Deep Learning for Anomaly Detection in Log Data: A Survey

Max Landauer, Sebastian Onder, Florian Skopik, and Markus Wurzenberger

***Abstract*—Automatic log file analysis enables early detection of relevant incidents such as system failures. In particular, self-learning anomaly detection techniques capture patterns in log data and subsequently report unexpected log event occurrences to system operators without the need to provide or manually model anomalous scenarios in advance. Recently, an increasing number of approaches leveraging deep learning neural networks for this purpose have been presented. These approaches have demonstrated superior detection performance in comparison to conventional machine learning techniques and simultaneously resolve issues with unstable data formats. However, there exist many different architectures for deep learning and it is non-trivial to encode raw and unstructured log data to be analyzed by neural networks. We therefore carry out a systematic literature review that provides an overview of deployed models, data pre-processing mechanisms, anomaly detection techniques, and evaluations. The survey does not quantitatively compare existing approaches but instead aims to help readers understand relevant aspects of different model architectures and emphasizes open issues for future work.**

***Index Terms*—log data, anomaly detection, neural networks, deep learning**

I. Introduction

LOG files provide a rich source of information when it comes to monitoring computer systems. Thereby, the majority of log events are usually generated as consequences of normal system operations, such as starting and stopping of processes, restarting of virtual machines, users accessing resources, etc. However, applications also produce logs when faulty or otherwise undesired system states occur, for example, failed processes, availability issues, or security incidents. These traces of unexpected and possibly unsafe system activities are important for system operators that timely need to act upon them to prevent or diminish system damage and avoid adverse cascading effects.

The main problem for this kind of log file analysis is that it is non-trivial to identify these relevant log events within the much larger number of less interesting traces of standard system usage. In particular, the sheer amount of logs produced by modern applications renders manual analysis infeasible and necessitates automatic mechanisms [1]. Unfortunately, manually coded signatures and rules that search for specific keywords in logs only have limited applicability and are not suitable for scenarios that are not known beforehand [2]. It is therefore necessary to deploy anomaly detection techniques that automatically learn models representing the normal baseline of system behavior and subsequently disclose any deviations from these models as possibly adverse activities that require attention by human operators.

Machine learning provides many viable techniques for the purpose of anomaly detection in log files and many different approaches have been proposed in the past, including clustering [3] and workflow mining [4], statistical analysis of event parameters [5], time-series analysis to recognize changes of event frequencies [6], and many more [2], [7]. Recently, researchers started using deep neural networks for log-based anomaly detection in an attempt to repeat the successes of deep learning from image and speech recognition that outperform conventional machine learning methods [8]. However, as system log events are generally unstructured and involve intricate dependencies, it is non-trivial to prepare the data in a way to enable ingestion by neural networks and extract features that are relevant for detection. Moreover, the wide variety of existing deep learning architectures such as recursive or convolutional neural networks makes it difficult to select an appropriate model for a specific use-case at hand and understand their respective requirements on the format and properties of the input data.

To the best of our knowledge there is currently only a limited overview of the state-of-the-art of log-based anomaly detection with deep learning [9]–[12]. As a consequence it is difficult to understand what features are suitable to be extracted from raw log data, how these features could be transformed into a format that is adequate to be ingested by neural networks, and which model architectures are appropriate for detecting specific patterns in logs. Existing surveys only compare few anomaly detection approaches and focus mainly on sequential patterns in log data [9]–[11] or focus on network traffic rather than system log data [12].

We therefore carry out a systematic literature review on deep learning for anomaly detection in log data. Our main focus is thereby to survey scientific publications on the deployed model architectures, their respective requirements and transformations for handling unstructured input log data, the methods used to differentiate between normal and anomalous data samples, and the presented evaluations. The goals of this review include helping readers understand the main challenges in this research field, creating a work of reference for practical applications, and proposing ideas for future work. We point out that this survey does not quantitatively compare detection performances of the reviewed approaches as only few open-

M. Landauer, S. Onder, F. Skopik, and M. Wurzenberger are with the Center for Digital Safety and Security, AIT Austrian Institute of Technology, Vienna, Austria.
E-mail: firstname.lastname@ait.ac.at

source implementations are available and comparisons of these approaches are already presented in other surveys [9], [10]. Overall, this survey analyses the following research questions:

RQ1: What are the main challenges of log-based anomaly detection with deep learning?
RQ2: What state-of-the-art deep learning algorithms are typically applied?
RQ3: How is log data pre-processed to be ingested by deep learning models?
RQ4: What types of anomalies are detected and how are they identified as such?
RQ5: How are the proposed models evaluated?
RQ6: To what extent do the approaches rely on labeled data and support incremental learning?
RQ7: To what extent are the presented results reproducible in terms of availability of source code and used data?

The remainder of this paper is structured as follows. Section II first explains the terms deep learning, log data, and anomaly detection, and then provides an overview of common challenges. We explain our methodology for selecting relevant publications and carrying out the survey in Sect. III. Section IV presents all results of our survey in detail. We discuss these results and answer our research questions in Sect. V. Finally, Sect. VI concludes this paper.

II. Background

In this section we first clarify some general concepts and terms relevant for anomaly detection in log data based on deep learning. We then outline scientific challenges that are specific to that research field.

A. Preliminary Definitions

The study carried out in this paper hinges on an understanding of three main concepts: deep learning, log data, and anomaly detection. However, the exact characteristics and consequential requirements of the respective fields may be used differently across research areas and existing literature. In the following, we therefore describe the basic properties of these three concepts.

1) Deep Learning: Artificial neural networks (ANN) have been developed in an attempt to recreate biological information processing systems in the form of connected communication nodes. For this purpose, varying numbers of nodes are arranged in sequences of layers, in particular, an input layer that reads in the data, several hidden layers connecting neighboring layers with weighted edges, and an output layer. Nodes activate when receiving specific signals on their connected edges, which in turn generates the input for subsequent layers. The main idea is that such networks are capable of recognizing non-linear structures in the input data and subsequently classifying the processed instances through training, which involves minimizing the error of classifications by adjusting the weights of edges accordingly. Thereby, ANN enable supervised training where labels for all classes are available (i.e., data samples are marked with labels such as normal or anomalous), semi-supervised training where labels of some classes are available, as well as unsupervised learning where no labels are available.

In general, deep learning algorithms are understood as neural networks with multiple hidden layers. Several different architectures of deep neural networks have been proposed in the past, such as recurrent neural networks (RNN) for sequential input data. Deep learning has been shown to outperform conventional machine learning methods (e.g., support vector machines or decision trees) and even human experts in many application areas such as image classification, speech recognition, and many more [8], [13].

2) Log Data: Log data are a chronological sequence of single- or multi-line events generated by applications to permanently capture specific system states, in particular, for manual forensic analysis in case that failures or other unexpected incidents occur. Log events are usually available in textual form and range from structured vectors (e.g., comma-separated values) over semi-structured objects (e.g., key-value pairs) to unstructured human-readable messages with heterogeneous event types. Despite the fact that no unified log format exists, log events usually contain their generation time stamp as one of their event parameters. Other parameters that are sometimes present in different types of log data are logging levels (e.g., *INFO* or *ERROR*) or process identifiers that link sequences of related events [3].

While single log events describe (part of) the system state in one particular point in time, groups of log events represent the dynamic workflows of the underlying program logic. The reason for this is that log events are generated by print statements purposefully placed by software developers throughout their code to support understanding of program activities and debugging. These statements comprise of static parts, i.e., hard-coded strings, and variable parts, i.e., parameters that are dynamically determined during program runtime. In the past, a large amount of research was directed towards automatic extraction of so-called log keys (also known as log signatures, log templates, or simply log events) that represent templates for the original print statements and enable parsing of logs [14]. These parsers allow to derive values from logs that are more suitable to be used for subsequent analysis than unstructured log messages, in particular, through (i) assignment of event type identifiers to log events and (ii) extraction of parameters from log messages [15].

3) Anomaly Detection: Anomalies are those instances in a data set that exhibit rare or otherwise unexpected characteristics and thus stand out from the rest of the data [7]. For the purpose of detection, the conformity of these data instances is usually measured through one or more continuous or categorical attributes that are associated with each instance and enable the computation of similarity metrics. For independent data, it is sufficient to declare single or small groups of instances with high dissimilarities to all other data points as outliers, which are also referred to as point anomalies. For all data where instances are not independent from each other, e.g., all kinds of ordered data including log data, two additional types of anomalies occur. First, contextual anomalies are instances that are only anomalous with respect to the context they occur (but not otherwise), such as the time of occurrence. For example, consider the start of a daily executed data backup procedure that suddenly takes place outside of the scheduled times. Second, collective anomalies are groups of instances that are only anomalous due to their combined occurrence (but not individually), such as a specific sequence of log events.

An implicit assumption of most anomaly detection techniques is that the analyzed data holds far fewer anomalies than normal instances. This enables that detection takes place in a fully unsupervised manner, i.e., no labeled data is necessary to train the models. However, many scientific approaches instead pursue semi-supervised detection, where training data containing only normal instances are available and evaluation then takes place on a test data set comprising both normal and anomalous instances. The main advantages of semi-supervised operation is that anomalous instances are not learned by the models and that it is often relatively simple to gather normal data, while modeling and labeling anomalies is less straightforward. Accordingly, approaches for supervised anomaly detection have lower applicability and are comparatively rare [4].

B. Challenges

Log-based anomaly detection has been an active field of research for decades. Thereby, most of the presented approaches rely on conventional machine learning techniques. However, the last few years have seen a strong increase of approaches that leverage deep learning to disclose anomalous log events that relate to unexpected system behavior. In the following, we summarize the main challenges that need to be overcome for effective and applicable detection.

- **Data representation**. Deep learning systems generally consume structured and numeric input data. It is non-trivial to feed log data into neural networks as they frequently involve a mix of heterogeneous event types, unstructured messages, and categorical parameters [11], [16].
- **Data instability**. As applications evolve, new log event types may occur that differ from the ones in the training data. In addition, the observed system behavior patterns are subject to change as technological environments and their utilization vary over time. Deep learning systems therefore need to incrementally update their models and adapt their baseline for normal system behavior to enable real-time detection [9]–[11], [16], [17].
- **Class imbalance**. Anomaly detection inherently assumes that normal events outnumber anomalous ones. Many approaches based on neural networks are known to perform sub-optimally for imbalanced data sets [10], [16].
- **Anomalous artifact diversity**. Manifestations of anomalies affect log events as well as parameters thereof in various ways, including changes of sequential patterns, frequencies, correlations, inter-arrival times, etc. Detection techniques are often designed only for properties of specific anomaly types and are therefore not generally applicable.
- **Label availability**. As anomalies represent unexpected system behavior, there are generally no labeled anomaly instances available for training. This restricts applications to semi- and unsupervised deep learning systems, which are known to achieve lower detection performance than supervised approaches [9], [10].
- **Stream processing**. Logs are generated as a continuous stream of data. To enable on-the-fly monitoring rather than forensic analysis, deep learning systems need to be designed for single-pass data processing when it comes to detection and model updating [3], [10].
- **Data volume**. Log data is generated in high volumes, with some systems producing millions [18] or even billions [19] of events daily. Efficient algorithms are required to ensure real-time processing in practical applications, in particular, when running on machines with few computational resources such as edge devices.
- **Interleaving logs**. Sequences of related log events may be interleaving each other when many processes operate simultaneously or distributed logs are collected centrally. It is non-trivial to retrieve the original event sequences when events lack session identifiers [10].
- **Model explainability**. Approaches based on neural networks generally suffer from a lower explainability than conventional machine learning methods. Difficulties to understand the reasons behind both correct and incorrect classifications are especially problematic when it comes to making justified decisions in response to critical system behavior or security incidents [9], [16].

III. Survey Method

This section outlines the method that was used to carry out the systematic literature review. We first describe our strategy for collecting relevant literature and then present the evaluation criteria that we used to analyze the retrieved papers.

A. Search Strategy

In this section we describe the process of gathering relevant publications to be included in the survey. First, an initial collection of literature is collected using a web search. Subsequently, relevant papers are selected using inclusion, exclusion, and quality criteria.

1) Initial Literature Collection with Search String: In order to obtain an initial set of approaches from the state-of-the-art, we assemble a search string to query common databases for scientific publications. In particular, we design the search string so that only publications containing the three main concepts relevant for this survey are retrieved: log data, anomaly detection, and deep learning. Since some publications use different terminology and to decrease the likelihood that relevant publications are missed, we also use the terms system log(s), event log(s), log file(s), log event(s) as alternatives for log data, and neural network(s) as an alternative for deep learning. Note that we consider both singular and plural forms of the terms. Figure 1 displays the final search string.

