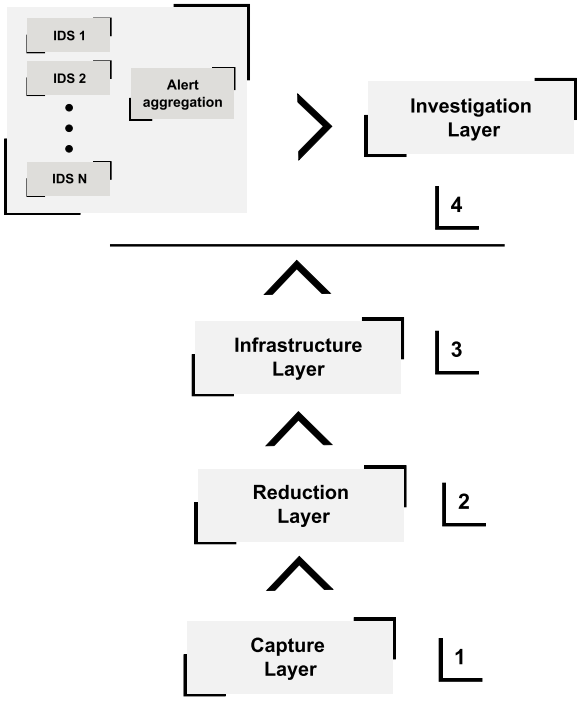


Fig. 2. Generic alert system architecture based on the previous work of [6] and [16].

anomaly-based IDSs identify deviations from typical system and user behaviors, allowing them to detect zero-day attacks. However, they are prone to generate excessive alerts through false positives and duplicates, which requires alert aggregation strategies to minimize the number of alerts presented to users [16].

3.2. Collaborative IDS (CIDS)

As technologies like the Internet of Things (IoT) expand, information and communication systems have evolved to be more intricate and decentralized. Consider a typical fog architecture as described in [17], which includes three main layers for distributed computation as shown in Fig. 3. First, the IoT layer is primarily used to collect data from sensors in various locations. Second, the fog layer serves as the connecting element of the overall system. Third, the cloud layer hosts the nodes with

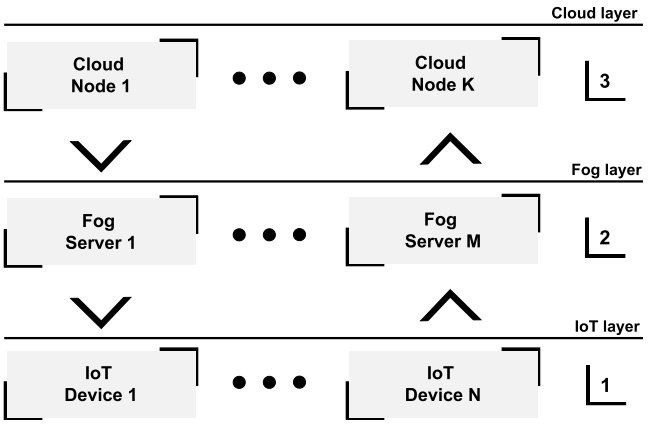


Fig. 3. The foundational fog architecture as described in [17] comprises K Cloud nodes, M fog servers, and N IoT devices. The numbering system organizes the layers from those nearest to those farthest from the user's view, assuming interaction with the IoT devices.

the highest computational capacity. This fog architecture can operate on multiple devices located in various global locations. Such a decentralized system does not integrate well with a centralized IDS, which may encounter issues like scalability or latency. CIDS were developed to address these challenges. These systems comprise two components: monitor nodes and analysis nodes. For simplicity, it is assumed that the capture layer resides in the monitor nodes, while the remaining layers are located on the other ones. According to [11], CIDS can be categorized into three types:

- **Centralized CIDS**: Their operation is similar to regular IDS, as they incorporate just one analysis node. Nevertheless, for scalability, they possess numerous monitoring nodes distributed across the system.
- **Decentralized CIDS**: The system employs a hierarchical configuration among the analysis nodes, with each monitoring node linked to a particular analysis node.
- **Distributed CIDS**: Generally, it employs a Peer-2-Peer structure where each node simultaneously performs analysis and monitoring.

Centralized CIDS are susceptible to failure due to their single point of vulnerability and do not scale as efficiently as other methods. Consequently, this study will not focus on centralized CIDS, as existing literature on Log Anomaly CIDS assumes that multiple analysis nodes come into play.

Decentralized systems are generally more challenging to develop and maintain than centralized systems, introducing unique obstacles for CIDS that IDS do not encounter. These systems depend on message exchanges between various nodes, posing a risk of interception or alteration by attackers. Additionally, when updates are made to the detection methods utilized in CIDS, not all nodes may be accessible, leading to nodes running different software versions. As highlighted in [8], a significant number of such methods in current research employ the FedAvg [18] algorithm for model training. The Pseudo-code Algorithm 1 describes a generic algorithm for this approach. Initially, it involves sampling N clients (Line 3). If the connection to a node in the sample using the $not\_fail()$ method is successful (line 4), the node is instructed to perform local training and return its local weights (Lines 5-6). These local weights are then merged through an aggregation process (Line 9). Finally, all available nodes are updated with the new global weights (Line 12). It is important to note that it is unnecessary for all nodes to participate in the training process, but the final model must be updated for all accessible nodes. In FedAvg, the aggregation function calculates a weighted average of the weights. The literature presents alternative aggregation techniques Fed+ [19] as well as modifications to FedAvg, such as DDFef [20]. As noted above, adversaries can intercept messages sent during the federated learning training process. This can alter the final model's performance and potentially create weaknesses in detection systems that can be exploited. Related literature on these types of attack is cited as [21–23].

As revealed in the survey by Zhang et al. [24], FedAvg is used in various disciplines. It is a fundamental training technique in federated deep learning with proof of convergence [25]. A key challenge in CIDS applications is the prohibition against transferring data to a central location due to privacy concerns. However, training techniques such as FedAvg eliminate this need, allowing each node to leverage the training data from other nodes effectively. In an empirical study conducted by Rahman et al. [26], three use cases were assessed by comparing federated with centralized CIDS, demonstrating that federated outcomes can closely approximate centralized performance depending on the data distribution among nodes, although typically results tend to be inferior. Campos et al. [27] conducted a similar study, evaluating FedAvg and Fed+ [19] in various dataset distributions, highlighting the critical role of data distribution and showing that Fed+ generally outperforms FedAvg. Communication costs can be further reduced in training by adapting FedAvg variations such as LotteryFL [28], which involves only transmitting a subset of the model to the central hub according to the hypothesis of the lottery ticket [29], exemplified in FedLog [30], a Log Anomaly CIDS. Another example is AMCNN-LSTM [31], where compressed gradients are transmitted instead of model weights.

Algorithm 1 Federated learning generic pseudo-code.

```
1:  for round in rounds do
2:      Ws_t ← []                                  ▷ List of trainable weights
3:      for n in sample_nodes(N) do                ▷ Sample N clients
4:          if n.not_fail() then
5:              n.do_train(local_epochs)
6:              Ws_t.append(n.get_weights())
7:          end if
8:      end for
9:      W_{t+1} ← aggregate_weights(Ws_t)
10:     for n in Nodes do
11:         if n.not_fail() then
12:             n.set_weights(W_{t+1})
13:         end if
14:     end for
15: end for
```

4. Literature discussion

As depicted in Fig. 1, this research investigates scholarly articles using the keywords *Log Anomaly Detection* (LAD) and *Intrusion Detection Systems* (IDS). This section will highlight the unique traits of these two fields. Our approach used the snowball methodology. We started by conducting a search on Google Scholar for articles with more than 300 citations dated between 2020 and 2024, or recent works that may capture the interest of the field. After accumulating a set of approximately 7 papers, we examined the citations listed in their related work sections. Those citations deemed essential were included if they formed the foundation for the authors' work, were used in their experiments, or appeared multiple times, specifically more than three, in various related studies. This approach was repeated until no additional relevant papers could be found. Articles were considered pertinent and included only if they demonstrated innovative methods that set them apart from others. If the method appeared irrelevant or unrelated to the topic, it was excluded. Furthermore, we included some papers due to their compelling empirical experiments [26] or interesting use cases [32]. LAD is primarily focused on identifying anomalies in log data, which can originate from both system malfunctions and cyber attacks. In contrast, works related to CIDS/IDS generally concentrate solely on anomalies caused by attacks within various data contexts. It should be noted that methods from these fields can often be used interchangeably, as illustrated by [33], where an LAD method [34] was applied within a CIDS setting. Table 1 lists the three types of dataset used:

- **Log Datasets**: datasets derived from the system's generated logs.
- **Network traffic Datasets**: datasets that examine the transmission of packages across a network. Most of these fields consist of numerical values such as byte size, sending time, and others.
- **Time series**: numerical sequences over time observed in various industrial contexts.

Table 1
Datasets found in the literature.

Datasets	Publications
Log	HDFS [40], BGL [41], Thunderbird [41], Hadoop [42], OpenStack [42], Rubis [43], Spirit [41], Spark [44] , AIT-LDS [45]
Network traffic	NSL-KDD [46], Bot-IoT [47], KDD99 [48], SWaT [49], WADI [49], UNSW-NB15 [50], TONi_IoT [51], Car Hacking [52]
Time series	Space Shuttle [53], Respiration [53], ECG [53], Power demand [53], Gesture [53] Gas pipeline [54], Nyc taxi [55]

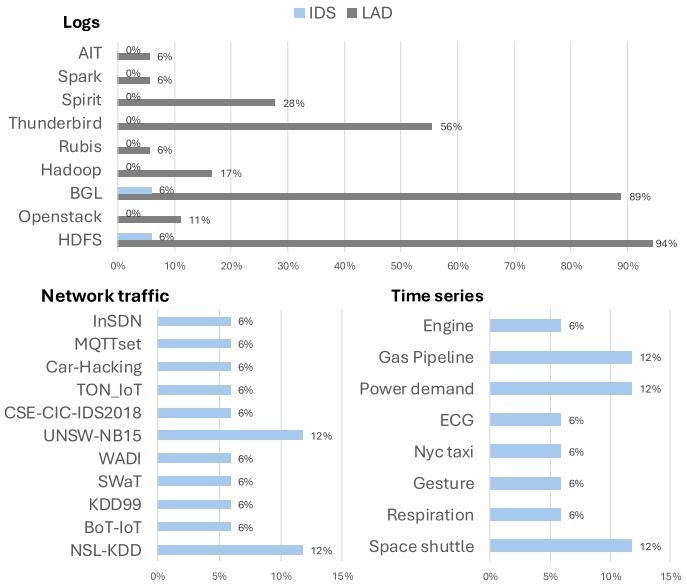


Fig. 4. Distribution of different datasets used in the literature related to publications on Log Anomaly Detection (LAD) and Intrusion Detection Systems (IDS).

Two additional datasets identified in the literature but not within these groups include the SEA dataset [35], which is derived from various UNIX commands and employed in [36], and MIMIC [37], a dataset concerning critical care utilized by [38]. Fig. 4 illustrates the distribution across multiple datasets referenced in the literature. Examining LAD among log datasets reveals that HDFS, BGL, and Thunderbird serve as primary baselines in numerous studies. Conversely, CIDS/IDS research is fragmented across different clusters, lacking a predominant baseline dataset, which complicates comparative analysis between methodologies. Notably, only two IDS methodologies are equipped to identify log anomalies [30,39], indicating a distinct deficit in CIDS/IDS solutions for log data. Studies that created their own datasets merely as a concept to showcase their specific method's performance were excluded from Table 1 and Fig. 4.

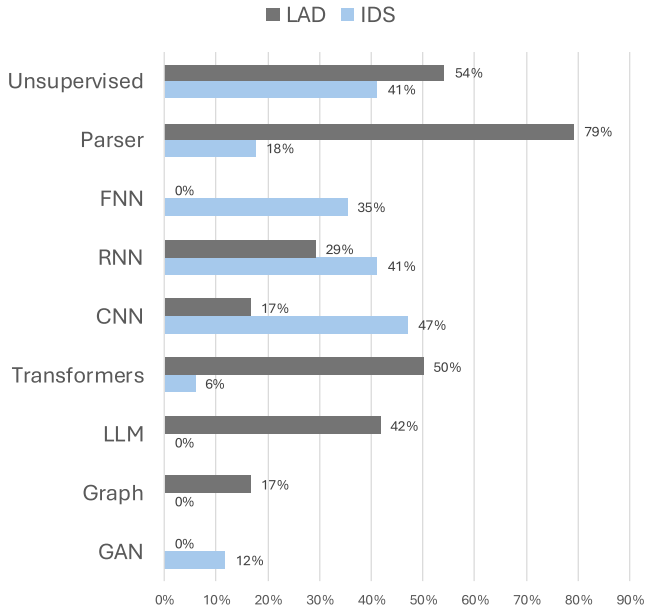


Fig. 5. Distribution of primary keywords in the literature related to publications on Log Anomaly Detection (LAD) and Intrusion Detection Systems (IDS).

We further explore the different methodologies cited in the literature (refer to Fig. 5), highlighting a divergence between how LAD and CIDS/IDS publications are distributed. LAD techniques employ contemporary Natural Language Processing models, such as transformers and Large Language Models, in addition to methods based on graph theory. Conversely, CIDS/IDS approaches tend to opt for broader deep learning techniques. This pattern is largely due to two reasons: first, as mentioned earlier, most CIDS/IDS research does not center on log datasets, lessening the demand for NLP methods; second, nonlog datasets are generally more straightforward, allowing techniques like Full Neural Networks (FNNs) to achieve strong results. The methodologies compared are detailed in Tables 2 and 3.

5. Log Anomaly Detection

In the preceding section, we have generally examined various Anomaly CIDS / IDS systems without diving into the specific techniques applicable to them. In the upcoming segment, we will offer a concise introduction to logs and explore the different strategies employed by multiple Log Anomaly Detection methods, as documented in the literature.

5.1. Logs and log events

A log is made up of two main elements: a fixed section and a variable section. The fixed portion functions as the template for the log message, while the variable part includes a set of values that change according to the state of execution. For example:

$$\text{log} = \text{“Connection to 127.0.0.1 was accepted”}. \tag{1}$$

The process involves separating the logs into two parts, $log = (t, v)$, where t represents the template and v corresponds to the variable list. In many real-world situations, templates are not accessible. Consequently, logs are managed as string variables instead of tuples. Several parsers have been created to address this issue, as noted in the literature: [4,90,91]. These parsers can be described by Eq. (2).

$$e, (t, v) = parser(log) \tag{2}$$

Where t = *“Connection to <*>was<*>”* and $v = [127.0.0.1, accepted]$. These approaches produce the most probable template and assign it an event number, also known as a log event e. It is important to recognize that the parsers are not infallible and cannot ensure that $(t^*, V^*) == (t, V)$ where (t^*, V^*) is the accurate result. For example, when using one of the common parsing methods, such as Drain [90], subsequent logs associated with the same initial template might be mistakenly classified under a different template and event number:

$$\begin{aligned} log_1 &= \text{“The operation was: successful”}, \\ log_2 &= \text{“The operation was: Exception in line 25,...”}. \end{aligned} \tag{3}$$

This occurs because Drain initially segments the logs based on word count, ensuring that they are not assigned the same event ID, even if they originate from an identical template. In this paper, we will not investigate into further specifics regarding the parsers. Nonetheless, it is essential to grasp the following attributes before transitioning to Log Anomaly Detection techniques:

- Parsers are not flawless; any mistakes they produce will be transferred to the Log Anomaly Detection methods that depend on them.
- Parsers such as Drain [90] and Spell [91] are updated in real time, implying that they undergo continuous modifications. This poses a challenge for Log Anomaly Detection methods that are updated in a non-continuous manner, including deep learning techniques.

It should be noted that new parsers that employ LLM techniques are in development and have the potential to significantly reduce parsing errors [83,92].

Table 2
Collaborative and Intrusion Detection Systems publications.

Method	Keywords	Architecture	Dataset
FL-LSTM [36]	Semi/supervised, Federated	RNN	Sea
MCPS FNN [38]	Semi/supervised, Federated	FNN	MIMIC
Multi-Chain AE [56]	Unsupervised, Federated	CNN	Created
DÏoT [39]	Parser, Unsupervised, Federated	RNN	Created
VLSTM [57]	Semi/supervised	RNN	UNSW-NB15
DeepFeed [58]	Semi/supervised, Federated	CNN, RNN	Gas Pipeline
FIDS [59]	Semi/supervised, Federated	FNN	NSL-KDD
AMCNN-LSTM [31]	Unsupervised, Federated	CNN, RNN	Space shuttle, Power demand, Engine
VAE-LSTM [60]	Unsupervised, Federated	CNN, RNN	Space shuttle, Respiration, Gesture, Nyc taxi, ECG, Power demand, Gas Pipeline
FSL-SCNN [61]	Unsupervised	CNN	Created, UNSW-NB15
HS-TCN [62]	Parser, Semi/supervised	CNN	Created
FID-GAN [63]	Unsupervised	FNN, GAN	NSL-KDD, SWaT, WADI
VANET-BiGAN [64]	Unsupervised, Federated	FNN, GAN	KDD99
Lockedge [65]	Semi/supervised, Federated	FNN	Bot-IoT
FED-IDS [32]	Semi/supervised, Federated	Transformer	TON_IoT, CarHacking
FeLIDS [66]	Semi/supervised, Federated	CNN, FNN	CSE-CIC-IDS2018, MQTset, InSDN
FedLog [30]	Parser, Semi/supervised, Federated	CNN	HDFS, BGL

Table 3
Log Anomaly Detection.

Method	Keywords	Architecture	Dataset
DeepLog [34]	Parser, Unsupervised	RNN	HDFS, Openstack
CNN [67]	Parser, Semi/supervised	RNN	HDFS
LogAnomaly [68]	Parser, Unsupervised	RNN	HDFS, BGL
LogRobust [69]	Parser, Semi/supervised	RNN	HDFS
LogTransfer [70]	Parser, Unsupervised	RNN	HDFS, Hadoop, Created
OC4Seq [71]	Parser, Unsupervised	RNN	HDFS, BGL, Rubis
LogBert [72]	Parser, Unsupervised	Transformer	HDFS, BGL, Thunderbird
NeuralLog [73]	Semi/supervised, LLM	Transformer	HDFS, BGL, Thuderbird, Spirit
LogFlash [74]	Parser, Unsupervised, Graph	-	Created
PLELog [75]	Parser, Semi/supervised	CNN	HDFS, BGL
DeepTraLog [76]	Parser, Unsupervised, Graph	GNN	Created
LogGD [77]	Parser, Semi/supervised, LLM, Graph	Transformer, GNN	HDFS, BGL, Thunderbird, Spirit
TransLog [78]	Parser, Semi/supervised	Transformer	HDFS, Hadoop, Thunderbird
GLAD [79]	Parser, Unsupervised, Graph	Transformer, GNN	BGL, AIT, Created
LogGT [80]	Parser, Semi/supervised, Graph, LLM	Transformer, GNN	HDFS, BGL, Thunderbird
LogBD [81]	Parser, Unsupervised, LLM	CNN, Transformer	Hadoop, Thunderbird
LogSD [82]	Parser, Semi/supervised	CNN	HDFS, BGL, Spirit
PreLog [83]	Semi/supervised, LLM	Transformer	HDFS, BGL, Spark
LogPromt [84]	Semi/supervised, LLM	Transformer	BGL, Spirit
SuperLog [85]	Semi/supervised, LLM	Transformer	BGL, Spirit
LogGPT [86]	Parser, Unsupervised, LLM	Transformer	HDFS, BGL, Thunderbird
LogLLaMA [87]	Parser, Unsupervised, LLM	Transformer	HDFS, BGL, Thunderbird
LogTAD [88]	Unsupervised	RNN	BGL, Thunderbird
MetaLog [89]	Paser, Unsupervised	RNN	HDFS, BGL, Thunderbird, Openstack

5.2. Methods

After reviewing the different publications relevant to Log Anomaly CIDS/IDS and LAD, we classified the methods into three separate groups. In the following parts, we will describe the different categories using pseudo-codes and process tables with an example of an anomaly sequence as aids.

5.2.1. Sequential-wise Log Anomaly Detection

To the best of our knowledge, this category was the first of its kind in the literature and is still used today (Algorithm 2). The core concept involves converting each log entry into an event ID according to Eq. (2) (Line 5). If an event was not included in the training set, it is deemed anomalous (Line 6); otherwise, the events are combined into an event sequence, which is then processed by a model to determine if it is a normal sequence (Lines 12-13). Table 4 presents an example illustrating the various outputs produced at each step. The first iterations of this method can be traced back to publications from the previous century [93]. In the realm of literature, Deeplog [34] is noteworthy as the first deep learning model in this category, employing an LSTM [94] with the subsequent formulation:

$$Loss = -log(P_\theta(e_t \mid e_{t-1}, e_{t-2}, \dots, e_0)). \quad (4)$$

Algorithm 2 Sequential-wise Log Anomaly Detection generic pseudo-code.

```
1:  model ← load_model()                          ▷ Machine learning model
2:  logs ← [log_1, log_2, .., log_n]              ▷ Input log sequence
3:  event_seq ← []
4:  for log in logs do
5:      e, _ ← parser(log)                        ▷ e ∈ Events
6:      if e ∉ know_events then
7:          return True
8:      end if
9:      event_seq.append(e)
10: end for
11: event_seq ← preprocessing(event_seq)          ▷ Events^{nx1} → ℝ^{nxm}
12: output ← model(event_seq)                     ▷ ℝ^{nxm} → ℝ^{nxo}
13: return is_anomaly(output)                     ▷ ℝ^{nxo} → {True, False}
```

Table 4
Example of the intermediate outputs of the steps in the Sequential-wise methods following Algorithm 2.