We then use this search string to gather publications from the following databases: Science Direct[1], Scopus[2], Springer-

[1]https://www.sciencedirect.com/
[2]https://www.scopus.com/

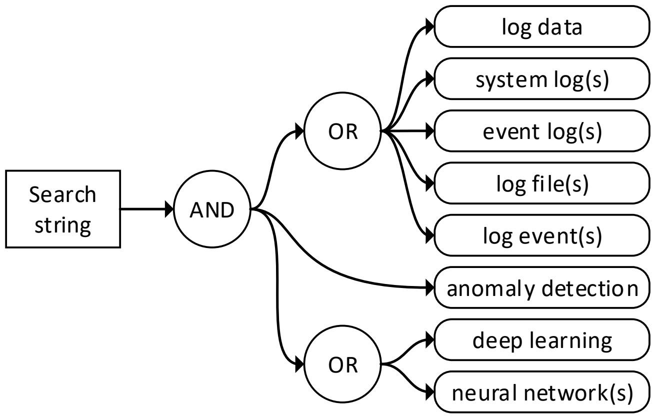


Fig. 1. Composition of the search string used to retrieve relevant literature.

Link[3], ACM Digital Library[4], IEEE Xplore[5], Google Scholar[6], and Web of Science[7]. The search was conducted in January of 2022 and returned a total of 2925 publications. In the following section, we describe our method for selecting relevant publications from this set.

2) Selection of Relevant Publications: To sort out publications that are not relevant for this survey and reduce the set of publications to a manageable size, we define multiple selection criteria and apply them on our initial collection. Our main criterion for including the publication in the survey is as follows: *The model proposed in the publication applies deep learning techniques (i.e., a multi-layered neural network) for anomaly detection in heterogeneous and unstructured log data*. Moreover, we define several exclusion criteria that we use to omit publications with low relevance or otherwise inappropriate format. The list of exclusion criteria is as follows.

- There is no indication stated in the paper that the presented approach is applicable or designed for application with log data.
- There exists a more recent publication that presents the same or a similar study.
- The publication only applies an existing approach without novel modifications from the original concept.
- The publication is in any language other than English.
- The publication is not available in electronic form.
- The publication is a book, technical report, lecture note, presentation, review, or thesis.

Note that we do not constraint the publications to a specific time range in order to avoid missing any older publications that are nonetheless relevant for this survey. However, we aim to omit publications of generally lower quality that do not meet the minimum scientific standards. We therefore only select publications that meet the following quality criteria in addition to our main inclusion criterion.

- The purpose of the study and its findings are explicitly stated.
- The applied deep learning models and their parameters are rigorously described.
- The publication includes a sound evaluation of the presented approach.
- The data sets used for evaluating the approach are referenced or described.
- Visualizations are clear and readable.

Our selection procedure is a two-stage process. First, we reduce the initial collections of publications using the inclusion and exclusion criteria based on the title and abstract of each paper. After this stage 331 publications remained. In the second stage, we carry out the selection using all aforementioned criteria based on the contents of each paper. We eventually obtained 61 papers that were included in this survey. The following section outlines our method for analyzing these publications.

B. Reviewed features

To ensure that we analyze the selected publications on a common scheme and address our research questions, we formulate a list of features that we assess for each paper. The following set of questions concerns the applied deep learning (DL) model and mode of operation.

DL-1: Which deep learning models are used?
DL-2: Which training loss functions are applied?
DL-3: Does the approach support online or incremental[8] learning?
DL-4: Does training take place in un-, semi-, or supervised manner?

We then analyze the different ways how the reviewed approaches feed raw log data into deep learning models. The following questions therefore address the pre-processing (PP) and transformation of logs into numeric vector or matrix representations.

PP-1: How are raw logs pre-processed?
PP-2: What features are extracted from pre-processed logs?
PP-3: How are extracted features represented as vectors?

The next set of questions deals with the anomaly detection (AD). In particular, we are interested in the different types of anomalies to understand whether they are linked to the features extracted from the raw logs and their representations as vectors.

AD-1: What types of anomalies are detected by the approach?
AD-2: How is the output of the deep neural network[9] used for anomaly detection?
AD-3: How are anomalies differentiated from normal data samples?

Utilizing openly accessible data sets for evaluations as well as publishing source code alongside papers is not only good scientific standard but also essential for others to validate presented results and carry out comparisons. The last set of questions therefore concerns evaluation and reproducibility

[3]https://link.springer.com/
[4]https://dl.acm.org/
[5]https://ieeexplore.ieee.org/
[6]https://scholar.google.com/
[7]https://www.webofscience.com/

[8]Online or incremental processing refers to single-pass procedures where the runtime grows approximately linear with the number of processed lines.
[9]The output layer of the neural network comprises one or more nodes with certain numeric values.

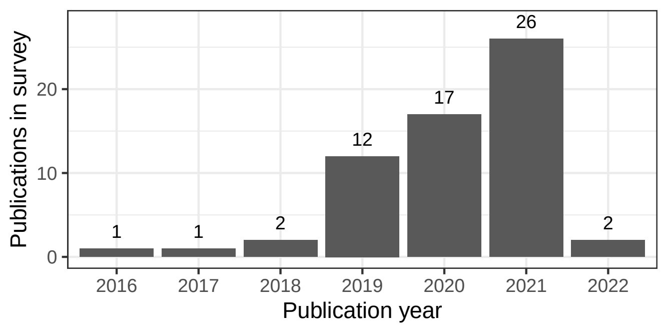


Fig. 2. Distribution of the number of publications per year.

(ER), in particular, employed evaluation metrics as well as availability of data sets and source code.

ER-1: What log data sets are used for evaluating the approach?
ER-2: What evaluation metrics are employed?
ER-3: Does the evaluation consider runtime performance measurements?
ER-4: What approaches are used as benchmarks?
ER-5: Are the used data sets publicly available?
ER-6: Is the source code of the approach publicly available?

All aforementioned questions were assessed for each publication individually. The resulting feature matrix is presented in Table II in the following section and serves as the basis for our analyses and discussions.

IV. Survey Results

This section provides the assessments of all reviewed publications with respect to the features outlined in the previous section. We first provide some general information on the meta-data of publications before going over each reviewed feature in detail.

A. Bibliometrics

This section provides an overview of the distribution of publications per year as well as their citation counts.

1) Publications per Year: Deep learning for anomaly detection in log data is a relatively new research field that has increasingly gained traction in the last years. Accordingly, a majority of the publications in this research area have only been published in the last two to three years. Figure 2 shows an overview of the publication years of all publications reviewed for this survey. As expected, 57 out of the 61 considered publications were published in 2019 or later. As this survey is carried out in the beginning of 2022, only two publications from that year are included. However, we expect to see an even higher number of publications in 2022 and beyond following the overall trend visible in the plot.

2) Citations: Citation counts are a common indicator to assess the relevance and influence of publications. We therefore state the top six publication with the highest citation counts (according to Google Scholar) in Table I. As of May 2022, the paper presenting DeepLog by Du et al. [20] that was published in 2017 has the highest citation count and is arguably the most influential of all reviewed publications as they were the first to propose an approach based on deep learning that enables detection of anomalous event sequences in log data. Several of the subsequently published papers rely on the groundwork of DeepLog and it is therefore fair to assume that this paper is at least to some degree responsible for the increase of relevant publications from 2019 and onward that is visible in Fig. 2.

Note that the publication by Yang et al. [21] predates DeepLog [20] but has a significantly lower citation count. The main reason for this is that the paper focuses on the analysis of tokens in single log events, a topic that received far less attention in subsequent research than the analysis of event sequences. This differentiation as well as assessments for all other features stated in Sect. III-B are presented in Table II.

B. Deep Learning Techniques

This section provides an overview of the properties of deep learning models applied in reviewed publications.

1) Deep Learning Models: There are many different types of deep learning models (DL-1) that are suitable to be used for anomaly detection in log data [13], [79]. The most basic form of a deep learning neural network is that of a **Multi-Layer Perceptron** (MLP), where all layers in the network are fully connected. Due to their simplicity, their classification accuracies are usually outperformed by other deep learning models that are specifically designed to capture common characteristics present in sequential data. Accordingly, they are rarely considered in the reviewed literature and only occur in combination with other deep learning models or as supplementary attention mechanisms [1], [45].

Convolutional Neural Networks (CNN) extend upon the architecture of MLPs by inserting convolutional and max pooling layers within the hidden layers. These layers enable that the neural networks capture more abstract features of the input data and at the same time reduce the input dimensions. This has proven especially effective when classifying 2-dimensional input data from images, where features such as lines are learned independent from their exact location in the image. This functionality is transferred to log data by arranging the log keys within a matrix so that the relationships between events, i.e., their temporal dependencies, are captured by the network [23]. There also exist several approaches that rely on specific types of CNNs, such as temporal convolutional networks (TCN) that are specifically designed to process time-series and capture their short- and long-term dependencies through dilated causal convolutions [61], [73].

As visible in Table II, **Recursive Neural Networks** (RNN) are the most commonly used neural network architectures in the surveyed literature, with 36 out of 61 reviewed approaches leveraging RNNs for anomaly detection. The main reason for this is that the architecture of RNNs leverages feedback mechanisms that retain their states over time and thus directly enable learning of sequential event execution patterns in input data, which are the key identifiers for anomalies in log data sets that are commonly used in evaluations (cf. Sect. IV-E). Several different types of RNNs have successfully been applied for this purpose. One of the most widespread architectures are

TABLE I
TOP SIX MOST CITED PUBLICATIONS

Citations	Approach	Year	Authors	Paper Title
753	DeepLog	2017	Du et al. [20]	DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning
154	LogRobust	2019	Zhang et al. [17]	Robust Log-Based Anomaly Detection on Unstable Log Data
145	LogAnomaly	2019	Meng et al. [22]	LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs
65	-	2018	Lu et al. [23]	Detecting Anomaly in Big Data System Logs Using Convolutional Neural Network
34	Logsy	2020	Nedelkoski et al. [24]	Self-Attentive Classification-Based Anomaly Detection in Unstructured Logs
28	nLSALog	2019	Yang et al. [25]	nLSALog: An Anomaly Detection Framework for Log Sequence in Security Management

Long Short-Term Memory (LSTM) RNNs that are developed to enable long-time storage of states and comprise cells with input gates, output gates, and forget gates. A majority of the approaches leveraging LSTM RNNs train the network with sequences of event occurrences (or modified and enriched versions thereof) and subsequently disclose unusual sequential patterns in test data as anomalies [20]. Some approaches make use of Bi-LSTM RNNs, which are basically two independent LSTM RNNs that work in parallel and process sequences in opposite directions, i.e., while one LSTM RNN processes the input sequences as usual from the first to the last element, the other LSTM RNN processes sequence elements starting from the last entry and predicts elements that chronologically precede them. Experiments suggest that Bi-LSTM RNNs outperform LSTM RNNs [17], [34], [46], [50], [55], [57], [72], [75]. Another popular choice for RNNs are Gated Recurrent Units (GRU) that simplify the cell architecture as they only rely on update and reset gates. One of the main benefits of GRUs is that they are computationally more efficient than LSTM RNNs, which is a relevant aspect for use cases focusing on edge devices [21], [34], [35], [37], [53], [56], [62], [68], [69].

While aforementioned deep learning models are primarily used for classification problems across different research fields, there are also models that are specifically designed to operate in unsupervised manner and are thus a natural choice for anomaly detection. One of them are **Autoencoders** (AE), which first create a code from the input data using an encoder and then try to approximate the input from the code using a decoder, thereby avoiding the need for labeled input data. The main idea is that through this process the neural network learns the main features from the input but neglects the noise in the data, similar to dimension reduction techniques such as principal component analysis (PCA). Any input data that is fed into an already trained network and yields a high reconstruction error is then considered as anomalous. Besides the standard model for Autoencoders, there are also several related types, such as Variational Autoencoders (VAE) that operate on statistical distributions [1], [52], [58], Conditional Variational Autoencoders (CVAE) that add conditional information such as event types to the training [49], and Convolutional Autoencoder (CAE) that leverage the advantages of CNNs regarding learning of location-independent features [58].

Generative Adversarial Networks (GAN) are another approach for unsupervised deep learning. They actually consist of two separate components that compete with each other: a generator that produces new data that resembles the input data, and a discriminator that estimates the probability that some data stems from the input data, which is used to improve the generator. Existing approaches use different models to construct GANs, including LSTM RNNs [41], [66], CNNs in combination with GRUs [35], and Transformers [1], [76].

Transformers (TF) make use of so-called self-attention mechanisms to embed data instances into a vector space, where similar instances should be closer to each other than dissimilar ones [24], [33], [39]. The goal of Transformers is to assign weights to specific inputs according to the context of their occurrence, such as words in sentences. Accordingly, Transformers have been particularly successful in the area of natural language processing (NLP). **Attention mechanisms** (AT) do not just appear in Transformers, but are also frequently used to improve classification and detection in other deep neural networks such as RNNs by weighting relevant inputs higher. This effect is particularly strong when long sequences are ingested by RNNs [73]. Attention mechanisms are usually realized as trainable networks such as MLPs [45]. In order to avoid confusions with the Transformer model, we state these additional attention mechanisms explicitly in Table II.