Step	Sub-Method (optional)	Output
Input	log_1, log_2, log_3	
For loop	parser(log_1)	e_1
	parser(log_2)	e_2
	parser(log_3)	e_3
Event Seq.	[e_1, e_2, e_3]	
Preprocessing	[([e_1, e_2], e_3)]	
Model	[$score_1$]	
Is Anomaly	[True]	

The model P_θ is trained to forecast the subsequent event in the sequence based on prior events. A comparable approach was employed in LogBert [72], which utilized a transformer [95] with BERT [96] masking loss alongside the Deep SVDD hypersphere loss [97]:

$$Loss = -log(P_\theta(e \in M \mid e \notin M)) + \alpha L_{SVDD}, \tag{5}$$

where:

$$L_{SVDD} = ||h_{DIST} - c||^2. \tag{6}$$

Here, M refers to the collection of events that are masked and meant to be predicted using the observable events. Meanwhile, h_{DIST} denotes the embedding space of a special token, encapsulating the information of the sequence, and c is the mean of h_{DIST} across all sequences within the training batch. The authors assert that incorporating both losses improves performance, and the masking loss is more effective than Eq. (4) for handling log data. Similarly to Eqs. (4) and (5), various models have been developed. LogAnomaly [68], for example, expands on the DeepLog framework with a template2Doc method to improve semantic retrieval. Similarly, LogTransfer [70] is designed to enable knowledge transfer across different datasets, while OC4Seq [71] combines two GRUs with SVDD hypersphere loss. In the realm of CIDS, DÏoT [39] utilizes a GRU [98] instead of an LSTM, and FedLog [30] employs a dual-input model, applying unique pre-processing techniques for each input. Furthermore, there are semi-supervised approaches like PLELog [75], which uses a probabilistic label estimation method, and LogSD [82] that incorporates distillation techniques with a framework featuring two encoders and a single decoder.

5.2.2. Embedding-wise Log Anomaly Detection

Sequential-wise anomaly detection techniques for logs are straightforward to implement, but present several challenges that embedding-based approaches aim to tackle (Pseudo-code 3). Firstly, Sequential-wise methods neglect the information contained within the log messages and heavily rely on parsers. Moreover, the introduction of new or altered logs from system updates can dramatically affect the model's performance. Studies such as LogRobust [69] seek to mitigate these shortcomings in anomaly detection. They achieve this by forgoing the use of event ID numbers and, instead, utilizing the templates obtained by the parsers (Line 6) as model input. This is accomplished through a word embedding layer (Line 9) known as FastText [99], which translates template words into an embedding space. Subsequently, a BiLSTM is employed to process the information, with the primary formulation being:

$$Loss = -log(P_\theta(y \mid E_\varphi(t_n), E_\varphi(t_{n-1}), \ldots, E_\varphi(t_0))) \tag{7}$$

where:

$$E_\varphi(t) = \sum_{word}^{word \in t} \text{idf}(word) \cdot \text{FastText}_\varphi(word). \tag{8}$$

We refer to $E_\varphi(\cdot)$ as the embedding encoder. Table 5 provides an illustration of the various steps involved in this approach. Typically, in many studies, this encoder originates from a pre-existing model with pretrained parameters, and during training, these parameters φ remain unaltered. Regrettably, the majority of the techniques adhering to Algorithm 3 are supervised, which constrains their application as an IDS element for identifying novel attacks, as indicated in Eq. (7). Analogous research like NeuralLog [73] operates without a parser and encodes logs using a pre-trained Bert model [96], whereas PreLog [83] is a large language model (LLM) trained on log data, capable of identifying supervised log anomalies through tailored prompts. Similar prompt-based LLM methods involve LogPrompt [84] and SuperLog [85]. Alternatively, some strategies use LLMs integrated with reinforcement learning for fine-tuning instead of relying on prompts, as seen in recent examples like LogGPT [86] and LogLLaMA [87]. Other approaches that do not precisely follow this pattern but fit within the same category include the method widely referenced as "CNN" in the academic literature [67], which utilizes a trainable embedding layer alongside a basic CNN framework [100].

Table 5
Example of the intermediate outputs of the steps in the Embedding-wise methods following Algorithm 3.

Step	Sub-Method (optional)	Output
Input	log_1, log_2, log_3	
For loop	parser(log_1)	t_1
	parser(log_2)	t_2
	parser(log_3)	t_3
Event Seq.	[t_1, t_2, t_3]	
Emb. Model	[emb_1, emb_2, emb_3]	
Model	[$score_1$]	
Is Anomaly	[True]	

Algorithm 3 Embedding-wise Log Anomaly Detection generic pseudo-code.

```
1: model ← load_model()                          ▷ Machine learning model
2: emb_model ← load_emb_model()                  ▷ Embedding model
3: logs ← [log_1, log_2, .., log_n]              ▷ Input log sequence
4: event_seq ← []
5: for log in logs do
6:     _, (t, _) ← parser(log)                   ▷ t ∈ Templates
7:     event_seq.append(t)
8: end for
9: emb_seq ← emb_model(tokenizer(event_seq))     ▷ Templates^{nx1} → ℝ^{nxm}
10: output ← model(emb_seq)                      ▷ ℝ^{nxm} → ℝ^{nxo}
11: return is_anomaly(output)                    ▷ ℝ^{nxo} → {True, False}
```

An alternative method, LogBD [81], uses domain adaptation between a target and a source dataset, integrates a Bert encoder [96], and is trained without supervision using SVDD hypersphere loss (Eq. (6)). It employs adversarial training, applying gradient reversal [101], to mediate between the target and source datasets and help the generalization of the model:

$$L_{adv} = \min_\alpha \max_\beta \begin{pmatrix} \mathbb{E}_{\forall h_s \in Source}[log(D_\beta(f_\alpha(E_\varphi(h_s))))] + \\ \mathbb{E}_{\forall h_t \in Target}[log(1 - D_\beta(f_\alpha(E_\varphi(h_t))))] \end{pmatrix} \tag{9}$$

where f_α comprises learnable layers appended to the frozen Bert model E_φ, and D_β denotes a discriminator classifier responsible for distinguishing between source and target datasets. The goal of gradient reversal is to confuse the discriminator, thereby hindering it from discerning the two distributions. The final objective function of LogBD:

$$Loss = L_{SVDD} - \lambda L_{adv}. \tag{10}$$

LogTAD [88] utilizes a comparable technique, replacing the architecture with an LSTM. In contrast, MetaLog [89] adopts a meta-learning strategy rather than adversarial training to achieve domain adaptation.

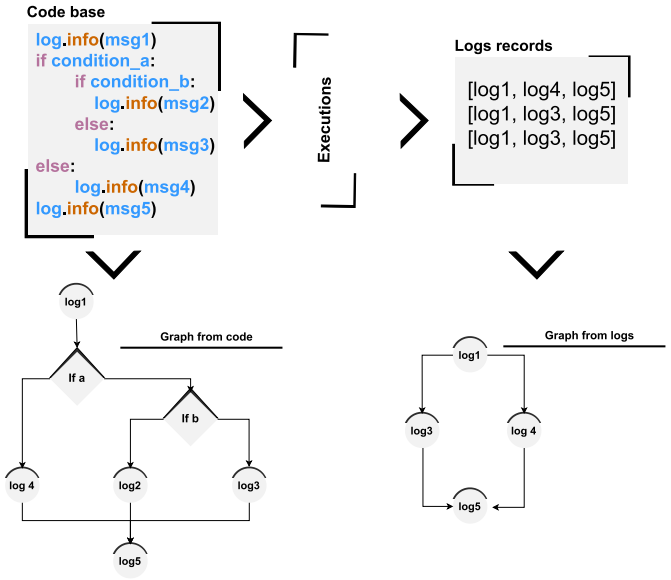


Fig. 6. Example of a graph generations and their main difference between using the code base and the log records.

5.2.3. Graph-wise Log Anomaly Detection

As demonstrated in previous methods, log data can be addressed using Natural Language Processing (NLP) techniques. In sequentially-wise, each log entry is defined by its relationship to other logs, independent of the intrinsic syntax of the log message. In terms of Embedding-wise, it also incorporates the information within the log message, with both approaches handling the data as a sequential chronology of tokens. Nevertheless, NLP problems can also be modeled as graphs [102]. Consider a series of logs from a deterministic program in which their interdependencies arise from the execution state. These relationships can be illustrated as a graph structure, potentially offering a more sophisticated problem representation. As shown in Fig. 6, it is feasible to develop a graph model of a code base using only the sequence of logs of each run. This graph does not strictly match the structural layout of the original code base, but maintains logical consistency (for example, $log3$ never precedes $log1$). It is important to note that since $condition\_b$ was consistently false during executions, $log2$ is omitted from the graph generated by the logs.

Consider defining graphs as $G = (V, E)$, with V representing the nodes (each event id is symbolized by a node) and E representing the edges such that $E = \{(e_1, e_2) | e_1, e_2 \in V\}$. Fig. 6 illustrates how logs can be depicted in this manner, enabling the use of graph anomaly detection techniques for the problem at hand, as detailed in the survey by [103]. The Algorithm 4 presents a standard pseudo-code where T and E denote the template and event sets, respectively. This algorithm is analogous to Algorithm 3 but processes sequences without considering their chronological order by generating graphs instead (Line 10), an illustration of intermediate outcomes is provided in Table 6. We can categorize various methods within this class based on their approach to constructing the initial graph:

Algorithm 4 Graph-wise Log Anomaly Detection generic pseudo-code.

```
1: model ← load_model()                          ▷ Machine learning model
2: emb_model ← load_emb_model()                  ▷ Embedding model
3: logs ← [log_1, log_2, .., log_n]              ▷ Input log sequence
4: event_seq ← []
5: for log in logs do
6:     e, (t, _) ← parser(log)                   ▷ (e ∈ E, t ∈ T)
7:     v ← emb_model(tokenizer(t))               ▷ T → R^{1xm}
8:     event_seq.append((e, v))
9: end for
10: graph ← make_graph(event_seq)                ▷ (e ∈ E, R^{1xm}) → G
11: output ← model(graph)                        ▷ G → R^{nxo}
12: return is_anomaly(output)                    ▷ R^{nxo} → {True, False}
```

Table 6
Example of the intermediate outputs of the steps in the Graph-wise methods following Algorithm 4.