Wan et al. [59] are the only authors to utilize **Graph Neural Networks** (GNN). While neural networks typically ingest ordered data, e.g., CNNs rely on 2-dimensional input data and RNNs require sequences of observations, GNNs are designed to ingest graph inputs, i.e., sets of vertices and edges. One possibility to transform log data into graphs is to generate session graphs, with vertices representing events and edges their sequential executions. Another less commonly applied type of deep learning model is the **Evolving Granular Neural Network** (EGNN), a fuzzy inference system that is gradually constructed from online data streams [32].

Our survey shows that many of the reviewed approaches rely on only one type of deep learning model. However, some authors also use combinations of different models. For example, Wang et al. [1] propose to use a MLP to combine the output of a VAE with that of a Transformer trained with adversarial learning.

TABLE II
Survey results. DL-1: Multi-Layer Perceptron (MLP), Convolutional Neural Network (CNN), Recursive Neural Network (RNN), Autoencoder (AE), Generative Adversarial Network (GAN), Transformer (TF), Attention mechanism (AT), Graph Neural Network (GNN), Evolving Granular Neural Network (EGNN); DL-2: Cross-Entropy (CE), Hyper-Sphere (HS), Mean Squared Error (MSE), Kullback-Leibler Divergence (KL), Marginal Likelihood (ML), Custom Loss Function (CF), Adversarial Training (AT), Not Available (NA); DL-3: Online (ON), Offline (OFF); DL-4: Supervised (SUP), Semi-supervised (SEMI), Unsupervised (UN); PP-1: Log key (KEY), Token (TOK), Combination (COM); PP-2: Token Sequence (TS), Token Count (TC), Event Sequence (ES), Event Count (EC), Parameter (PA), Event Interval Time (EI); PP-3: Event ID sequence (ID), Count Vector (CV), Statistical Feature Vector (FV), Semantic Vector (SV), Positional Embedding (PE), One-Hot Encoding (OH), Embedding Layer/Matrix (EL), Deep Encoded Embedding (DE), Parameter Vector (PV), Time Embedding (TE), Graph (G), Transfer Matrix (TM); AD-1: Outlier (OUT), Sequential (SEQ), Frequency (FREQ), Statistical (STAT); AD-2: Binary Classification (BIN), Input Vector Transformations (TRA), Reconstruction Error (RE), Multi-class Classification (MC), Probability Distribution (PRD), Numeric Vector (VEC); AD-3: Label (LAB), Threshold (THR), Highest Probabilities (TOP).

Approach	DL-1	DL-2	DL-3	DL-4	PP-1	PP-2	PP-3	AD-1	AD-2	AD-3	ER-6
Baril et al. [26] (NoTIL)	RNN	CE	OFF	SEMI	KEY	ES, EC	CV	FREQ	VEC	THR	NO
Bursic et al. [27]	RNN, AE	MSE	OFF	UN	TOK	PA, TS	DE	OUT	RE	THR	NO
Catillo et al. [28] (AutoLog)	AE	MSE	ON	SEMI	COM	STAT, TC	FV	FREQ	RE	THR	YES
Cheansunan et al. [29]	CNN	NA	OFF	SEMI	KEY	ES	EL	SEQ	PRD	TOP	NO
Chen et al. [30]	RNN, CNN	NA	OFF	SEMI	KEY	ES, EC	SV, CV	SEQ, FREQ	VEC, PRD	TOP	NO
Chen et al. [31] (LogTransfer)	RNN, CNN	NA	OFF	SUP	KEY	ES	SV	SEQ	BIN	LAB	YES
Decker et al. [32]	EGNN	CF	OFF	SUP	KEY	EC, STAT	FV	FREQ	MC	THR, LAB	NO
Du et al. [20] (DeepLog)	RNN	CE, MSE	ON	SEMI	KEY	ES, PA, EI	PV, OH	SEQ	VEC, PRD	THR, TOP	RE
Du et al. [33] (LogAttention)	TF, AM	NA	OFF	SUP	KEY	ES	SV	SEQ	BIN	LAB	NO
Farzad et al. [34]	RNN, AE	CE	OFF	SUP	TOK	TS	EL	SEQ	BIN	LAB	NO
Farzad et al. [35]	RNN, AE, CNN, GAN	CE	OFF	SUP	TOK	TS	EL	OUT	BIN	LAB	NO
Farzad et al. [36]	RNN	CE	OFF	SUP	TOK	TS	EL	OUT	BIN	LAB	NO
Gu et al. [37]	RNN, AM	CE	OFF	SEMI	KEY	ES	SV	SEQ	PRD	TOP	NO
Guo et al. [38] (FLOGCNN)	CNN	CE	OFF	SUP	COM	TS	SV	SEQ	BIN	LAB	NO
Guo et al. [39] (LogBERT)	TF, AM	CE, HS	OFF	SEMI	KEY	ES	PE, EL	SEQ	PRD	TOP	YES
Guo et al. [40] (TransLog)	TF, AM	CE	OFF	SUP	KEY	ES	SV	SEQ	BIN	LAB	NO
Han et al. [41] (LogTAD)	RNN, GAN	HS, CF, AT	OFF	UN	KEY	ES	SV	SEQ	TRA	THR	YES
Hashemi et al. [42] (OneLog)	CNN	CE	OFF	SUP	TOK	ES	DE	SEQ	BIN	LAB	YES
Hirakawa et al. [43]	CNN, TF	CF	OFF	UN	TOK	ES	SV	OUT	BIN	THR	NO
Huang et al. [44] (HitAnomaly)	TF, AM	CE	OFF	SUP	KEY	ES, PA	SV, PV	SEQ	BIN	LAB	NO
Li et al. [45] (LogSpy)	CNN, MLP, AM	CE	OFF	SUP	KEY	STAT	DE, EL	STAT	BIN	LAB	NO
Li et al. [46] (SwissLog)	RNN, AM	CE	OFF	SUP	KEY	ES, EI	SV, TE	STAT, SEQ	BIN	LAB	NO
Liu et al. [47] (LogNADS)	RNN	MSE	OFF	SUP	KEY	ES, PA	SV, PV	SEQ	BIN	LAB	NO
Lu et al. [23]	CNN	NA	OFF	SUP	KEY	ES	EL	SEQ	BIN	LAB	NO
Lv et al. [48] (ConAnomaly)	RNN	NA	OFF	SUP	KEY	ES	SV	SEQ	BIN	THR	NO
Meng et al. [22] (LogAnomaly)	RNN	NA	OFF	SEMI	KEY	ES, EC	SV, CV	SEQ, FREQ	VEC, PRD	TOP	RE
Nedelkoski et al. [24] (Logsy)	TF	HS	OFF	UN	TOK	TS	PE, EL	OUT	TRA	THR	RE
Otomo et al. [49]	AE	KL, ML, CF	OFF	SEMI	KEY	ES, EC	CV, OH	FREQ	TRA	THR	NO
Ott et al. [50]	RNN	CE, MSE	OFF	SEMI	KEY	ES	SV	SEQ	VEC, PRD	THR, TOP	NO
Patil et al. [51]	RNN	CE	OFF	SUP	KEY	ES	OH	SEQ	BIN	LAB	NO
Qian et al. [52] (VeLog)	AE	CF	ON	SEMI	KEY	ES, EC	ID, CV	SEQ, FREQ	RE	THR	NO
Studiawan et al. [53]	AE	MSE	OFF	SEMI	KEY	EC, STAT, EI, TC	FV	STAT, FREQ	RE	THR	NO
Studiawan et al. [54] (pylogsentiment)	RNN	CE	OFF	SUP	TOK	TS	SV	OUT	BIN	LAB	YES
Sun et al. [55] (AllContext)	RNN, AM	CE	OFF	SUP	KEY	ES	SV	OUT, SEQ	BIN, MC	LAB	NO
Sundqvist et al. [56] (BoostLog)	RNN	NA	ON	SEMI	KEY	ES	ID	SEQ	PRD	THR	NO
Syngal et al. [57]	RNN, AE	CE	OFF	SUP	TOK	ES, EC	CV, OH	SEQ, FREQ	RE	THR	NO
Wadekar et al. [58]	AE	CE, KL, ML	OFF	UN	KEY	ES	OH	SEQ	BIN	THR	NO
Wan et al. [59] (GLAD-PAW)	GNN	CE	OFF	UN	KEY	ES	G	SEQ	PRD	TOP	NO
Wang et al. [60]	RNN	NA	OFF	SUP	TOK	TS	SV	OUT	BIN	LAB	NO
Wang et al. [1] (CATLog)	AE, MLP, TF	CE, CF	OFF	SUP	KEY	ES, TC	SV, DE, CV	SEQ, FREQ	BIN	LAB	NO
Wang et al. [61] (LightLog)	CNN	CE	OFF	SUP	KEY	ES	SV	SEQ	BIN	LAB	YES
Wang et al. [62] (OC4Seq)	RNN	HS	OFF	UN	KEY	ES	EL	SEQ	TRA	THR	YES
Wibisono et al. [63]	TF, AM	NA	OFF	SEMI	KEY	ES	OH	SEQ	PRD	TOP	NO
Wittkopp et al. [64] (A2Log)	TF, AM	CF	OFF	UN	TOK	TS	SV	OUT	BIN, TRA	THR	NO
Xi et al. [65]	RNN, AM	CE	OFF	SEMI	KEY	ES	SV	SEQ	PRD	TOP	NO
Xia et al. [66] (LogGAN)	RNN, GAN	AT	ON	SEMI	KEY	ES	OH	SEQ	PRD	THR	NO
Xiao et al. [67]	RNN, CNN	CE	OFF	SEMI	KEY	ES	EL	SEQ	PRD	TOP	NO
Xie et al. [68] (ATT-GRU)	RNN, AM	MSE	OFF	SEMI	KEY	ES, PA, EI	PV, EL	SEQ	VEC, PRD	THR, TOP	NO
Yang et al. [21]	RNN	CE	OFF	SEMI	TOK	TS	SV	OUT	VEC	THR	NO
Yang et al. [25] (nLSALog)	RNN, AM	CE	OFF	SEMI	KEY	ES	EL	SEQ	PRD	TOP	NO
Yang et al. [69] (PLELog)	RNN	NA	OFF	SEMI	KEY	ES	SV	SEQ	BIN	LAB	NO
Yen et al. [70] (CausalConvLSTM)	RNN, CNN	CE	ON	SEMI	KEY	ES	OH	SEQ	PRD	TOP	NO
Yin et al. [71] (LogC)	RNN	CE	OFF	SEMI	KEY	ES, PA	OH	SEQ	PRD	TOP	NO
Yu et al. [72]	RNN	CE	OFF	SEMI	KEY	ES, EC	SV, ID, CV	SEQ, FREQ	PRD	TOP	NO
Zhang et al. [17] (LogRobust)	RNN, AM	CE	OFF	SUP	KEY	ES	SV	SEQ	BIN	LAB	RE
Zhang et al. [73] (LogAttn)	AE, CNN, AM	CE	OFF	SEMI	TOK	ES, EC	SV, CV	SEQ, FREQ	RE	THR	NO
Zhang et al. [74] (LSADNET)	CNN, TF	CE	ON	SEMI	KEY	ES, STAT	SV, TM	STAT, SEQ	PRD	TOP	NO
Zhang et al. [75] (SentiLog)	RNN	NA	OFF	SUP	TOK	TS	SV	OUT	BIN	LAB	NO
Zhao et al. [76] (Trine)	TF, GAN	NA	OFF	SEMI	TOK	ES	SV, DE, PE	SEQ	BIN	LAB	NO
Zhou et al. [77] (LogSayer)	RNN, CNN	NA	ON	SUP	KEY	EC, STAT	FV	STAT	BIN	LAB	NO
Zhu et al. [78] (LogNL)	RNN	NA	OFF	SEMI	KEY	ES, PA, EI	SV, PV	SEQ	VEC, PRD	THR	NO

2) Training loss function: Loss functions (DL-2) are essential for training of deep neural networks as they quantify the difference between the output of the neural network and the expected result [80]. The most common loss function in the reviewed publications is the **Cross-Entropy** (CE), in particular, the categorical cross-entropy for multi-class prediction [20], [57] or binary cross-entropy that only differentiates between the normal and anomalous class [61]. Other common loss functions include the **Hyper-Sphere Objective Function** (HS) where the distance to the center of a hyper-sphere represents the anomaly score [24], [39], [41], [62], the **Mean Squared Error** (MSE) that is used for regression [20], [27], [28], [47], [50], [53], [68], and the **Kullback-Leibler Divergence** (KL) and **Marginal Likelihood** (ML) that are useful to measure loss in probability distributions [49], [58].

Some of the presented approaches are trained with **Custom Loss Functions** (CF), including combinations of CE and HS [64] or KL and ML [49]. Some authors also define objective functions specifically for **Adversarial Training** (AT) of GANs [41], [66]. Out of all publications, 14 do not state the loss function and are therefore marked as **Not Available** (NA).

3) Operation mode: When applying anomaly detection in real world scenarios, not all log data is available at any time; instead, events are generated as a continuous stream and should be analyzed only at their time of occurrence in order to enable (close to) real-time detection. Thereby, the structural integrity and statistical properties of the generated logs vary over time, e.g. the overall system utilization is not stationary as user interactions and the technological environment are subject to change. In addition, log templates change or new log events occur when applications generating these logs are modified [17].

To keep up with these changes in an automated way, algorithms need to adopt online or incremental learning (DL-3), i.e., ingest input data with linear time complexity so that data sets are processed in a single pass where each data instance is only handled once. While it is usually always possible to carry out detection in an online fashion when trained models are considered static, it is significantly more challenging to develop algorithms that support online learning and dynamically update their models to adapt to new events or patterns by incorporating them into the baseline for detection [81], [82]. As continuous learning is still an open problem, we mark all approaches that enable dynamic model updates at least to some degree as **online** (ON) and all other approaches with offline training phases as **offline** (OFF) in Table II.

The reviewed approaches addressed dynamic model adaptation in multiple ways. Du et al. [20] suggest to update the weights of their neural network when false positives are identified during the detection to reflect the correct event probability distributions without the need to re-train from scratch. Meng et al. [22] argue that such a manual feedback loop is infeasible in practice and therefore resort to re-training where new event types are mapped to existing ones. Yen et al. [70] automatically re-train their neural network on batches of new log data when false positive rates exceed a certain threshold. However, calculating the false positive rates relies on labeled data and thus does not remove the human from the loop. A promising solution to aforementioned problems is presented by Decker et al. [32], who employ an evolving classifier that is specifically designed to handle unstable data streams and could thus enable continuous learning.

In general, both online and offline models can work in **supervised** (SUP) as well as **unsupervised** (UN) fashion (DL-4). However, we noticed that almost all supervised learning approaches in the reviewed publications opt for an offline training phase. This is reasonable as the label information required for supervised learning relies on manual analysis or validation and therefore can only be generated forensically for delimited data sets, but not for data streams.

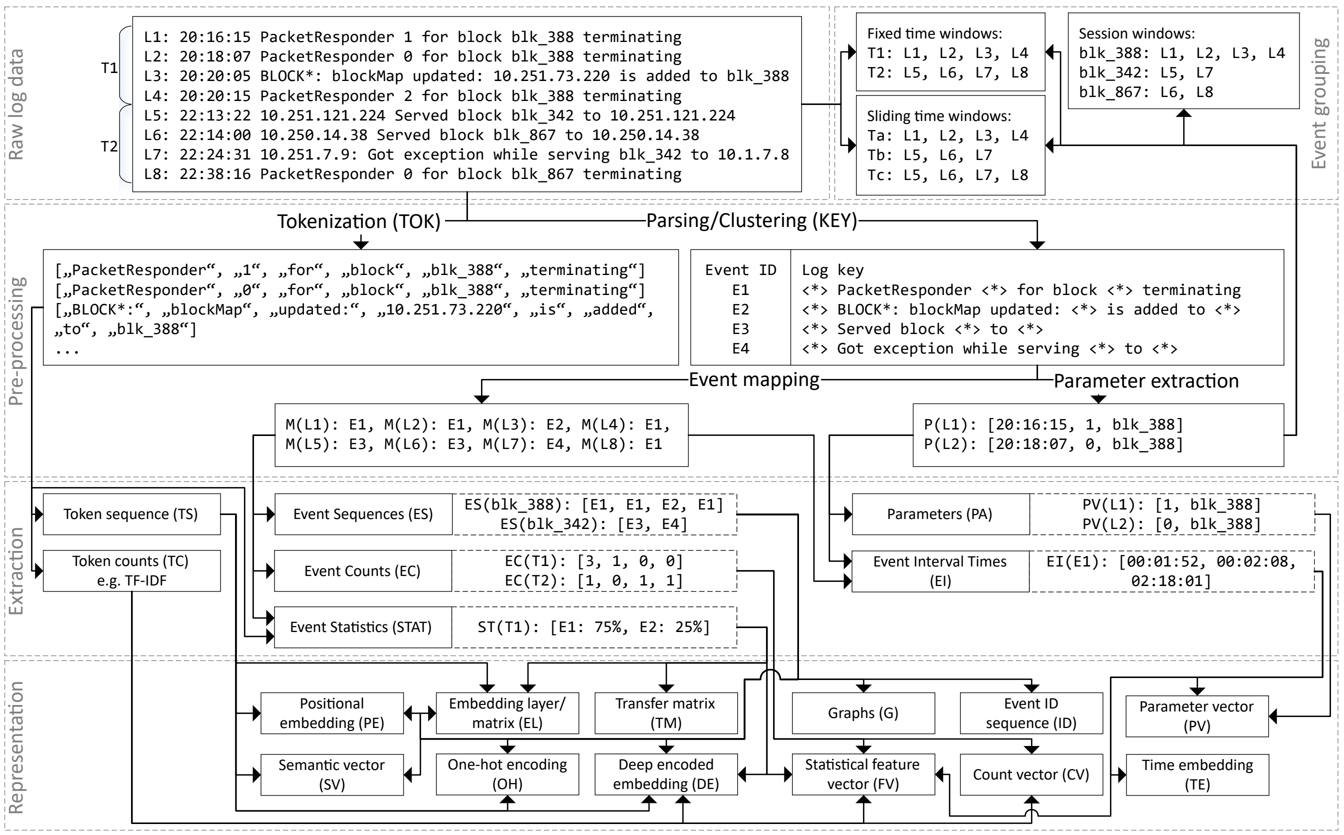
We also made the observation that many reviewed approaches claim to enable unsupervised learning, but are in fact operated in **semi-supervised** (SEMI) fashion as they usually assume that training takes place only on normal data that is free of anomalies [30]. The main problem is that anomalies that are present in training data would incorrectly change the weights of the neural networks and thus deteriorate their detection in the subsequent detection phase. Accordingly, only deep learning models that are designed for unsupervised learning are capable of handling anomalies in training data, for example, the approach based on a CVAE model presented by Otomo et al. [49]. Note that neural network architectures that would support unsupervised learning may also be applied for semi-supervised detection and are therefore not necessarily marked as unsupervised.

C. Log Data Preparation

This section outlines all steps necessary to prepare raw and textual log data for deep learning. This includes grouping of events into windows or sessions, tokenization and log parsing, extraction of features from the logs, as well as transformation of these features into vector representations that are suitable as input to neural networks. Figure 3 provides an overview of these data preparation methods used in the reviewed literature and illustrates them on the sample log lines L1-L8. In the following sections, we discuss all stated methods in detail.

1) Pre-processing: As log data is generally unstructured it is necessary to pre-process them in some way before feeding them into deep learning systems (PP-1). Our survey shows that there are two main approaches to handle unstructured data. The most common approach is to leverage parsers, which are usually referred to as **log keys** (KEY), to extract unique event identifiers for each line as well as event parameter values as described in Sect. II-A2. Thereby, authors usually re-use existing log keys for well-known data sets or create the keys using state-of-the-art approaches for parser generation, such as Drain [83] or Spell [84]. Typically, each line matches exactly one key; it is thus easy to assign a unique event type identifier to each log line during event mapping. For example, in Fig. 3 line L1 is mapped to event type identifier E1 as it matches the corresponding log key. Moreover, the figure also shows that parsing allows to extract all parameters from log events, in particular, the time stamp and identifiers for packet responder and file block are extracted from L1 and stored in a list.

An alternative to parsing are **token-based** (TOK) strategies that split log messages into lists of words, for example, by



```
L1: 20:16:15 PacketResponder 1 for block blk_388 terminating
L2: 20:18:07 PacketResponder 0 for block blk_388 terminating
L3: 20:20:05 BLOCK*: blockMap updated: 10.251.73.220 is added to blk_388
L4: 20:20:15 PacketResponder 2 for block blk_388 terminating
L5: 22:13:22 10.251.121.224 Served block blk_342 to 10.251.121.224
L6: 22:14:00 10.250.14.38 Served block blk_867 to 10.250.14.38
L7: 22:24:31 10.251.7.9: Got exception while serving blk_342 to 10.1.7.8
L8: 22:38:16 PacketResponder 0 for block blk_867 terminating
```

```
Fixed time windows:
T1: L1, L2, L3, L4
T2: L5, L6, L7, L8

Sliding time windows:
Ta: L1, L2, L3, L4
Tb: L5, L6, L7
Tc: L5, L6, L7, L8

Session windows:
blk_388: L1, L2, L3, L4
blk_342: L5, L7
blk_867: L6, L8
```

```
[„PacketResponder“, „1“, „for“, „block“, „blk_388“, „terminating“]
[„PacketResponder“, „0“, „for“, „block“, „blk_388“, „terminating“]
[„BLOCK*:“, „blockMap“, „updated:“, „10.251.73.220“, „is“, „added“,
„to“, „blk_388“]
...
```

```
Event ID   Log key
   E1      <*> PacketResponder <*> for block <*> terminating
   E2      <*> BLOCK*: blockMap updated: <*> is added to <*>
   E3      <*> Served block <*> to <*>
   E4      <*> Got exception while serving <*> to <*>
```

```
M(L1): E1, M(L2): E1, M(L3): E2, M(L4): E1,
M(L5): E3, M(L6): E3, M(L7): E4, M(L8): E1

P(L1): [20:16:15, 1, blk_388]
P(L2): [20:18:07, 0, blk_388]
```

```
ES(blk_388): [E1, E1, E2, E1]
ES(blk_342): [E3, E4]

EC(T1): [3, 1, 0, 0]
EC(T2): [1, 0, 1, 1]

ST(T1): [E1: 75%, E2: 25%]

PV(L1): [1, blk_388]
PV(L2): [0, blk_388]

EI(E1): [00:01:52, 00:02:08,
         02:18:01]
```

Fig. 3. Different stages of data preparation to transform raw log data into numeric vectors.

splitting them at white spaces. It is then common to clean the data by transforming all letters to lowercase and removing special characters and stop words before obtaining the final word vectors. While such approaches draw less semantic information from the single tokens, they have the advantage of being more flexible as they rely on generally applicable heuristics rather than pre-defined parsers and are therefore widely applicable. Some approaches make use of a **combination** (COM) of parsing and token-based pre-processing strategies, in particular, by generating token vectors from parsed events rather than raw log lines [28], [38].

2) Event grouping: As pointed out in Sect. II-A3, simple outlier detection does not require any grouping of logs since single unusual events are regarded as anomalies independent from the context in which they occur in. However, deep learning is most often applied to disclose unusual patterns of multiple log events, such as changes of event sequences or temporal log correlations. For these cases it is necessary to logically organize events into groups that are then analyzed individually or in relation to each other.

We illustrate common event grouping strategies in the top right of Fig. 3. Grouping into time windows is almost always feasible as log events are generally produced with a time stamp that documents their time of generation [3]. Since time stamps commonly occur in the beginning of log lines and are thus relatively easy to extract, it is not necessary to parse the remainder of the usually more complex log messages. There are two main strategies for time-based grouping [4]. First, **sliding time windows** are windows of a specific duration that are shifted across the log data with a fixed step size that is generally smaller and a whole number divisor of the window size. For every step where the time window is moved all logs with a time stamp within the current start and end time of the window are allocated to the same group, where each log line may appear in multiple groups as time windows overlap. For the sample logs in Fig. 3 we assume time window of 2 hours starting at 20:00:00 and a step size of 30 minutes. As visible in the figure, lines L5, L6, and L7 are contained in both Tb and Tc. Second, **fixed time windows** are special cases of sliding time windows where the step size is set to the same value as the window size. While this strategy results in a less fine-grained view on the data, the advantage is that each log event will only be allocated to exactly one time window, which makes subsequent computations such as time-series analysis easier. The log events in Fig. 3 are exemplarily grouped into fixed time windows of 2 hours, yielding $T1$ containing the first set of four lines and $T2$ containing the subsequent set of four lines. It must be noted that grouping based on sliding or fixed windows may also be carried out by numbers of lines rather than time so that each group of lines has the same size. While this avoids the need to process time stamps and ensures that the group sizes are fixed (e.g., there cannot be any empty groups), it is more difficult to consider event frequencies as time-series as the resulting windows represent varying time spans.

An entirely different grouping strategy are **session windows** that rely on an event parameter that acts as an identifier for a specific task or process where the event originated from. Grouping log events by these session identifiers allows to extract event sequences that correctly depict underlying program workflows even when multiple sessions are carried out in parallel on the monitored system. Unfortunately, not all types of log data come with such an identifier for sessions. In the sample logs depicted in Fig. 3 the file block identifier (e.g., blk_388) acts as a session identifier and thus allows us to exemplarily extract three event sequences.