Step	Sub-Method (optional)	Output
Input	log_1, log_2, log_3	
For loop	parser(log_1) & emb_model(t_1)	e_1, emb_1
	parser(log_2) & emb_model(t_2)	e_2, emb_2
	parser(log_3) & emb_model(t_3)	e_3, emb_3
Event Seq.	[(e_1, emb_1), (e_2, emb_2), (e_3, emb_3)]	
Make graph	[g_1]	
Model	[$score_1$]	
Is Anomaly	[True]	

- *Edge formulation*: In LogFlash [74], an edge depicts the frequency of occurrence of various logs, defined as $G = (V, E, F_E)$, where $F_E \in \mathbb{R}$ serves as the edge weights. In particular, this approach updates the graph dynamically, enabling adaptation to changing system conditions.
- *Edge-Node formulation*: An embedding representation in the graph can be characterized as $G = (V, E, F_V, F_E)$, with $F_V \in \mathbb{R}^n$ serving as embedding vectors. This formulation is common among the majority of publications in the field:
  - *Use logs as embedding nodes*: In the study by [77], the LogGD approach utilizes a Bert [96] model to encode the templates, thus producing the embeddings for each node. In a similar vein, LogGT [80] was created to facilitate transfer learning between source and target datasets.
  - *Use logs and traces as embedding nodes*: In DeepTraLog [76], the authors construct a graph from the system's logs and traces. They adopted a technique similar to LogRobust [69] to determine the embedding values assigned to each node. Each word in the log uses the pre-trained GloVe model [104], and these are subsequently combined using a weighted sum using TFxIDF.
  - *Use fields in each log*: GLAD [79] builds a varied graph by employing Sentence-BERT [105] to generate embeddings for every node, while using BART [106] to identify fields in each log.

After generating the graphs for various publications, a Graph Neural Network (GNN) framework is utilized: DeepTraLog employs GGNN [107], LogGD GTN [108], GLAD GCN [109], and LogGT HGT [110].

5.2.4. Comparsion Log Anomaly Detection

The LAD approaches can be classified according to the categories listed before. Table 7 provides a brief overview of these categories. Sequential-based methods, like DeepLog, rely solely on the event ID in a sequential arrangement, whereas embedding-based methods, such as LogRobust, utilize the template instead of the event ID, sometimes incorporating the log message, as seen in NeuralLog. Graph-based techniques typically employ a template or message in a non-sequential format, often including an Event ID to distinguish between different nodes within the graph.

Table 7
Comparison between the different categories.

Type	Event ID	Template	Message	Sequential
Sequential-wise	Yes	No	No	Yes
Embedding-wise	No	Yes	Maybe	Yes
Graph-wise	Yes	Yes	Maybe	No

6. Log dataset

Datasets such as HDFS and BGL are frequently regarded as straightforward for detecting log anomalies in academic studies. Research has shown that heuristic techniques can achieve precision comparable to deep learning approaches [111], with F1 scores surpassing 98% in BGL under specific conditions. These datasets were originally designed for general anomaly detection, not specifically for intrusion detection systems (IDS). On the other hand, the ADFA dataset was specifically created for IDS applications [112]. The study points out that while earlier IDS systems could identify intrusions through command frequency, they fall short when dealing with contemporary threats distributed across multiple traces. This indicates that excelling on standard datasets does not ensure practical effectiveness. To keep addressing these challenges, the updated AIT version 2 dataset was developed incorporating various systems and applications [113]. As the development of new software architectures continues, attacks on these systems will evolve as well. Therefore, it is essential to consistently create updated benchmark datasets. Although these datasets can realistically reflect contemporary cybersecurity advancements, they risk becoming outdated over time.

6.1. Log dataset generation framework

We propose a different strategy to improve the design of intrusion detection systems (IDSs) by focusing on direct access and code manipulation to produce accurately anomalies from logs. This approach seeks to improve the evaluation of new methodologies and enrich the understanding of actual data. We emphasize the importance of accurate data interpretation, making connections to fields such as computer vision, where examining datasets helps to minimize biases [114]. The open source framework for generating verification datasets operates according to the following steps:

1. Each execution of the framework collects the logs generated by the method via $log.info$.
2. The templates and logs from the execution processes the data by segmenting the logs into the following features: event ID, log level, time difference, message, and template used. This streamlined structure allows to generate datasets that can be applied directly to various Log Anomaly methods without additional preprocessing.
3. Each dataset embodies a challenge, contains a specific anomaly, and comprises its own training and testing data.
4. The method can employ the training data to adjust the model, though altering hyperparameters is not permitted.
5. The test data function as a unit tests: if a method fails to detect the majority of anomalies, it should be deemed a Fail; otherwise, it is considered a Pass.

An example of this challenge is described in Pseudo-code Algorithm 5, which illustrates a script trying to reach a resource. Under normal circumstances, the script manages to succeed after several attempts, but under irregular conditions, it does not succeed (Line 4). If it fails, the final log is msg_2 (Line 6) else msg_4 (Line 13). Logging methods were designed to allow dynamic adjustments to log messages: adding, removing, or modifying them without altering the code, as described in [69]. These methods emulate consistent software updates with the goal of assessing a technique's robustness against various log versions derived from the training dataset.

At the beginning of a new project, it is typical for the general information and communication systems to be developed alongside the CIDS/IDS as shown in Fig. 7. A particularly concerning in this situations is the lack of data for training models use in the CIDS/IDS. This creates a bottleneck because data collection for training and verification is only possible with nearly finalized software versions. To address this, established datasets from existing literature are used initially, though they may not capture specific features needed for the project, such as certain log attributes [111]. Our framework helps to improve model verification during early evaluations, prior to obtaining definitive project data.

Algorithm 5 Challenge pseudo-code.

```
1:  resource ← Resource()
2:  i ← 0
3:  logs.info(msg_1)
4:  while not resource.init() do
5:      if i >= 10 then
6:          logs.info(msg_2)
7:          return None
8:      end if
9:      logs.info(msg_3)
10:     sleep()
11:     i ← i + 1
12: end while
13: logs.info(msg_4)
14: return resource
```

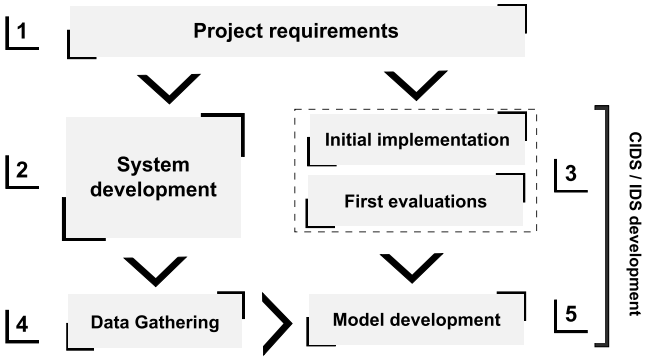


Fig. 7. Example of a project of a information and communication systems with a CIDS/IDS use for based detection. Note that Step 2 and 3 can be done in parallel.

6.2. Design challenges

An anomaly is a data point that deviates noticeably from a given distribution, as mentioned in [115]. We define a challenge as a function $C(N, \text{as\_anomaly})$, which produces N data points classified as normal or abnormal. This challenge will be executed in an unsupervised setting using a model $f(x)$ that returns true if the data point x is deemed anomalous and False otherwise. Pseudo-code Algorithm 6 demonstrates the execution of a challenge C. The process begins with training the model on Line 2, followed by preparing test sets on Lines 3-4 to determine the final F1 score. Each challenge must include an anomaly of one of the following types:

- **Point Anomalies**: occur when a single instance is considered abnormal relative to the rest.
- **Contextual Anomalies**: occur when an instance appears abnormal within a particular context.
- **Collective Anomalies**: occur when a collection of instances is considered anomalous compared to the others.

These anomaly types are based on the work of [115]. As mentioned earlier, a model that does not identify anomalies is considered inadequate in overcoming the challenges. Effective reasoning can demonstrate that a model will fail to meet a specific challenge, thus eliminating the need for empirical evaluation. However, this logic should not be reversed; models frequently use shortcuts in the learning phase [116], potentially leading to outcomes that do not align with our original expectations.

Algorithm 6 Run challenge C pseudo-code.

```
function RUN_CHALLENGE(N: int)→ float
    θ ← train(f, C(N, False))
    normal ← C(N, False)
    abnormal ← C(N, True)
    return get_f1(f_θ(normal), f_θ(abnormal))
end function
```

6.3. Implemented challenges

We pinpoint several issues or risks highlighted in previous publications. These were transformed into challenges and are defined as standard use cases typically encountered in regular programs. Although this straightforward methodology does not cover all possible issues that an Log Anomaly CIDS might face, we assert that it sufficiently demonstrates the framework's effectiveness, as evidenced in the Results section. The challenges are grouped into four distinct categories.

6.3.1. Resource access

A class representing a resource that must be accessed by the code. This class is tailored to simulate sensor behavior, where multiple attempts may be required for successful initialization. Fig. 8 illustrates a diagram depicting resource challenges 1-3. Green arrows are exclusive to normal executions, whereas red arrows appear only in abnormal situations.

- **Challenge 1**: Should the resource remain undiscovered after various attempts, the loop ends without issuing any confirmation message. A model is capable of recognizing this anomaly through two methods: either by noting the sequence length, as normal sequences are shorter, or by identifying the missing final event ID, which signals resource initialization.
- **Challenge 2**: Should the resource be initialized, the code will yield an event ID that confirms it; otherwise, it will provide an event ID indicating an error. A model formulated to identify event IDs not present in the training dataset will be successful.
- **Challenge 3**: Similarly to challenge 2, the sequence IDs for both abnormal and normal events are indistinguishable, since the logs utilize identical templates. Consequently, only models that examine the individual log messages are able to identify anomalies.

6.3.2. Load dependencies

Several dependencies are being loaded sequentially, and the time distribution between these loads varies between normal and abnormal scenarios.

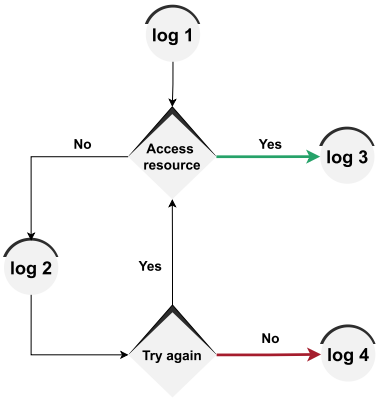


Fig. 8. Diagram of the resource challenges. Note that challenge 1 does not have log 4.

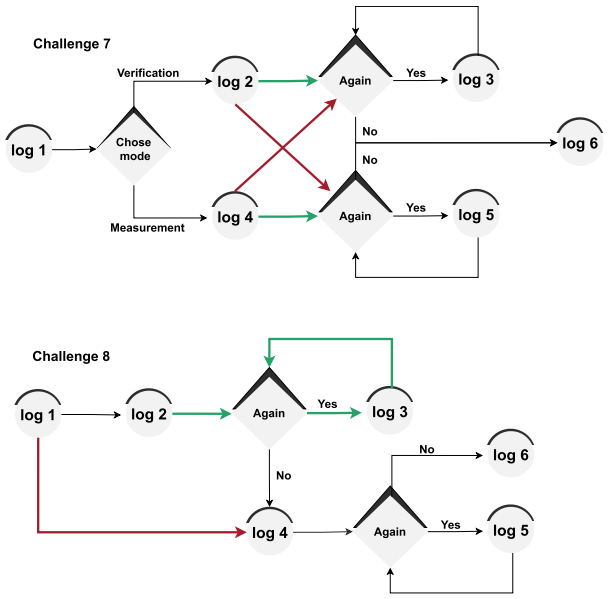


Fig. 9. Diagram of the challenges 7 and 8.