3) Feature extraction: The parsing- and token-based pre-processing strategies described in Sect. IV-C1 enable the extraction of structured features from the otherwise unstructured logs (PP-2). Some of these features are directly derived from pre-processing logs, e.g., the **Token Sequence** (TS) may be used without any further modifications for analyzing each log line as a sentence of words. On the other hand, **Token Counts** (TC) require an additional step of computation where the tokens in each line are compared and counted, including advanced weighting techniques for each token such as term frequency–inverse document frequency (TF-IDF) that estimates token relevance based on token occurrences across all observed log lines.

Considering the outcome of the event mapping step it is simple to extract **Event Sequences** (ES), i.e., sequences of event type identifiers that are usually separated by fixed, sliding, or session windows (cf. Sect. IV-C2). For example, Fig. 3 shows that the event sequence for file block blk_388 is $[E1, E1, E2, E1]$ corresponding to the event identifiers for the respective log keys matching the lines L1-L4. **Event Counts** (EC) are vectors of length d, where the i-th element of the vector depicts the number of occurrences of the i-th log key and d is the total number of available log keys. The example in Fig. 3 shows the event count vector for time window $T1$ as $[3, 1, 0, 0]$, indicating that three log lines corresponding to the first log key (E1) appeared in $T1$, in particular, the lines L1, L2, and L4. Besides frequencies, many other **statistical properties** (STAT) may be computed from event occurrences, such as the percentage of seasonal logs [77], the lengths of log messages [53], log activity rates [32], entropy-based scores for chunks of log lines [28], or the presence of sudden bursts in event occurrences [77].

Parameters (PA) of log events are extracted as lists of values. Since the semantic meaning of each parameter is known after parsing, the values in each vector can be analyzed with methods that are appropriate for the respective value types, e.g., numeric or categorical. One special parameter of log events is the time stamp as it allows to put event occurrences into chronological order and infer dynamic dependencies. Accordingly, **Event Interval Times** (EI), i.e., inter-arrival times of log lines that belong to the same event type, are a frequently extracted feature for anomaly detection.

4) Feature representation: The extracted features described in the previous section comprise numeric or categorical vectors and are suitable to be consumed by neural networks. For example, event sequences are represented as **Event ID sequence vectors** (ID), i.e., chronologically ordered sequences of log key identifiers, and fed into RNNs to learn dependencies of event occurrences and disclose unusual sequence patterns as anomalies [56]. Event counts are represented as ordered **Count Vectors** (CV) and are also similarly used as input for RNNs [26]. Event statistics are another type of input that do not require any specific processing other than representing them in a **Statistical Feature Vector** (FV) where the position of each element in the vector corresponds to one particular feature.

While it is possible to directly use the extracted features, most of the approaches presented in the reviewed publications in fact rely on combinations or otherwise transformed vector representations of the original feature vectors (PP-3). Thereby, the most common representation is that of a **Semantic Vector** (SV). Within the field of NLP it is common practice to transform words of a sentence into so called semantic vectors that encode context-based semantics (e.g. Word2Vec [85], BERT [86], or GloVe [87]) or language statistics (e.g. TF-IDF [88]). Since each log line comprises sequences of tokens analogous to words in a sentence, it stands to reason to apply methods from natural language processing on the token sequences. Similarly, a sequence of multiple subsequent events can be regarded as a sentence, where each unique log key represents a word. Semantic encoding is typically achieved by training deep neural methods on a specific log file or by relying on pre-trained models. Semantic vectors are sometimes used in combination with **Positional Embedding** (PE), where elements (typically tokens) are encoded based on their relative positions in a sequence. To add the positional information to the encoded log messages authors usually use sine and cosine functions for even and odd token indices respectively [24], [76].

One-Hot Encoding (OH) is one of the most common techniques to handle categorical data and is therefore frequently applied on event types (as a finite number of log keys is defined in the parser) or token values. Formally, the one-hot encoding of a value i from an ordered list of d values is a vector of length d where the i-th element is 1 and all others are 0 [20]. While most authors use one-hot encoded data directly as an input to neural networks, it is also possible to combine it with other features such as count vectors so that the applied neural network is capable of discerning the input and learn separate models for different log keys [49].

Embedding Layers/Matrices (EL) are typically used to resolve the problems with respect to sparsity of high-dimensional input data such as one-hot encoded event types when a large number of log keys are required to parse the logs [25], [62]. They are usually randomly initialized parameters which are trained alongside the classification models to create optimal vector encodings for log messages [23]. The vector encodings are usually arranged in a matrix so that the respective vector for a particular log key is obtained by multiplying the matrix with a one-hot encoded log key vector. The main difference to semantic vectors is that embedding layers/matrices are generally not trained towards NLP objectives, i.e., they do not aim to learn the semantics of words like Word2Vec; instead, embedding layers/matrices are only trained to minimize the loss function of the classification network. Some authors also use custom embedding models based on deep learning; we refer to their output as **Deep**

Encoded Embeddings (DE). This includes a combination of character-, event- and sequence-based embeddings [42], attention mechanisms using MLPs and CNNs [45], and token counts with label information fed into VAEs [1].

Other than aforementioned methods that operate with event types and mostly use tokens only for encoding these events in vector format, approaches that rely on **Parameter Vectors** (PV) directly use the actual values extracted from the parsed log messages. Thereby, extracted parameters that are numeric may be used for multi-variate time-series analysis with RNNs [20], [68], [78], while categorical values could be one-hot encoded and vectorized with word embedding methods [44]. Either way, as different log events have varying numbers of parameters with different semantic meaning, it is usually necessary to analyze the parameters of each event type on their own [20], [68], [78]. The time stamp of log events is a special parameter as it allows to put other parameters in temporal context, which is required for time-series analysis. However, the time stamps themselves may be used for **Time Embedding** (TE) and serve as input to neural networks [27]. For this purpose, Li et al. [46] generate vectors for sequences of time differences between event occurrences by applying soft one-hot encoding.

While aforementioned representations are used in different variations in several publications, **Graphs** (G) are an entirely different approach that is only employed by Wan et al. [59]. The key idea is to transform event sequences into session graphs and then apply neural networks that are specifically designed for these data structures. Another less common strategy for encoding dependencies between log messages is the so-called **Transfer Matrix** (TM). In particular, the $d \times d$-dimensional matrix encodes the probabilities that any of the d log keys follows another [74].

D. Anomaly Detection Techniques

The previous sections described methods to prepare the input log data for ingestion by neural networks. However, in many cases it is non-trivial to retrieve the information whether a data sample presented to the neural network is anomalous or not as there are no dedicated output nodes for anomalies that are not known beforehand. This section therefore outlines the anomaly detection techniques applied by the reviewed approaches. First, we state different types of anomalies that are commonly targeted by approaches. Second, we investigate how the detection or classification result is retrieved from the network output. Finally, we state different decision criteria that are used to differentiate normal from anomalous samples based on these resulting measures.

1) Anomaly types: The reviewed approaches address different types of anomalies as outlined in Sect. II-A3 (AD-1). **Outliers** (OUT) are single log events that do not fit to the overall structure of the data set. Most commonly, outlier events are detected based on their unusual parameter values [43], token sequences [60], [64], or occurrence times [27]. Comparatively few approaches consider outliers since the majority of reviewed approaches focus on collective anomalies, in particular, involving sequences of events.

Sequential (SEQ) anomalies are detected when execution paths change, i.e., the applications that generate logs execute events differently than before. This could result in additional, missing, or differently ordered events within otherwise normal event sequences as well as completely new sequences that could even involve previously unseen event types. A common method to detect these anomalies is to check whether a specific event type in a sequence of events is expected to occur given all the events that occur before or thereafter.

While the detection of sequential anomalies inherently assumes that events occur as ordered sequences, **frequency** (FREQ) anomalies only consider the number of event occurrences. Nonetheless, event frequencies may be used to infer dependencies between events, for example, the numbers of events related to opening and closing files should be the same as every file will eventually be closed and needs to be opened before doing so [22]. The main idea for detecting frequency anomalies is that changes of system behavior affect the number of usual event occurrences that are most often counted within time windows.

Some approaches also consider anomalies that base on certain quantitatively expressed properties of multiple log events that go beyond event counts, such as their inter-arrival times [46] or seasonal occurrence patterns [77]. We refer to them as **statistical** (STAT) anomalies, because their detection generally assumes that the event occurrences follow specific stable distributions over time. The following section describes how the output of the neural networks is used to determine the aforementioned types of anomalies.

2) Network output: In general, the output of a neural network consists either of a single node or multiple nodes in its final layer (AD-2). Accordingly, the resulting value extracted from the network is a scalar or vector of numeric values. One possibility is to consider these results as an anomaly score that expresses to what degree the log events presented to the network represent an anomaly or not. As these scores are generally difficult to interpret on their own, it is usually necessary to compare them with some threshold. In **binary classification** (BIN) this idea is used to estimate whether the input presented to the neural network is either normal or anomalous. For supervised approaches the numeric output can be interpreted as probabilities that the input corresponds to either class. On the other hand, anomaly scores that are generated by semi- or unsupervised approaches are generally not normalized, e.g., the distance between the input and the center of a hyper-spherical cluster can become arbitrarily large, and therefore need to be compared to empirically determined thresholds. Similarly, **Input vector transformations** (TRA) that transform the input into a new vector space and generate clusters for normal data are capable of detecting outliers by their large distances to cluster centers. Another related method is to leverage the **reconstruction error** (RE) of Autoencoders that first encode the input data in a lower dimensional space and then attempt to reconstruct them to their original form. In this case, input samples are considered anomalous if they are difficult to reconstruct, i.e., yield a large reconstruction error, because they do not correspond to the normal data that the network was trained on.

While aforementioned approaches pursue binary classification that separates normal from anomalous input, there are also concepts that are capable of differentiating between more than two classes. **Multi-class classification** (MC) assigns distinct labels to specific types of anomalies but requires supervised learning in order to capture the patterns specific to these classes in the training phase. To resolve this issue, it is also possible to consider event types as the target of classification. The most common approach for this is to train the models to predict the next log key following a sequence of observed log events. When a softmax function is used as an activation for the output, this prediction yields a **probability distribution** (PRD) with the individual probabilities for each log key. The problem of predicting the next log event in a sequence can also be formulated as a regression task when events are considered as **numeric vectors** (VEC), in particular, semantic or parameter vectors. Thereby, the neural network outputs a vector representing the expected event vector instead of the respective event type.

3) Detection method: The different strategies for obtaining the network output described in the previous section already give a rough idea on the methods used to differentiate normal from anomalous behavior and eventually report anomalies (AD-3). When the network output directly corresponds to a particular **label** (LAB), for example, as accomplished by binary classification, it is simple to generate anomalies for all samples that are labeled as anomalous. For all approaches that output some kind of numeric value or anomaly score it is straightforward to use a **threshold** (THR) for differentiation. This threshold is also useful to tune the detection performance of the approach and find an acceptable tradeoff between TPR and FPR by empirical experimentation. Another approach is to model the anomaly scores obtained from the network as statistical distributions. In particular, Du et al. [20] and Xie et al. [68] use a Gaussian distribution to detect parameter vectors with errors outside of specific confidence intervals as anomalous. A different approach is proposed by Otomo et al. [49], who apply clustering on the reconstruction errors and detect all samples that are sufficiently far away from the normal clusters as anomalies.

When the output of the neural network is a multi-class probability distribution for known log keys it is common to consider the top n log keys with the **highest probabilities** (TOP) as candidates for classification. Thereby, an anomaly is only detected if the actual type of the log event is not within the set of candidates. The number of considered candidates n regulates the tradeoff between TPR and FPR analogous to the aforementioned threshold. Figure 4 shows how the network output relates to the applied detection techniques. Both BIN and MC rely on supervised learning and are therefore able to directly assign labels to new input samples. All other techniques enable semi- or unsupervised training. In particular, RE, TRA, and VEC produce anomaly scores that are compared against thresholds, while PRD is typically compared against the top log keys with highest probabilities. Note that there are some exceptions to this overall pattern. For example, the approach by Yang et al. [69] supports semi-supervised training through the use of probabilistic labels, and the approaches by

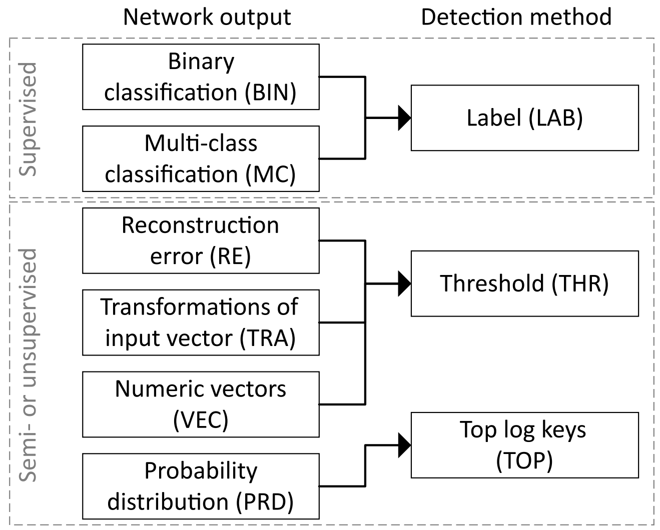

Network output
Detection method
Supervised
Binary classification (BIN)
Multi-class classification (MC)
Label (LAB)
Semi- or unsupervised
Reconstruction error (RE)
Transformations of input vector (TRA)
Numeric vectors (VEC)
Probability distribution (PRD)
Threshold (THR)
Top log keys (TOP)

Fig. 4. Most common combinations of network output types and detection techniques.

Zhang et al. [17] and Syngal et al. [57] are supervised despite relying on reconstruction errors.

E. Evaluation & Reproducibility

This section provides an overview of evaluations carried out in the reviewed publications. We present a list of openly available data sets that are useful for evaluations and state commonly used evaluation metrics and benchmarks. We also discuss reproducibility of evaluations with respect to the availability of open-source implementations.

1) Data sets: Data sets are essential in scientific publications to validate the approach and show improvements over state-of-the-art detection rates (ER-1). Our literature review reveals that there are only few data sets that are commonly used when evaluating log data anomaly detection approaches using deep learning. Table III shows an overview of all data sets used in the reviewed publications. As visible in the table, the vast majority of evaluations rely on only four data sets (HDFS, BGL, Thunderbird, and OpenStack). In the following, we briefly describe each data set.

The **HDFS** log data set stems from the Hadoop Distributed File System (HDFS) running on a high-performance computing cluster with 203 nodes that computes many standard MapReduce jobs. More than 24 million logs are collected over a period of two days. The data set comprises sequences of heterogeneous log events for specific file blocks that act as identifiers for sessions. Some of the event sequences correspond to anomalous execution paths that are mostly related to performance issues such as write exceptions, which were manually labeled [98]. The sample logs shown in Fig. 3 are simplified versions of the logs from the HDFS data set.

The **BGL** data set comprises more than four million log events that were collected over more than 200 days from a BlueGene/L (BGL) supercomputer running at the Lawrence Livermore National Labs. The log events were generated with a severity field that allows to separate them into classes;

TABLE III
COMMON PUBLIC LOG DATA SETS

Data set	Year	Use-case	Labels	Sessions	Anomaly sources	Used in evaluation
HDFS [18]	2009	High-perf. comp.	✓	✓	Failures	[1], [17], [20], [22], [23], [25], [27], [29]–[31], [33], [37]–[40], [42], [44], [46]–[48], [51], [52], [55], [56], [58], [59], [61], [62], [65]–[74], [76]–[78]
BlueGene/L (BGL) [89]	2007	High-perf. comp.	✓	-	Failures	[1], [20], [22], [24], [25], [28], [30], [33]–[37], [39]–[44], [46]–[48], [54], [55], [59], [61], [62], [64]–[66], [69], [73], [74]
Thunderbird [89]	2007	High-perf. comp.	✓	-	Failures	[24], [34]–[36], [38]–[41], [55], [60], [64], [71], [73]
OpenStack [20]	2017	Virtual machines	✓	✓	Failures	[20], [26], [34]–[36], [42], [44], [50], [52], [63], [76]–[78]
Hadoop [90]	2016	High-perf. comp.	✓	✓	Failures	[28], [31], [42], [54]
Spirit [89]	2007	High-perf. comp.	✓	-	Failures	[24], [64]
Digital Corpora [91]	2009	OS logs	-	-	Failures	[53], [54]
DFRWS 2009 [92]	2009	OS logs	-	-	Exfiltration	[53], [54]
Honeynet 2011 [93]	2011	OS logs	-	-	Intrusions	[53], [54]
Windows [94]	2020	OS logs	-	-	-	[53], [54]
Linux Security Logs [95]	2020	OS logs	-	-	Intrusions	[53]
Honeynet 2010 [96]	2010	OS logs	-	-	Intrusions	[54]
Android [94]	2020	Mobile OS logs	-	-	-	[46]
HPC RAS [97]	2011	High-perf. comp.	-	✓	Failures	[24]
Spark [94]	2020	High-perf. comp.	-	-	Failures	[54]
Zookeeper [94]	2007	High-perf. comp.	-	-	Failures	[54]

however, the logs were additionally labeled manually by system administrators. The anomalies occurring in the logs correspond to both hardware and software problems. Other log data sets from the same family that are also presented in the study by Oliner et al. [89] are **Thunderbird** and **Spirit**. These data sets comprise system logs (syslog) and similarly comprise alerts related to system problems such as disk failures. The events in both data sets include automatically generated alert tags that can be used as labels. The **HPC RAS** data set [97] is another log data set from the same family of supercomputers. Reliability, Availability, and Serviceability (RAS) logs are usually used to understand the reasons for system failure. However, this log data set does not include labels for anomalies.

The **OpenStack** data set was collected from an OpenStack platform where automatic scripts continuously and randomly carry out tasks related to handling of virtual machines, including creation, pausing, deletion, etc. Other than aforementioned data sets that mostly comprise randomly occurring failures, the authors of the OpenStack data set purposefully injected anomalies at specific points in time, including timeouts and errors. The events include instance identifiers that can be used to identify sessions [20].

Similar to the HDFS data, the **Hadoop** data set comprises logs from a computing cluster that runs the MapReduce jobs WordCount and PageRank. After an initial training phase, the authors purposefully trigger failures on the nodes, in particular, by shutting down a machine, disconnecting the network, and filling up the hard disk of one server. The logs are separated into different files according to application identifiers that act similar to session identifiers and are also used by the authors to assign labels to anomalous program executions [90].

Loghub [94] comprises several data sets that are used in evaluations, including logs from high-performance computing systems. The **Spark** data set contains logs from the distributed data processing engine Apache Spark running on 32 machines. The logs include normal and anomalous executions, but are not labeled. **ZooKeeper** is another Apache service used in distributed computing and configuration management. Loghub also comprises logs from conventional operating systems. The **Windows** log data set was collected on a laboratory Windows 7 machine by monitoring CBS (Component Based Servicing), which operates on a package and update level. Similarly, the **Android** data set was collected from a mobile phone in a laboratory setting. Both data sets are not labeled and do not involve any purposefully injected anomalies. Logs from the Linux operating system are provided in the disk images of the **Digital Corpora**, where failures such as invalid authentications occur in the data [91].

When it comes to the detection of anomalous behavior, all aforementioned data sets mainly provide failure events that are generated as part of legitimate system usage. However, there are also data sets that instead involve events generated from malicious activities and thus enable evaluation of anomaly-based intrusion detection systems. For example, the **DFRWS 2009** data set contains system logs from Linux devices that involve data exfiltration, unauthorized accesses, as well as the use of backdoor software [92]. The **Honeynet 2010** [96] and **Honeynet 2011** [93] data sets comprise common log files from compromised Linux machines that were illegitimately accessed. The Public Security Log Sharing Site [95] provides **Linux Security Logs** from diverse sources and affected by different real-world intrusions, such as brute-force attacks. Unfortunately, none of these security log files come with labels for malicious events and thus authors need to generate their own ground truths to evaluate their approaches on these data sets [53].

2) Evaluation metrics: Quantitative evaluation (ER-2) of anomaly detection approaches typically revolves around counting the numbers of correctly detected anomalous samples as true positives (TP), incorrectly detected non-anomalous samples as false positives (FP), incorrectly undetected anomalous samples as false negatives (FN), and correctly undetected non-anomalous samples as true negatives (TN). In the most basic setting where events are labeled individually and samples represent single events (e.g., as in the BGL data set), it is relatively straightforward to evaluate detected events with binary classification [34], [36]. Some of the reviewed papers additionally consider a multi-class classification problem for data sets where different types of failures have distinct labels by computing the averages of evaluation metrics over all classes [55] or plotting confusion matrices [32].

Given that log events are sometimes aggregated with diverse methods prior to detection, it stands to reason that there are different ways to determine whether a sample is anomalous or not, and whether it counts as a correct detection or not. For example, aggregation of logs in windows could require to count detected events as true positives as long as they are close enough to the actual anomaly in the event sequence [26], [57]. Since a majority of the reviewed papers rely on the HDFS data set where labels are only available for whole event sessions rather than single events, the most common method to compute aforementioned metrics relies on counting of in-/correctly identified non-/anomalous sessions [30], [33], [44], [45], [62].

Regardless of how the positive and negative samples are counted, almost all authors eventually evaluate their approaches using the well-known metrics precision ($P = \frac{TP}{TP+FP}$), recall or true positive rate ($R = TPR = \frac{TP}{TP+FN}$), false positive rate ($FPR = \frac{FP}{FP+TN}$) and F1-score ($F1 = \frac{2 \cdot P \cdot R}{P+R}$). Less common evaluation metrics are the accuracy ($ACC = \frac{TP+TN}{TP+TN+FP+FN}$) used in 15 publications as well as the area under curve which is computed for precision-recall-curves [31] and receiver operator characteristic (ROC) curves [43], [60]. Other metrics that are more specific to deep learning applications are the number of model parameters [38], [61] and time to train models or run the detection (ER-3) [29], [32], [37], [47], [52], [68]. Some authors also assess characteristics of their approaches that go beyond standard anomaly detection evaluations, for example, whether training on combinations of multiple data sets improves the overall performance of classification [42] or whether their approaches are robust against changes of log patterns over time [17], [42], [44].

3) Benchmark approaches: Most publications compare the evaluation metrics stated in the previous section with benchmark approaches to show their improvements over the state-of-the-art (ER-4). Figure 5 shows the most commonly used benchmarks in the reviewed publications. Note that we only

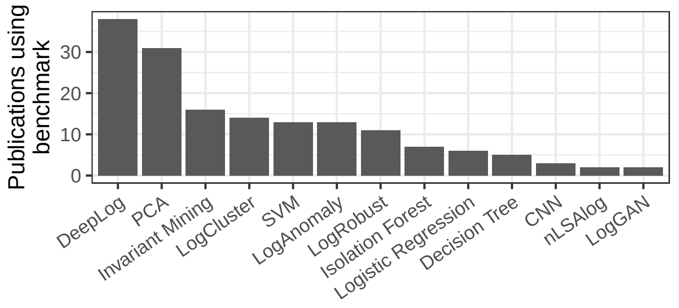


Fig. 5. Overview of common benchmark approaches used in evaluations.

considered approaches that appear in at least two different publications for this visualization.

As visible in the figure, **DeepLog** [20] is the most commonly used benchmark with appearances in 38 out of the 61 reviewed publications. DeepLog relies on LSTM RNNs to predict upcoming log events in sequences and thereby disclose observed events as anomalies if they are expected to occur with low probabilities. The popularity of DeepLog for comparisons can be explained by the facts that DeepLog was the first to detect sequential anomalies in log data using deep learning (cf. Sect. IV-A2) and that open-source re-implementations are available online.

The second most commonly used benchmark leverages principal component analysis (**PCA**) and relies on event counts rather than sequences. In particular, Xu et al. [98] create message count vectors for event type identifiers and use PCA to transform them into subspaces where anomalies appear as outliers that have a high distance to all other samples. Lou et al. [99] also generate message count vectors but use **Invariant Mining** to discover linear relationships between log events that represent execution workflows. Event sequences that violate previously identified invariants are declared as anomalies.

LogCluster [90] generates vectors for log sequences and then clusters them using a similarity metric. The approach is mainly designed for log filtering but can also be applied for detection of unusual log patterns. Support vector machines (**SVM**) [100] are yet another method relying on event count vectors. However, other than PCA and invariant mining, SVM typically operate in supervised manner. To alleviate this issue and enable application in semi- or unsupervised cases, authors therefore resort to one-class SVM [101] or Support Vector Data Description (SVDD) [102].

Similar to DeepLog and contrary to aforementioned conventional machine learning approaches, **LogAnomaly** [22] leverages LSTM RNNs to analyze log event sequences. To this end, they propose the so-called template2Vec embedding method to extract semantic vectors from the tokens that make up the events. **LogRobust** [17] also makes use of semantic vectors but is specifically designed to handle unknown log events that appear as part of software evolution.

Isolation Forest [103] is an anomaly detection technique where the analyzed data is recursively split until a single data instance is isolated from all other points; thereby, anomalous points are identified as they are expected to require fewer splits until their isolation. **Logistic Regression** [104] is a classifier for numeric input data that works best in linear classification cases [53]. **Decision Tree** [105] on the other hand is a supervised classification method where instances traverse a binary search tree. Thereby, each internal node splits the data by a specific predicate and each leaf of the tree determines the class of the instances.