- **Challenge 4**: All dependencies generally require a similar load time, but in abnormal circumstances, one of them may take significantly longer.
- **Challenge 5**: In abnormal cases, the load time of two dependencies is interchanged.
- **Challenge 6**: The same as challenge 4, but the time difference is much smaller.

6.3.3. X-Ray machine

Medical devices often exhibit a well-defined behavior. In this study, our objective is to evaluate the effectiveness of anomaly detection techniques in recognizing these states and identifying operational outliers. Fig. 9 illustrates the appearance of the diagrams for Challenges 7 and 8. Similarly to Fig. 8, the red edges are exclusive to abnormal scenarios, and the green edges denote normal ones.

- **Challenge 7**: The machine operates in either verification or measurement mode, constantly carrying out tasks associated with the chosen mode. Issues arise when the machine mistakenly performs functions of the non-selected mode, which can be identified by log flag statuses and variations in task completion times.
- **Challenge 8**: Initially, the machine is required to execute the verification process before proceeding with the measurements. However, in cases of anomalous behavior, the verification steps are bypassed.

6.3.4. Collaborative setting

CIDS introduce several additional complexities that are not addressed by traditional IDS. For example, a federated learning approach might encounter challenges in data imbalance, which can affect its performance. Numerous benchmarks exist in the literature to assess the efficacy of the model under such conditions [117]. Consequently, we will focus on various challenges unique to CIDS that are not frequently discussed in the literature.

- **Challenge 9**: The system comprises three clients that execute identical code. As each client operates its own parser, the final template of one client's output deviates from the others. Models that either bypass the parser or can adjust the final templates are necessary to distinguish normal activity from anomalies.
- **Challenge 10**: Similar to challenge 9, three clients are executing the same code. However, unlike the others, one client was compromised

prior to the training phase, generating anomalous data within the training set. Only models that can isolate the compromised client will successfully identify these anomalies.

7. Results

In order to demonstrate the practicality of employing the framework, we utilized DeepLog [34] to address several challenges. This approach is commonly seen as a standard in previous log data research. To our knowledge, no official version of the DeepLog code exists. However, multiple implementations are available, and we selected [118] because it has a high number of stars on GitHub. To facilitate a more robust comparison in the discussion of results, we implemented two heuristic approaches inspired by [111]. These approaches illustrate how straightforward techniques can outperform a deep learning model in specific instances. The methods are as detailed below (Algorithms 1, 5 and 6).

- **Length + Event**: Initially, verify whether an event has appeared in the training dataset before; next, ensure that the sequence length fits within the range of those in the training sequences. If both conditions are met, the sequence is deemed nominal; otherwise, it is treated as an anomaly.
- **Time**: When the duration of a sequence falls short of or exceeds the durations present in the training datasets, it is regarded as an anomaly.

7.1. Experiments setup

We trained the different methods in all the challenges and used the default HDFS dataset from the Deeplog repository as a reference. Some hyperparameters were altered from the original DeepLog code:

- **For all cases**: The batch size was adjusted from 2048 down to 100 primarily because the challenge datasets contain merely 100 cases. Moreover, the Deeplog implementation allows the test set to be run using either only the unique event sequences or running through all event sequences. We decide running all event sequences.
- **Only for challenges**: Given the shorter log sequences and fewer unique events relative to HDFS, we set the window size to 3 and limited the model output to 10. In Deeplog's implementation, a sequence is flagged as anomalous if the next log event is not within the model's top n predictions. Thus, following this reasoning, we reduce n from 9 to 2.

The study evaluates models using three different test sets, each consisting of 100 cases, to adequately represent the data distribution. The goal is to achieve an F1 score of at least 0.7, as identifying all cases as anomalies yields an F1 score of 0.67. The model was trained three times per challenge with randomly generated datasets, calculating the average and standard deviation of the results. The primary aim is to determine whether various methods pass the challenges, thus, solely performance metrics are employed. The experiments were repeated with the HDFS dataset on a system with an Intel Core Ultra 7 165U x14 CPU and 16GB of RAM. We divided the results into two sections:

- **Centralized Setup**: we run Challenges 1-8 and HDFS in a centralize set-up.
- **Collaborative Setup**: The primary objective is to evaluate how conventional FedAvg [18] addresses the challenges. Therefore, we concentrate solely on utilizing Deeplog, executing Challenges 9-10 and HDFS.

7.2. Centralized setup

We separately present the results achieved in Deeplog as well as those from the heuristics.

Table 8
Simple approach with the results of the test datasets v1.

Dataset	Precision	Recall	F1	Test
HDFS	0.98 ± 0.00	0.84 ± 0.01	0.90 ± 0.00	PASS
Challenge 1	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	PASS
Challenge 2	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	PASS
Challenge 3	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	FAIL
Challenge 4	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	FAIL
Challenge 5	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	FAIL
Challenge 6	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	FAIL
Challenge 7	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	FAIL
Challenge 8	1.00 ± 0.00	0.38 ± 0.04	0.55 ± 0.04	FAIL

Table 9
Simple time approach with the results of the test datasets v1.

Dataset	Precision	Recall	F1	Test
HDFS	NaN	NaN	NaN	NaN
Challenge 1	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	PASS
Challenge 2	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	FAIL
Challenge 3	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	FAIL
Challenge 4	0.86 ± 0.03	1.00 ± 0.00	0.93 ± 0.02	PASS
Challenge 5	0.39 ± 0.04	0.06 ± 0.02	0.10 ± 0.04	FAIL
Challenge 6	0.94 ± 0.03	0.61 ± 0.01	0.74 ± 0.02	PASS
Challenge 7	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	FAIL
Challenge 8	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	FAIL

Table 10
Simple time approach with the results of the test datasets v2.

Dataset	Precision	Recall	F1	Test
HDFS	NaN	NaN	NaN	NaN
Challenge 1	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	PASS
Challenge 2	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	FAIL
Challenge 3	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	FAIL
Challenge 4	0.88 ± 0.00	1.00 ± 0.00	0.94 ± 0.00	PASS
Challenge 5	0.50 ± 0.10	0.08 ± 0.02	0.14 ± 0.02	FAIL
Challenge 6	0.98 ± 0.01	0.60 ± 0.03	0.74 ± 0.02	PASS
Challenge 7	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	FAIL
Challenge 8	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	FAIL

7.2.1. Results with Deeplog

Table 11 displays the results. Furthermore, we applied the model to test the datasets using a second version of the challenge code, which is shown in Table 12. The remaining methodology remains unchanged. Challenges 4, 5, and 6 consistently have the same sequence of log events, differing only in execution time. As DeepLog does not account for timing variations in procedures, it inherently failed these challenges. Further analysis of these outcomes is provided in the discussion section.

7.2.2. Results with heuristics

The outcomes derived from the Length + Event method are reported in Table 8, whereas the results associated with the Time method are displayed in Tables 9 and 10. Due to the absence of time data in the HDFS dataset utilized for these experiments, the Time heuristics indicate NaN for this field.

7.3. Collaborative setup

We adapted the existing Deeplog configuration to operate in a federated setting utilizing the Flower framework[119]. The weight updating process employed the FedAvg [18] algorithm, which included three clients, three global rounds, and three local epochs per round. The hyperparameters of the centralized version were maintained, with each client having equal-sized training datasets. The outcomes are presented in Tables 11 and 12. We also reevaluated the system's performance using HDFS datasets to verify reliability. Similarly to [26], the federated configuration showed lower performance; however, it should be considered that with enhanced optimization, this performance disparity could be minimized.

Table 11
DeepLog model with the results of the test datasets v1. Fed. HDFS, Challenge 9 and Challenge 10 are part of the Collaborative Setup.

Dataset	Precision	Recall	F1	Test
HDFS	0.96 ± 0.00	0.94 ± 0.01	0.95 ± 0.00	PASS
Challenge 1	0.0 ± 0.00	0.0 ± 0.00	0.00 ± 0.00	FAIL
Challenge 2	0.79 ± 0.01	1.00 ± 0.00	0.88 ± 0.01	PASS
Challenge 3	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	FAIL
Challenge 4	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	FAIL
Challenge 5	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	FAIL
Challenge 6	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	FAIL
Challenge 7	0.87 ± 0.05	0.84 ± 0.08	0.86 ± 0.06	PASS
Challenge 8	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	FAIL
Fed. HDFS	0.82 ± 0.01	0.94 ± 0.01	0.88 ± 0.02	PASS
Challenge 9	0.67 ± 0.00	0.67 ± 0.00	0.67 ± 0.00	FAIL
Challenge 10	0.75 ± 0.00	1.00 ± 0.00	0.86 ± 0.00	PASS

Table 12
DeepLog model with the results of the test datasets v2. Fed. HDFS, Challenge 9 and Challenge 10 are part of the Collaborative Setup.

Dataset	Precision	Recall	F1	Test
HDFS	NaN	NaN	NaN	NaN
Challenge 1	0.50 ± 0.00	1.00 ± 0.00	0.67 ± 0.00	FAIL
Challenge 2	0.50 ± 0.00	1.00 ± 0.00	0.67 ± 0.00	FAIL
Challenge 3	0.50 ± 0.00	1.00 ± 0.00	0.67 ± 0.00	FAIL
Challenge 4	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	FAIL
Challenge 5	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	FAIL
Challenge 6	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	FAIL
Challenge 7	0.50 ± 0.00	1.00 ± 0.00	0.67 ± 0.00	FAIL
Challenge 8	0.50 ± 0.00	1.00 ± 0.00	0.67 ± 0.00	FAIL
Fed. HDFS	NaN	NaN	NaN	NaN
Challenge 9	0.50 ± 0.00	1.00 ± 0.00	0.67 ± 0.00	FAIL
Challenge 10	0.50 ± 0.00	1.00 ± 0.00	0.67 ± 0.00	FAIL

7.4. Result discussion

Based on the results, while Deeplog achieves strong performance with HDFS, it struggles with many challenges. Notably, the small window size of 3 limits the LSTM's ability to consider more than three preceding events, thereby constraining the architecture's capability. Nevertheless, this limitation does not fully account for its difficulties with various challenges. If we maintain the original window size, the LSTM merely needs to predict the final events in the sequence, as the sequences are typically short in these challenges. An approach to addressing this issue with a larger window size is to incorporate padding at the beginning of the sequence. This perspective highlights the importance of using challenges from the beginning of the project, as illustrated in Fig. 7. These discussions contribute to developing more robust models. By identifying early on which challenges may pose difficulties for the model, we can adapt or alleviate the risks in the project's initial phases. We will evaluate the results of each challenge individually.