Another benchmark that relies on deep learning are **CNN**. In particular, the approach by Lu et al. [23] semantically encodes sequences of event identifiers and embeds them into a matrix for convolution. Finally, **nLSAlog** [25] leverages LSTM RNNs and is thus similar to DeepLog. Out of all reviewed publications, only eight do not involve any benchmark approach for comparison.

4) Reproducibility: Authors of scientific publications should pursue to enable reproducibility of their presented results for many reasons, including the possibility for others to validate the correctness of the approach, to extend the algorithms with additional features, to carry out their own experiments on other data sets, and to use the approach as benchmarks in new publications. We consider an approach reproducible when both the data used in the evaluation (ER-5) as well as the original source code (ER-6) are publicly available.

As outlined in Sect. IV-E1, a majority of the reviewed publications carry out their evaluations on the same few data sets that are publicly available. Some authors also evaluate their approaches on private data sets that are synthetically generated in testbeds [21], [26], [45], [56], [62], [63], [75], collected from academic institutions [49], [69], or obtained from industrial real-world applications [17], [28], [31], [32], [57], [64], [69]. Overall, 54 out of the 61 reviewed publications involve evaluations on at least one of the publicly available data sets from Table III.

While it is relatively common to evaluate approaches on public data sets, there are unfortunately only few authors that publish implementations of their approaches alongside the papers. During our review we were only able to find the original source code of presented approaches from 8 publications. However, we also point out that there exist some re-implementations of scientific approaches in the deep-loglizer toolbox provided by Chen et al. [9]. We mark approaches where implementations by the original authors exist with (YES), re-implementations by other authors as (RE), and all others as (NO) in Table IV. We encourage authors to publish their code to improve the reproducibility of their results and hope to see more open-source implementations in the future.

V. Discussion

The previous sections presented the results of our survey in detail. In the following we summarize these results, discuss open issues in the research area on log-based anomaly detection using deep learning, and propose ideas for future research in course of answering our research questions from Sect. I.

***RQ1**: What are the main challenges of log-based anomaly detection with deep learning?* When carrying out our systematic literature review we assessed whether and to what degree the current state-of-the-art addresses the research challenges enumerated in Sect. II-B. It turns out that data instability, i.e., the appearance of previously unknown events, is one of the main issues addressed by the reviewed approaches. The key idea to resolving this problem is currently to represent logs as semantic vectors so that new or changed events can still be compared to known events by measuring their similarities [17], [28], [41], [46], [48], [50], [57], [65], [69], [73]. There are many techniques for generating numeric vectors to represent log events (cf. Sect. IV-C4) and thus resolve the issue of feeding unstructured and textual input data to neural networks. Imbalanced data sets are another challenge that is specifically addressed by some approaches. In particular, authors suggest to use sampling techniques as well as context-aware embedding methods as possible solutions [35], [45], [55], [62], [66].

Some approaches are specifically designed to enable applicability in scenarios that demand efficient and lightweight algorithms, e.g., deployment on edge devices. This is achieved by leveraging low-dimensional vector representations as well as convolutional neural networks that are more efficient than recursive neural networks [29], [38], [61]. Similarly, some approaches support log stream processing and enable adaptive learning (i.e., dynamically changing the baselines for anomaly detection) by incrementally re-training the models with manually identified and labeled false positive samples [70]. It must be noted that there is currently no solution how to automatically determine that re-training is required without label information or manual intervention. The challenge of interleaving logs is generally solved by leveraging session identifiers directly from parsed log data; however, there are also approaches that evade the need for sequences altogether, e.g., by relying on sentiment analysis [75].

A way to address the challenge that only few labeled data is available is provided by transfer learning, where models are trained on one system and tested on another [31], [40]. The main idea is that the log event patterns learned by the neural networks are similar across different domains and that already seen anomalies can be recognized and classified. Guo et al. [38] are the only authors to consider federated learning, where learning takes place in a distributed manner across multiple systems. Hashemi et al. [42] also go into this direction as they combine multiple data sets to evaluate whether this affects the performance of their model. We believe that federated learning could be an interesting topic for future publications as there exist many real-world scenarios where log data is monitored in distributed machines but orchestration of deployed detectors takes place centrally [106].

The challenge of facing diverse artifacts of anomalies is only partially addressed since the vast majority of approaches focus on sequences and frequencies of log events, but only few consider event parameters or inter-arrival times for detection. We recommend to also consider techniques that address other patterns that appear in normal system behavior and may be useful to detect specific anomalies, such as changes of parameter value correlations, periodic behavior, statistical distributions, etc. Moreover, we observed that the explainability of the proposed deep learning models is relatively low, i.e., it is non-trivial to understand the criteria for classifications and thus interpretation of false positives as well as false negatives is generally difficult. This hinders root cause analysis of detected anomalies and produces an overhead for system operators. Specifically in security-critical systems (e.g., intrusion detection) it is vital to understand the functioning - and thus the limits - of deployed anomaly detectors. We would therefore recommend to direct future research in log-based anomaly detection towards explainable artificial intelligence [107].

RQ2*: What state-of-the-art deep learning algorithms are typically applied?* Our review shows that diverse types of deep learning algorithms are used in scientific publications and that it is common to combine several approaches. Thereby, RNNs are clearly the most applied models, because they are a natural choice for capturing sequential patterns in log data. CNNs are used as an efficient alternative to RNNs as they are also able to pick up event dependencies. On the other hand, Autoencoders and Transformers are frequently applied as they support unsupervised learning. While GANs, MLPs, GNNs, and EGNNs are only used by few approaches, they have beneficial properties (cf. Sect. IV-B1) that make these models worth considering. Similarly, we are convinced that other deep learning architectures that are not explored in the reviewed literature could yield interesting insights, e.g., deep belief networks or deep reinforcement learning [13], [79].

We believe that the application of specific techniques is mostly motivated by the log data to be analyzed and anomalies to be detected. The fact that all of the commonly used log data sets involve anomalies that manifest as sequentially occurring events (cf. Sect. IV-E1) thus explains the tendency towards RNNs. However, as anomalies could also manifest as point or contextual anomalies (cf. Sect. II-A3) we recommend to consider alternative log data sets with different types of anomalies and to develop approaches for these cases. For example, in our earlier works [108], [109] we published log data sets where anomalies affect combinations, compositions, and distributions of event parameter values in addition to frequencies and sequences of log events.

RQ3*: How is log data pre-processed to be ingested by deep learning models?* Our survey shows that there are three main ways to approach feature extraction from raw log data. First, by tokenizing log messages, which is a simple method that does not require any parsers but lacks semantic interpretation of the tokens. Second, by parsing the messages and extracting information from collections of log events, such as sequences, counts, or statistics. Third, by extracting parameters including the time stamps from parsed log events. There are a multitude of methods to represent these features as numeric vectors to be ingested by deep learning models.

While traditional machine learning methods such as SVM or PCA work best with event count vectors, most approaches leveraging deep learning neural networks use semantic vectors to yield the best results [1]. Thereby, the tokens that make up the log messages are represented as numeric vectors and considered in the context of their sequence of event occurrences. Most approaches employ these sequential features, while frequencies, one-hot encoded data, and embedding layers are used less often or only as a contributing feature.

RQ4*: What types of anomalies are detected and how are they identified as such?* Almost all reviewed approaches focus on sequential anomalies that either manifest in the sequences of events, the sequences of tokens within events, or a combination of both. Only few approaches make use of event counts or detect single log lines as outliers without their context of occurrence. The detection technique is generally driven by the output of the neural networks. While binary or multi-class classifications are directly used to report anomalies, all numeric outputs such as anomaly scores or reconstruction errors are compared against pre-defined thresholds and probability distributions of log events are used to check whether the actual events is within the top candidates. Determining these thresholds is usually carried out empirically for a particular log file.

RQ5*: How are the proposed models evaluated?* Our survey on commonly used log data sets from Sect. IV-E1 shows that there are only four data sets that are used by a majority of the approaches: HDFS, BGL, Thunderbird, and OpenStack. All these data sets are collected from scenarios involving high-performance computing and virtual machines that are affected by randomly occurring failures. It is obvious that the availability of anomaly labels make these data sets particularly attractive for scientific evaluations. As mentioned before, we argue that a larger and more diverse set of input data sets would be beneficial to evaluate whether the proposed approaches are capable of detecting anomalous artifacts other than unusual sequences. In particular, the consequences of cyber attacks rather than failures could result in log artifacts that are suitable for detection and an appropriate use-case for anomaly detection.

We manually analyzed the HDFS data set as it is the most popular of all and found that it is far from challenging to achieve competitive detection rates. The reason for this is that many of the anomalous event sequences are trivial to identify as they involve event types that never occur in the training data or involve fewer elements than the shortest normal sequences. Using these two heuristics we are able to achieve $F1 = 90.41\%$, $ACC = 99.48\%$, $P = 98.37\%$, $R = 83.65\%$, $FPR = 0.04\%$, $TNR = 99.96\%$ on the test data. Moreover, using the sequence length heuristic in combination with the simple Stide algorithm [110] that moves a sliding window of a given length over the data and looks for sub-sequences that have not been seen before in the training data further improves the evaluation metrics to $F1 = 95.14\%$, $ACC = 99.71\%$, $P = 95.27\%$, $R = 95.01\%$, $FPR = 0.14\%$, $TNR = 99.86\%$ when using a window size of 2. DeepLog only yields slightly better detection scores of $F1 = 95.72\%$, $ACC = 99.75\%$, $P = 95.12\%$, $R = 96.32\%$, $FPR = 0.15\%$, $TNR = 99.85\%$ [20]. We provide the code for both experiments (heuristics and Stide) in a reproducible form as an open-source implementation[10]. It is not clear to us why Stide was not used as a benchmark in any of the reviewed publications: Stide only takes a fraction of the time for training and processing the test data in comparison to deep (and also most conventional) learning models, and additionally has a much better explainability than neural networks as the resulting model simply consists of a set of sequences. We

[10]https://github.com/ait-aecid/stide

argue that the fact that such a simple algorithm achieves competitive detection rates to deep learning models further urges authors to consider additional data sets where more diverse anomalous artifacts are present and the benefits of their approaches such as robustness against data instability become apparent.

Another issue that we noticed is that evaluations are usually carried out on the basis of anomalous sequences, i.e., the whole sequence is considered normal or anomalous rather than its elements [20]. However, parts of long sequences may actually represent normal system behavior while only a few elements should be considered anomalies. It would be interesting to evaluate whether detection approaches are able to pinpoint exactly which parts of sequences are anomalous, which would also be practical for manual investigations of reported anomalies by system operators. Obviously this requires that data sets are labeled on the granularity of single events rather than sessions.

Finally, we note that almost all evaluations rely on metrics such as the F-score (cf. Sect. IV-E2) that are known to not accurately depict the classification or detection performance when data sets are highly imbalanced. To avoid misinterpretations of evaluation results, it is recommended to also compute metrics that are more robust against class imbalance, such as the specificity or true negative rate [10].

RQ6*: To what extent do the approaches rely on labeled data and support incremental learning?* Log-based anomaly detection is most often applied in use-cases that aim to disclose unexpected system behavior such as failures or cyber attacks. Since these artifacts are not known beforehand and thus no labels can exist, un- or semi-supervised approaches are generally more widely applicable and therefore preferable [7]. Since semi-supervised learning can be achieved with most neural network architectures including RNNs, but only specific deep learning models support fully unsupervised operation, we find 28 semi-supervised approaches in our reviewed literature as opposed to only 8 out of 61 approaches that are unsupervised. We did not expect to see the relatively large amount of 25 supervised approaches that require at least partially labeled anomalies for training. Moreover, with 53 out of 61 a vast majority of approaches only support offline training. This includes most supervised models and also all other approaches that do not intend to dynamically and automatically update the trained models over time. Only 8 of the reviewed approaches enable continuous model adjustments through re-training or EGNN model architectures [32].

RQ7*: How reproducible are the presented results in terms of availability of source code and used data?* As pointed out in Sect. IV-E4, a majority of the reviewed approaches make use of publicly available data sets to evaluate their approaches and only 7 publications rely on private data sets. The reproducibility of the presented results is thus relatively high, assuming that readers are willing to re-implement the approaches based on the descriptions from the papers from scratch. We could only find 8 publicly available source codes of approaches published by the original authors as well as 4 re-implementations, indicating a low reproducibility overall. We encourage authors to publish reproducible experiments in the future to also enable large-scale quantitative comparisons in surveys.

VI. Conclusion

This paper presents a survey of 61 scientific approaches that pursue the detection of anomalous events or processes in system log data using deep learning. The survey shows that diverse model architectures are suitable for this purpose, including models for sequential input data such as recursive or convolutional neural networks, language-based models such as transformers, as well as unsupervised models such as Autoencoders or generative adversarial networks. Similarly, there are different features used for training and subsequently for detection, such as sequences and counts of events or tokens as well as parameter values or statistics derived from the events. To enable processing of these features as input to neural networks it is necessary to encode them as numeric vectors, for example, through semantic vectorization or one-hot encoding. Anomalies are then detected either directly through classification or by deriving some kind of anomaly score from the network that allows to discern normal from anomalous system behavior. The survey shows that there are open challenges that are not sufficiently resolved by existing approaches, including detection techniques that go beyond sequential anomalies, low explainability of trained models and classification results, lack of representative evaluation data sets containing diverse attack artifacts, and a low reproducibility.