- *Resource access (Challenges 1-2)*: In Challenge 1, abnormal sequences significantly exceed the nominal ones in length. This is identified by counting the number of events per sequence, as done in the Length + Events approach, or by measuring the total duration of the sequence as in the Time method. However, Deeplog struggles to detect these anomalies. This is primarily because its LSTM-based internal states focus solely on predicting the next event, neglecting the overall sequence structure. It performs better in spotting anomalous events in Challenge 2, but the Length + Events strategy still achieves a higher F1 score.
- *Resource access (Challenge 3)*: A key feature of Challenge 3 is the variability in log entries. In a normal situation, the log concludes with "Errors found None," whereas, in an abnormal situation, it reads, "Errors found ['Exception: something unexpected has happened, please reboot']." Despite this, both scenarios share the template "Errors found <*>" resulting in identical event IDs. Additionally, there are no temporal or spatial variances that could aid in the detection. This challenge emphasizes the necessity of examining the variables within the logs, as sequential methods prove insufficient. None of the considered methods effectively address this challenge.
- *Load Dependencies (Challenges 4-6)*: The effectiveness hinges on evaluating the timing of the sequences, rendering the Time heuristic the only viable option. However, it falls short in Challenge 5 because it does not consider the time needed to perform each event tuple.
- *X-Ray Machine (Challenges 7-8)*: It is essential to correlate events, a task that heuristics cannot perform. DeepLog utilizes an LSTM, which simplifies this task as long as the layer does not exploit a shortcut. By examining the internal architecture of the layer [94], we hypothesize that its behavior in this challenge is influenced by the presence of the forget gate, allowing it to disregard previous inputs. When the model focuses solely on the most recent event ID, it recognizes that if it detects a measurement log, subsequent logs should be of the same type, and vice versa (Fig. 9). This approach succeeds in Challenge 7, but fails in Challenge 8; if a significant portion of the sequence is suddenly skipped, the LSTM will not detect it. The creation of Challenge 8, aimed at causing Deeplog to fail, was develop inspired by the experiments detailed in [116].
- *Collaborative setting (Challenges 9-10)*: The anomalies are quite simple to understand. The major challenges originate in the federated process itself. Deeplog does not succeed in Challenge 9 because templates from different clients are not validated during training. This problem could be addressed by adjusting the training script. Nevertheless, Challenge 10 is successful because FedAvg efficiently removes anomalous instances from the training data. It is crucial to point out that although FedAvg can address Challenge 10, it does not demonstrate its capability to handle more intricate attacks such as [21,22] and [23].

8. Discussion

We structured our discoveries around three primary research questions, which were addressed in various sections of this publication. In this section, we will offer research answers (RA) to these research questions based on earlier observations.

RQ1: What methods and baselines are used in the literature for Anomaly CIDS?

RA1: In the literature discussion section, we demonstrate that not all Anomaly CIDS primarily rely on logs to identify system abnormalities. By examining the baseline datasets used for the evaluation (as shown in Table 1), we can identify the type of input domain involved. Interestingly, using Log Anomaly CIDS is relatively uncommon, prompting us to also consider similar LAD techniques to enhance sample distribution. Based on our research, we determine that the most commonly used baseline datasets for Log Anomaly CIDS and comparable methods are HDFS and BGL.

When analyzing the methods used, it is essential to divide them into two categories. Firstly, under the collaborative framework, as detailed in the section Anomaly CIDS, we explore different strategies for node division: Centralized, Decentralized, and Distributed CIDS. In addition, the literature indicates that the most common method of updating the local models of the nodes within CIDS is federated learning, with FedAvg being the most widely used technique, although other methods are also identified. In the subsequent category, which concentrates on the algorithm specific to Log Anomaly Detection, various architectures are shown in Fig. 5. However, recent developments are more effectively depicted in Fig. 1, which emphasizes the increasing adoption of LLM and graph-based methods in contemporary Log Anomaly Detection techniques.

RQ2: How can we categorize the different Log Anomaly CIDS and other Log Anomaly approaches?

RA2: Earlier works, as noted in the Related Work section, categorizes approaches based on the overarching architecture employed, such as CNN, RNN, or Transformers. Although this classification may be informative, we contend that it is not the most insightful point of view. In our study, we shift the focus by categorizing based on the manner in which data is processed and the objective function utilized to optimize the models. By examining these characteristics, we have established three principal categories:

- *Sequential-wise*: Each log sequence undergoes a transformation into an event ID, and the model is optimized by attempting to forecast the subsequent or masked event in a sequence. This method is relatively straightforward and requires less computational power. Nonetheless, merely using event IDs leads to a significant loss of log message information. Additionally, there is substantial reliance on how logs are parsed, making this method susceptible to alterations in logs following future code updates.
- *Embedding-wise*: An embedding encoder is employed to analyze the logs, as opposed to relying solely on the event ID numbers. These techniques were developed to address the limitations associated with Sequential-wise methods. However, the majority of methods in this category are supervised, which poses challenges for their application in zero-day attacks.
- *Graph-wise*: Function similarly to the Embedding-wise approach, but instead of arranging the logs sequentially by chronological order, they are connected as a graph. This technique often involves greater complexity and necessitates additional steps compared to other methods.

RQ3: To what extent can we improve the reliability of Log Anomaly CIDS?

RA3: Creating Log Anomaly CIDS necessitates considering several elements, such as strategies for anomaly detection and enhancements to local models. Although federated learning is used to prevent the sharing of sensitive data between nodes, it generally achieves lower results than centralized training. Often, initial CIDS designs do not include specific training and testing datasets, leading to a dependence on benchmark datasets that might not ensure optimal deployment performance. This paper introduces an open framework for early evaluation, allowing the thorough testing of models against various challenges to identify vulnerabilities and improve robustness. The framework effectively identifies the shortcomings in current methods, such as DeepLog. Although this framework proposal and application do not directly provide an answer to RQ3, we believe it represents progress towards developing more reliable Log Anomaly CIDS.

8.1. Threats to validity

We conducted an empirical assessment of the proposed framework, concentrating on Deeplog due to its prominence as a standard benchmark in the field. Since the framework is offered as a proof of concept, only a limited evaluation was performed. We recognize that the absence of additional models and the limited number of challenges pose a threat to validity. A more comprehensive empirical analysis will be carried out in future research.

9. Conclusion

This research seeks to provide a comprehensive summary of Log Anomaly CIDS, emphasizing its main characteristics, prevailing trends, and obstacles. Furthermore, we have created a novel open-source framework to aid the progress of these systems. Our findings are supported by numerous recent studies on this subject, demonstrating that only a small number of Anomaly CIDS rely on logs for anomaly detection. We examine and classify multiple LAD techniques and offer tools to enhance the robustness of these systems. We hope that our work will aid the research and development of upcoming Log Anomaly CIDS.

In future work, our main aim is to further refine the distinct challenges of the framework. We plan to achieve this by developing a more precise methodology to identify a broader spectrum of anomalies and risks that Log Anomaly CIDS might face. In addition, we intend to introduce challenges on the same subject with varying levels of difficulty to enhance model comparison. Finally, we will incorporate multiple baseline datasets to establish a unified testing framework. Additionally, we aim to investigate alternative metrics for CIDS, such as memory or energy usage, and assess any potential risks these methods might face in real world scenarios.

Declaration of generative AI and AI-assisted technologies in the writing process

During the preparation of this work the author(s) used the Writefull Overleaf extension [120] to improve readability. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the published article.

CRediT authorship contribution statement

André García Gómez: Writing – review & editing, Writing – review & editing, Methodology, Investigation, Formal analysis, Data curation, Conceptualization; **Max Landauer:** Writing – review & editing, Supervision; **Markus Wurzenberger:** Writing – review & editing, Supervision; **Florian Skopik:** Writing – review & editing, Supervision; **Edgar Weippl:** Writing – review & editing, Supervision.

Data availability

The code and data are available on Github. A link is provided in the document.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Andre Garcia Gomez reports financial support was provided by European Union. Andre Garcia Gomez reports financial support was provided by European Defence Fund. Andre Garcia Gomez reports financial support was provided by Austrian Research Promotion Agency. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

Funded by the European Union under the European Defence Fund (GA No. 101121403 - NEWSROOM and GA No. 101121414 - LATACC). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them. This work is co-funded by the Austrian FFG Kiras project ASOC (GA no. FO999905301).

References

[1] CrowdStrike, Global threat report, 2025, (https://www.crowdstrike.com/en-us/global-threat-report/).
[2] J.O. Kephart, D.M. Chess, The vision of autonomic computing, Computer 36 (1) (2003) 41–50.
[3] Y.S. Wu, et al., Collaborative intrusion detection system (CIDS): a framework for accurate and efficient IDS, in: The 19th Annual Computer Security Applications Conference, IEEE, 2003, pp. 234–244.
[4] A. Makanju, A.N. Zincir-Heywood, E.E. Milios, A lightweight algorithm for message type extraction in system application logs, Trans. Knowl. Data Eng. 24 (11) (2011) 1921–1936.

[5] Q. Lin, H. Zhang, J.G. Lou, Y. Zhang, X. Chen, Log clustering based problem identification for online service systems, in: Proceedings of the 38th International Conference on Software Engineering Companion, 2016, pp. 102–111.

[6] M.A. Inam, Y. Chen, A. Goyal, et al., Sok: history is a vast early warning system: Auditing the provenance of system intrusions, in: Symposium on Security and Privacy, IEEE, 2023, pp. 2620–2638.

[7] B. Debnath, M. Solaimani, et al., LogLens: a real-time log analysis system, in: 38th International Conference on Distributed Computing Systems, IEEE, 2018, pp. 1052–1062.

[8] A.A. Wardana, P. Sukarno, Taxonomy and survey of collaborative intrusion detection system using federated learning, Comput. Surv. (2024) 88.

[9] M. Landauer, S. Onder, F. Skopik, M. Wurzenberger, Deep learning for anomaly detection in log data: a survey, Mach. Learn. Appl. 12 (2023) 100470.

[10] A. Garcia, log-gym, 2025, (https://github.com/ait-aecid/log-gym).

[11] E. Vasilomanolakis, S. Karuppayah, M. Mühlhäuser, M. Fischer, Taxonomy and survey of collaborative intrusion detection, Comput. Surv. 47 (4) (2015) 1–33.

[12] H.J. Liao, C.H.R. Lin, Y.C. Lin, K.Y. Tung, Intrusion detection system: a comprehensive review, J. Netw. Comput. Appl. 36 (1) (2013) 16–24.

[13] A. Khraisat, I. Gondal, P. Vamplew, J. Kamruzzaman, Survey of intrusion detection systems: techniques, datasets and challenges, Cybersecurity 2 (1) (2019) 1–22.

[14] M. Landauer, F. Skopik, M. Wurzenberger, A. Rauber, System log clustering approaches for cyber security applications: a survey, Comput. Secur. 92 (2020) 101739.

[15] M. Landauer, M. Wurzenberger, F. Skopik, W. Hotwagner, G. Höld, Aminer: a modular log data analysis pipeline for anomaly-based intrusion detection, Digit. Threats 4 (1) (2023) 1–16.

[16] M. Landauer, F. Skopik, M. Wurzenberger, A. Rauber, Dealing with security alert flooding: using machine learning for domain-independent alert aggregation, Trans. Privacy Secur. 25 (3) (2022) 1–36.

[17] N.A. Angel, et al., Recent advances in evolving computing paradigms: Cloud, edge, and fog technologies, Sensors 22 (1) (2021) 196.

[18] H.B. McMahan, et al., Federated learning of deep networks using model averaging, 2 (2) (2016). arXiv preprint arXiv:1602.05629

[19] A. Kundu, P. Yu, L. Wynter, S.H. Lim, Robustness and personalization in federated learning: a unified approach via regularization, in: Int. Conf. Edge Comput. Commun., IEEE, 2022, pp. 1–11.

[20] R. Xu, et al., Dual defense: enhancing privacy and mitigating poisoning attacks in federated learning, in: The Thirty-eighth Annual Conference on Neural Information Processing Systems, 2024

[21] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, V. Shmatikov, How to backdoor federated learning, in: International Conference on Artificial Intelligence and Statistics, PMLR, 2020, pp. 2938–2948.

[22] C. Xie, O. Koyejo, I. Gupta, Fall of empires: breaking byzantine-tolerant sgd by inner product manipulation, in: Uncertainty in Artificial Intelligence, PMLR, 2020, pp. 261–270.

[23] G. Baruch, M. Baruch, Y. Goldberg, A little is enough: circumventing defenses for distributed learning, Adv. Neural Inf. Process. Syst. 32 (2019) 8635–8645.

[24] C. Zhang, Y. Xie, H. Bai, et al., A survey on federated learning, Knowl. Based Syst. 216 (2021) 106775.

[25] X. Li, K. Huang, W. Yang, et al., On the convergence of fedavg on non-iid data, (2019). arXiv preprint arXiv:1907.02189

[26] S.A. Rahman, H. Tout, et al., Internet of things intrusion detection: centralized, on-device, or federated learning?, Network 34 (6) (2020) 310–317.

[27] E.M. Campos, P.F. Saura, et al., Evaluating Federated Learning for intrusion detection in Internet of Things: review and challenges, Comput. Netw. 203 (2022) 108661.

[28] A. Li, J. Sun, B. Wang, et al., Lotteryfl: personalized and communication-efficient federated learning with lottery ticket hypothesis on non-iid datasets, (2020). arXiv preprint arXiv:2008.03371

[29] J. Frankle, et al., The lottery ticket hypothesis: finding sparse, trainable neural networks, (2018). arXiv preprint arXiv:1803.03635

[30] B. Li, S. Ma, R. Deng, et al., Federated anomaly detection on system logs for the internet of things: a customizable and communication-efficient approach, Trans. Netw. Serv. Manage. 19 (2) (2022) 1705–1716.

[31] Y. Liu, et al., Deep anomaly detection for time-series data in industrial IoT: a communication-efficient on-device federated learning approach, Internet Things J. 8 (8) (2020) 6348–6358.

[32] M. Abdel-Basset, N. Moustafa, et al., Federated intrusion detection in blockchain-based smart transportation systems, Trans. Intell. Transp. Syst. 23 (3) (2021) 2523–2537.

[33] P. Himler, et al., Anomaly detection in log-event sequences: a federated deep learning approach and open challenges, ML Appl. 16 (2024) 100554.

[34] M. Du, F. Li, G. Zheng, V. Srikumar, Deeplog: anomaly detection and diagnosis from system logs through deep learning, in: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 1285–1298.

[35] M. Schonlau, W. DuMouchel, et al., Computer intrusion: detecting masquerades, Stat. Sci. 16 (2001) 58–74.

[36] R. Zhao, Y. Yin, Y. Shi, Z. Xue, Intelligent intrusion detection based on federated learning aided long short-term memory, Phys. Commun. 42 (2020) 101157.

[37] A.E.W. Johnson, T.J. Pollard, et al., MIMIC-III, a freely accessible critical care database, Sci. Data 3 (1) (2016) 1–9.

[38] W. Schneble, G. Thamilarasu, Attack detection using federated learning in medical cyber-physical systems, in: Proceedings of the 28th International Conference on Computer Communication and Networks, 29, 2019, pp. 1–8.

[39] T.D. Nguyen, S. Marchal, et al., DÏoT: a federated self-learning anomaly detection system for IoT, in: 39th International Conference on Distributed Computing Systems, IEEE, 2019, pp. 756–767.

[40] W. Xu, et al., Mining console logs for large-scale system problem detection, SysML 8 (2008) 4.

[41] A. Oliner, J. Stearley, What supercomputers say: a study of five system logs, in: 37th IFIP International Conference on Dependable Systems and Networks (DSN'07), IEEE, 2007, pp. 575–584.

[42] S. He, et al., Loghub: a large collection of system log datasets towards automated log analytics, (2020) arXiv–2008. arXiv e-prints

[43] C. Amza, A. Chanda, A.L. Cox, S. Elnikety, et al., Specification and implementation of dynamic web site benchmarks, in: International Workshop on Workload Characterization, IEEE, 2002, pp. 3–13.

[44] J. Zhu, et al., Loghub: a large collection of system log datasets for ai-driven log analytics, in: 34th International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2023, pp. 355–366.

[45] M. Landauer, F. Skopik, M. Wurzenberger, W. Hotwagner, A. Rauber, Have it your way: generating customized log datasets with a model-driven simulation testbed, Trans. Reliab. 70 (1) (2020) 402–415.

[46] S. Revathi, A. Malathi, A detailed analysis on NSL-KDD dataset using various machine learning techniques for intrusion detection, Int. J. Eng. Res. Technol. 2 (12) (2013) 1848–1853.

[47] N. Moustafa, The Bot-IoT dataset, 2019, https://doi.org/10.21227/r7v2-x988

[48] G. Meena, R.R. Choudhary, A review paper on IDS classification using KDD 99 and NSL KDD dataset in WEKA, in: 2017 International Conference on Computer, Communications and Electronics (Comptelix), 2017, pp. 553–558. https://doi.org/10.1109/COMPTELIX.2017.8004032

[49] i. labs, itrust labs datasets, (https://itrust.sutd.edu.sg/itrust-labs_datasets/dataset_info/). Accessed: 2024-11-26.

[50] N. Moustafa, J. Slay, The evaluation of network anomaly detection systems: statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set, Inform. Secur. J. 25 (1-3) (2016) 18–31.

[51] A. Alsaedi, N. Moustafa, et al., TON_IoT telemetry dataset: a new generation dataset of IoT and IIoT for data-driven intrusion detection systems, IEEE Access 8 (2020) 165130–165150.

[52] H.M. Song, J. Woo, H.K. Kim, In-vehicle network intrusion detection using deep convolutional neural network, Veh. Commun. 21 (2020) 100198.

[53] E. Keogh, J. Lin, A. Fu, Hot sax: efficiently finding the most unusual time series subsequence, in: Fifth International Conference on Data Mining, IEE, 2005, pp. 8–pp.

[54] T. Morris, W. Gao, Industrial control system traffic data sets for intrusion detection research, in: Critical Infrastructure Protection VIII: 8th IFIP WG 11.10 International Conference, ICCIP, Arlington, VA, USA, March 17-19, 2014, Revised Selected Papers 8, Springer, 2014, pp. 65–78.

[55] NYC, Taxi and Limousine comision dataset, (https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page). Accessed: 2024-11-27.

[56] D. Preuveneers, V. Rimmer, I. Tsingenopoulos, J. Spooren, W. Joosen, E. Ilie-Zudor, Chained anomaly detection models for federated learning: An intrusion detection case study, Appl. Sci. 8 (12) (2018) 2663.

[57] X. Zhou, Y. Hu, W. Liang, J. Ma, Q. Jin, Variational LSTM enhanced anomaly detection for industrial big data, Trans. Ind. Inform. 17 (5) (2020) 3469–3477.

[58] B. Li, Y. Wu, et al., DeepFed: federated deep learning for intrusion detection in industrial cyber–physical systems, Trans. Ind. Inform. 17 (8) (2020) 5615–5624.

[59] P.H. Mirzaee, et al., Fids: a federated intrusion detection system for 5g smart metering network, in: International Conference on Mobility, Sensing and Networking, IEEE, 2021, pp. 215–222.

[60] T.T. Huong, T.P. Bac, et al., Detecting cyberattacks using anomaly detection in industrial control systems: a federated learning approach, Comput. Ind. 132 (2021) 103509.

[61] X. Zhou, et al., Siamese neural network based few-shot learning for anomaly detection in industrial cyber-physical systems, Trans. Ind. Inform. 17 (8) (2020) 5790–5798.

[62] Y. Cheng, et al., Leveraging semisupervised hierarchical stacking temporal convolutional network for anomaly detection in IoT communication, Internet Things J. 8 (1) (2020) 144–155.

[63] P.F. de Araujo-Filho, et al., Intrusion detection for cyber–physical systems using generative adversarial networks in fog environment, Internet Things J. 8 (8) (2020) 6247–6256.

[64] J. Shu, L. Zhou, W. Zhang, et al., Collaborative intrusion detection for VANETs: a deep learning-based distributed SDN approach, Trans. Intell. Transp. Syst. 22 (7) (2020) 4519–4530.

[65] T. Thu Huong, et al., LocKedge: low-complexity cyberattack detection in IoT edge computing, (2020) arXiv–2011. arXiv e-prints

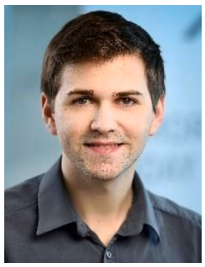
[66] O. Friha, M.A. Ferrag, et al., FELIDS: federated learning-based intrusion detection system for agricultural Internet of Things, J. Parallel Distrib. Comput. 165 (2022) 17–31.

[67] S. Lu, et al., Detecting anomaly in big data system logs using convolutional neural network, in: 16th International Conference on Dependable, Autonomic and Secure Computing, 16th International Conference on Pervasive Intelligence and Computing, 4th International Conference on Big Data Intelligence and Computing and Cyber Science and Technology Congress, IEEE, 2018, pp. 151–158.

[68] W. Meng, Y. Liu, Y. Zhu, et al., Loganomaly: unsupervised detection of sequential and quantitative anomalies in unstructured logs, in: IJCAI, 19, 2019, pp. 4739–4745.

[69] X. Zhang, et al., Robust log-based anomaly detection on unstable log data, in: Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2019, pp. 807–817.
[70] R. Chen, et al., Logtransfer: cross-system log anomaly detection for software systems with transfer learning, in: 31st International Symposium on Software Reliability Engineering, IEEE, 2020, pp. 37–47.
[71] Z. Wang, Z. Chen, J. Ni, et al., Multi-scale one-class recurrent neural networks for discrete event sequence anomaly detection, in: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, 2021, pp. 3726–3734.
[72] H. Guo, et al., Logbert: log anomaly detection via bert, in: International Joint Conference on Neural Networks, IEEE, 2021, pp. 1–8.
[73] V.H. Le, H. Zhang, Log-based anomaly detection without log parsing, in: International Conference on Automated Software Engineering (ASE), IEEE, 2021, pp. 492–504.
[74] T. Jia, Y. Wu, C. Hou, Y. Li, Logflash: real-time streaming anomaly detection and diagnosis from system logs for large-scale software systems, in: 32nd International Symposium on Software Reliability Engineering, IEEE, 2021, pp. 80–90.
[75] L. Yang, J. Chen, et al., Semi-supervised log-based anomaly detection via probabilistic label estimation, in: 43rd International Conference on Software Engineering (ICSE), IEEE, 2021, pp. 1448–1460.
[76] C. Zhang, et al., Deeptralog: trace-log combined microservice anomaly detection through graph-based deep learning, in: International Conference on Software Engineering, 2022, pp. 623–634.
[77] Y. Xie, et al., Loggd: detecting anomalies from system logs with graph neural networks, in: 22nd International conference on software quality, reliability and security, IEEE, 2022, pp. 299–310.
[78] H. Guo, X. Lin, et al., Translog: a unified transformer-based framework for log anomaly detection, (2021). arXiv preprint arXiv:2201.00016
[79] Y. Li, Y. Liu, H. Wang, Z. Chen, W. Cheng, Y. Chen, W. Yu, H. Chen, C. Liu, Glad: content-aware dynamic graphs for log anomaly detection, in: 2023 IEEE International Conference on Knowledge Graph (ICKG), IEEE, 2023, pp. 9–18.
[80] P. Wang, X. Zhang, Z. Cao, W. Xu, W. Li, LogGT: cross-system log anomaly detection via heterogeneous graph feature and transfer learning, Expert Systems with Applications 251 (2024) 124082.
[81] S. Liu, L. Deng, H. Xu, W. Wang, LogBD: a log anomaly detection method based on pretrained models and domain adaptation, Appl. Sci. 13 (13) (2023) 7739.
[82] Y. Xie, H. Zhang, M.A. Babar, LogSD: detecting anomalies from system logs through self-supervised learning and frequency-based masking, Softw. Eng. 1 (FSE) (2024) 2098–2120.
[83] V.H. Le, H. Zhang, PreLog: a pre-trained model for log analytics, Manage. Data 2 (3) (2024) 1–28.
[84] Y. Liu, et al., Interpretable online log analysis using large language models with prompt strategies, in: Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension, 2024, pp. 35–46.
[85] Y. Ji, Y. Liu, F. Yao, M. He, S. Tao, X. Zhao, S. Chang, X. Yang, W. Meng, Y. Xie, et al., Adapting large language models to log analysis with interpretable domain knowledge, (2024). arXiv preprint arXiv:2412.01377
[86] X. Han, S. Yuan, M. Trabelsi, Loggpt: log anomaly detection via gpt, in: 2023 IEEE International Conference on Big Data (BigData), IEEE, 2023, pp. 1117–1122.
[87] Z. Yang, I.G. Harris, LogLLaMA: transformer-based log anomaly detection with LLaMA, (2025). arXiv preprint arXiv:2503.14849
[88] X. Han, S. Yuan, Unsupervised cross-system log anomaly detection via domain adaptation, in: Proceedings of the 30th ACM International Conference on Information & Knowledge Management, 2021, pp. 3068–3072.
[89] C. Zhang, T. Jia, G. Shen, P. Zhu, Y. Li, Metalog: generalizable cross-system anomaly detection from logs with meta-learning, in: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, 2024, pp. 1–12.
[90] P. He, J. Zhu, Z. Zheng, M.R. Lyu, Drain: an online log parsing approach with fixed depth tree, in: International Conference on Web Services, IEEE, 2017, pp. 33–40.
[91] M. Du, et al., Spell: streaming parsing of system event logs, in: 16th International Conference on Data Mining, IEEE, 2016, pp. 859–864.
[92] Y. Xiao, V.-H. Le, H. Zhang, Stronger, Faster, and Cheaper Log Parsing with LLMs, (2024). arXiv preprint arXiv:2406.06156
[93] S. Forrest, S.A. Hofmeyr, A. Somayaji, T.A. Longstaff, A sense of self for unix processes, in: Proceedings 1996 IEEE Symposium on Security and Privacy, IEEE, 1996, pp. 120–128.
[94] S. Hochreiter, Long short-term memory, Neural Comput. 9 (1997) 1735–1780.
[95] A. Vaswani, Attention is all you need, Advances in Neural Information Processing Systems 30 (2017) 5998–6008.
[96] J. Devlin, Bert: pre-training of deep bidirectional transformers for language understanding, (2018). preprint arXiv:1810.04805
[97] L. Ruff, R. Vandermeulen, N. Goernitz, et al., Deep one-class classification, in: International Conference on Machine Learning, PMLR, 2018, pp. 4393–4402.
[98] J. Chung, C. Gulcehre, K. Cho, Y. Bengio, Empirical evaluation of gated recurrent neural networks on sequence modeling, (2014). arXiv preprint arXiv:1412.3555
[99] A. Joulin, Fasttext. zip: compressing text classification models, (2016). arXiv preprint arXiv:1612.03651
[100] Y. LeCun, B. Boser, et al., Handwritten digit recognition with a back-propagation network, Adv. Neural Inform. Process. Syst. 2 (1989) 396–404.
[101] Y. Ganin, E. Ustinova, et al., Domain-adversarial training of neural networks, J. Mach. Learn. Res. 17 (59) (2016) 1–35.
[102] L. Wu, Y. Chen, K. Shen, et al., Graph neural networks for natural language processing: a survey, Found. Trends® Mach. Learn. 16 (2) (2023) 119–328.
[103] X. Ma, J. Wu, S. Xue, et al., A comprehensive survey on graph anomaly detection with deep learning, Trans. Knowl. Data Eng. 35 (12) (2021) 12012–12038.
[104] J. Pennington, R. Socher, C.D. Manning, Glove: global vectors for word representation, in: Conference on Empirical Methods in Natural Language Processing, 2014, pp. 1532–1543.
[105] N. Reimers, Sentence-BERT: sentence embeddings using siamese BERT-networks, (2019). arXiv preprint arXiv:1908.10084
[106] M. Lewis, Bart: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension, (2019). arXiv preprint arXiv:1910.13461
[107] Y. Li, D. Tarlow, M. Brockschmidt, R. Zemel, Gated graph sequence neural networks, (2015). arXiv preprint arXiv:1511.05493
[108] Y. Shi, Z. Huang, et al., Masked label prediction: unified message passing model for semi-supervised classification, (2020). arXiv preprint arXiv:2009.03509
[109] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, (2016). arXiv preprint arXiv:1609.02907
[110] Z. Hu, Y. Dong, K. Wang, Y. Sun, Heterogeneous graph transformer, in: Web conference, 2020, pp. 2704–2710.
[111] M. Landauer, F. Skopik, M. Wurzenberger, A critical review of common log data sets used for evaluation of sequence-based anomaly detection techniques, Softw. Eng. 1 (FSE) (2024) 1354–1375.
[112] G. Creech, J. Hu, Generation of a new IDS test dataset: Time to retire the KDD collection, in: Wireless Communications and Networking Conference (WCNC), IEEE, 2013, pp. 4487–4492.
[113] M. Landauer, F. Skopik, et al., Maintainable log datasets for evaluation of intrusion detection systems, Trans. Depend. Secure Comput. 20 (4) (2022) 3466–3482.
[114] R. Chakraborty, Y. Wang, J. Gao, R. Zheng, C. Zhang, F. De la Torre, Visual data diagnosis and debiasing with concept graphs, (2024). arXiv preprint arXiv:2409.18055
[115] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: a survey, Comput. Surv. 41 (3) (2009) 1–58.
[116] R. Geirhos, J.H. Jacobsen, et al., Shortcut learning in deep neural networks, Nat. Mach. Intell. 2 (11) (2020) 665–673.
[117] D. Zeng, S. Liang, X. Hu, et al., Fedlab: a flexible federated learning framework, (2021). arXiv preprint arXiv:2107.11621
[118] wuyifan18, DeepLog, 2023, (https://github.com/wuyifan18/DeepLog/tree/master).
[119] D.J. Beutel, et al., Flower: a friendly federated learning research framework, (2020). arXiv preprint arXiv:2007.14390
[120] WriteFull, WriteFull, (https://www.writefull.com/).

André García Gómez joined the Austrian Institute of Technology in 2024 and is currently employed as a PhD candidate in the Cyber Security Research Group. His main research interests are anomaly detection, machine learning, log data analysis, and data science. André received his master's degree in Artificial Intelligence in 2021 from the Università della Svizzera italiana.



Dr. Max Landauer joined the Austrian Institute of Technology in 2017 and is currently employed as a Senior Scientist in the Cyber Security Research Group. His main research interests are anomaly detection, cyber threat intelligence, log data analysis, and cyber security testbeds. Max obtained his master's degree in Computer Science in 2018 and finished his PhD studies in 2022 at the Vienna University of Technology.

Dr. Markus Wurzenberger is a Senior Scientist and Project Manager at the Austrian Institute of Technology. Since 2014 he is part of the Cyber Security Research Group of AIT's Center for Digital Safety and Security. His main research interests are log data analysis with focus on anomaly detection and cyber threat intelligence (CTI). Markus obtained a PhD in computer science in 2021. In 2015 Markus obtained his master's degree in Technical Mathematics at the Vienna University of Technology.

Dr. Dr. Florian Skopik is Head of the Cyber Security Research Program at the Austrian Institute of Technology (AIT) with a team comprising around 30 people. He spent 10+ years in cyber security research, before, and partly in parallel, another 15 years in software development. Nowadays, he coordinates national and large-scale international research projects, as well as the overall research direction of the team. His main interests are centered on critical infrastructure protection, smart grid security, and national cyber security and defense.

Edgar Weippl is research director of SBA Research and full professor at the University of Vienna. Edgar's research focuses on blockchain and distributed ledger technologies and security of production systems engineering. Edgar is member of the editorial board of Computers & Security and associate editor of IEE Transactions on Information Forensics and Security. He is Austria's representative at IFIP TC 11: Security and Privacy Protection in Information Processing Systems. He was General Chair of SACMAT 2015, PC Chair of Esorics 2015, General Chair of ACM CCS 2016, PC Chair of ACM SACMAT 2017, General Chair Euro S&P 2021 and 2024.