Acknowledgments

This work was partly funded by the FFG project DECEPT (873980) and the EU project PANDORA (SI2.835928).

References

[1] Q. Wang, X. Zhang, X. Wang, and Z. Cao, "Log sequence anomaly detection method based on contrastive adversarial training and dual feature extraction," *Entropy*, vol. 24, no. 1, p. 69, 2021.

[2] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16–24, 2013.

[3] M. Landauer, F. Skopik, M. Wurzenberger, and A. Rauber, "System log clustering approaches for cyber security applications: A survey," *Computers & Security*, vol. 92, p. 101739, 2020.

[4] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: System log analysis for anomaly detection," in *2016 IEEE 27th international symposium on software reliability engineering (ISSRE)*. IEEE, 2016, pp. 207–218.

[5] C. Kruegel and G. Vigna, "Anomaly detection of web-based attacks," in *Proceedings of the 10th ACM conference on Computer and communications security*, 2003, pp. 251–261.

[6] M. Landauer, M. Wurzenberger, F. Skopik, G. Settanni, and P. Filzmoser, "Dynamic log file analysis: An unsupervised cluster evolution approach for anomaly detection," *computers & security*, vol. 79, pp. 94–116, 2018.

[7] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

[8] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[9] Z. Chen, J. Liu, W. Gu, Y. Su, and M. R. Lyu, "Experience report: Deep learning-based system log analysis for anomaly detection," *arXiv preprint arXiv:2107.05908*, 2021.

[10] V. H. Le and H. Zhang, "Log-based anomaly detection with deep learning: How far are we?" *arXiv preprint arXiv:2202.04301*, 2022.

[11] R. B. Yadav, P. S. Kumar, and S. V. Dhavale, "A survey on log anomaly detection using deep learning," in *2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*. IEEE, 2020, pp. 1215–1220.

[12] D. Kwon, H. Kim, J. Kim, S. C. Suh, I. Kim, and K. J. Kim, "A survey of deep learning-based network anomaly detection," *Cluster Computing*, vol. 22, no. 1, pp. 949–961, 2019.

[13] I. H. Sarker, "Deep learning: a comprehensive overview on techniques, taxonomy, applications and research directions," *SN Computer Science*, vol. 2, no. 6, pp. 1–20, 2021.

[14] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2019, pp. 121–130.

[15] L. Bao, Q. Li, P. Lu, J. Lu, T. Ruan, and K. Zhang, "Execution anomaly detection in large-scale systems through console log analysis," *Journal of Systems and Software*, vol. 143, pp. 172–186, 2018.

[16] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A survey," *arXiv preprint arXiv:1901.03407*, 2019.

[17] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li *et al.*, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 807–817.

[18] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, 2009, pp. 117–132.

[19] H. Mi, H. Wang, Y. Zhou, M. R.-T. Lyu, and H. Cai, "Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1245–1255, 2013.

[20] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 1285–1298.

[21] T. Yang and V. Agrawal, "Log file anomaly detection," *CS224d Fall*, vol. 2016, 2016.

[22] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun *et al.*, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs," in *IJCAI*, vol. 19, no. 7, 2019, pp. 4739–4745.

[23] S. Lu, X. Wei, Y. Li, and L. Wang, "Detecting anomaly in big data system logs using convolutional neural network," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. IEEE, 2018, pp. 151–158.

[24] S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso, and O. Kao, "Self-attentive classification-based anomaly detection in unstructured logs," in *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2020, pp. 1196–1201.

[25] R. Yang, D. Qu, Y. Gao, Y. Qian, and Y. Tang, "Nlsalog: An anomaly detection framework for log sequence in security management," *IEEE Access*, vol. 7, pp. 181 152–181 164, 2019.

[26] X. Baril, O. Coustié, J. Mothe, and O. Teste, "Application performance anomaly detection with lstm on temporal irregularities in logs," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 1961–1964.

[27] S. Bursic, V. Cuculo, and A. D'Amelio, "Anomaly detection from log files using unsupervised deep learning," in *International Symposium on Formal Methods*. Springer, 2019, pp. 200–207.

[28] M. Catillo, A. Pecchia, and U. Villano, "Autolog: Anomaly detection by deep autoencoding of system logs," *Expert Systems with Applications*, vol. 191, p. 116263, 2022.

[29] P. Cheansunan and P. Phunchongharn, "Detecting anomalous events on distributed systems using convolutional neural networks," in *2019 IEEE 10th International Conference on Awareness Science and Technology (iCAST)*. IEEE, 2019, pp. 1–5.

[30] H. Chen, R. Xiao, and S. Jin, "Unsupervised anomaly detection based on system logs," in *Proceedings of the 33rd International Conference on Software Engineering & Knowledge Engineering (SEKE 2021)*, 2021, pp. 92–97.

[31] R. Chen, S. Zhang, D. Li, Y. Zhang, F. Guo, W. Meng, D. Pei, Y. Zhang, X. Chen, and Y. Liu, "Logtransfer: Cross-system log anomaly detection for software systems with transfer learning," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2020, pp. 37–47.

[32] L. Decker, D. Leite, F. Viola, and D. Bonacorsi, "Comparison of evolving granular classifiers applied to anomaly detection for predictive maintenance in computing centers," in *2020 IEEE Conference on Evolving and Adaptive Intelligent Systems (EAIS)*. IEEE, 2020, pp. 1–8.

[33] Q. Du, L. Zhao, J. Xu, Y. Han, and S. Zhang, "Log-based anomaly detection with multi-head scaled dot-product attention mechanism," in *International Conference on Database and Expert Systems Applications*. Springer, 2021, pp. 335–347.

[34] A. Farzad and T. A. Gulliver, "Log message anomaly detection and classification using auto-b/lstm and auto-gru," *arXiv preprint arXiv:1911.08744*, 2019.

[35] A. Farzad, "Log message anomaly detection with oversampling," *International Journal of Artificial Intelligence and Applications (IJAIA)*, vol. 11, no. 4, 2020.

[36] A. Farzad and T. A. Gulliver, "Two class pruned log message anomaly detection," *SN Computer Science*, vol. 2, no. 5, pp. 1–18, 2021.

[37] S. Gu, Y. Chu, W. Zhang, P. Liu, Q. Yin, and Q. Li, "Research on system log anomaly detection combining two-way slice gru and ga-attention mechanism," in *2021 4th International Conference on Artificial Intelligence and Big Data (ICAIBD)*. IEEE, 2021, pp. 577–583.

[38] Y. Guo, Y. Wu, Y. Zhu, B. Yang, and C. Han, "Anomaly detection using distributed log data: A lightweight federated learning approach," in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–8.

[39] H. Guo, S. Yuan, and X. Wu, "Logbert: Log anomaly detection via bert," in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–8.

[40] H. Guo, X. Lin, J. Yang, Y. Zhuang, J. Bai, B. Zhang, T. Zheng, and Z. Li, "Translog: A unified transformer-based framework for log anomaly detection," *arXiv preprint arXiv:2201.00016*, 2021.

[41] X. Han and 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.

[42] S. Hashemi and M. Mäntylä, "Onelog: Towards end-to-end training in software log anomaly detection," *arXiv preprint arXiv:2104.07324*, 2021.

[43] R. Hirakawa, K. Tominaga, and Y. Nakatoh, "Software log anomaly detection through one class clustering of transformer encoder representation," in *International Conference on Human-Computer Interaction*. Springer, 2020, pp. 655–661.

[44] S. Huang, Y. Liu, C. Fung, R. He, Y. Zhao, H. Yang, and Z. Luan, "Hitanomaly: Hierarchical transformers for anomaly detection in system log," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2064–2076, 2020.

[45] H. Li and Y. Li, "Logspy: System log anomaly detection for distributed systems," in *2020 International Conference on Artificial Intelligence and Computer Engineering (ICAICE)*. IEEE, 2020, pp. 347–352.

[46] X. Li, P. Chen, L. Jing, Z. He, and G. Yu, "Swisslog: Robust and unified deep learning based log anomaly detection for diverse faults," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2020, pp. 92–103.

[47] X. Liu, W. Liu, X. Di, J. Li, B. Cai, W. Ren, and H. Yang, "Lognads: Network anomaly detection scheme based on log semantics representation," *Future Generation Computer Systems*, vol. 124, pp. 390–405, 2021.

[48] D. Lv, N. Luktarhan, and Y. Chen, "Conanomaly: Content-based anomaly detection for system logs," *Sensors*, vol. 21, no. 18, p. 6125, 2021.

[49] K. Otomo, S. Kobayashi, K. Fukuda, and H. Esaki, "Latent variable based anomaly detection in network system logs," *IEICE TRANSACTIONS on Information and Systems*, vol. 102, no. 9, pp. 1644–1652, 2019.

[50] H. Ott, J. Bogatinovski, A. Acker, S. Nedelkoski, and O. Kao, "Robust and transferable anomaly detection in log data using pre-trained language models," in *2021 IEEE/ACM International Workshop on Cloud Intelligence (CloudIntelligence)*. IEEE, 2021, pp. 19–24.

[51] A. Patil, A. Wadekar, T. Gupta, R. Vijan, and F. Kazi, "Explainable lstm model for anomaly detection in hdfs log file using layerwise relevance propagation," in *2019 IEEE Bombay Section Signature Conference (IBSSC)*. IEEE, 2019, pp. 1–6.

[52] Y. Qian, S. Ying, and B. Wang, "Anomaly detection in distributed systems via variational autoencoders," in *2020 IEEE International*

Conference on Systems, Man, and Cybernetics (SMC). IEEE, 2020, pp. 2822–2829.

[53] H. Studiawan and F. Sohel, "Anomaly detection in a forensic timeline with deep autoencoders," *Journal of Information Security and Applications*, vol. 63, p. 103002, 2021.

[54] H. Studiawan, F. Sohel, and C. Payne, "Anomaly detection in operating system logs with deep learning-based sentiment analysis," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 5, pp. 2136–2148, 2020.

[55] P. Sun, E. Yuepeng, T. Li, Y. Wu, J. Ge, J. You, and B. Wu, "Context-aware learning for anomaly detection with imbalanced log data," in *2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 2020, pp. 449–456.

[56] T. Sundqvist, M. H. Bhuyan, J. Forsman, and E. Elmroth, "Boosted ensemble learning for anomaly detection in 5g ran," in *IFIP International Conference on Artificial Intelligence Applications and Innovations*. Springer, 2020, pp. 15–30.

[57] S. Syngal, S. Verma, K. Karthik, Y. Katyal, and S. Ghosh, "Server-language processing: A semi-supervised approach to server failure detection," in *2021 2nd International Conference on Computing, Networks and Internet of Things*, 2021, pp. 1–7.

[58] A. Wadekar, T. Gupta, R. Vijan, and F. Kazi, "Hybrid cae-vae for unsupervised anomaly detection in log file systems," in *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. IEEE, 2019, pp. 1–7.

[59] Y. Wan, Y. Liu, D. Wang, and Y. Wen, "Glad-paw: Graph-based log anomaly detection by position aware weighted graph attention network," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2021, pp. 66–77.

[60] M. Wang, L. Xu, and L. Guo, "Anomaly detection of system logs based on natural language processing and deep learning," in *2018 4th International Conference on Frontiers of Signal Processing (ICFSP)*. IEEE, 2018, pp. 140–144.

[61] Z. Wang, J. Tian, H. Fang, L. Chen, and J. Qin, "Lightlog: A lightweight temporal convolutional network for log anomaly detection on the edge," *Computer Networks*, vol. 203, p. 108616, 2022.

[62] Z. Wang, Z. Chen, J. Ni, H. Liu, H. Chen, and J. Tang, "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.

[63] S. R. Wibisono and A. I. Kistijantoro, "Log anomaly detection using adaptive universal transformer," in *2019 International Conference of Advanced Informatics: Concepts, Theory and Applications (ICAICTA)*. IEEE, 2019, pp. 1–6.

[64] T. Wittkopp, A. Acker, S. Nedelkoski, J. Bogatinovski, D. Scheinert, W. Fan, and O. Kao, "A2log: attentive augmented log anomaly detection," *arXiv preprint arXiv:2109.09537*, 2021.

[65] L. Xi, Y. Xin, S. Luo, Y. Shang, and Q. Tang, "Anomaly detection mechanism based on hierarchical weights through large-scale log data," in *2021 International Conference on Computer Communication and Artificial Intelligence (CCAI)*. IEEE, 2021, pp. 106–115.

[66] B. Xia, Y. Bai, J. Yin, Y. Li, and J. Xu, "Loggan: A log-level generative adversarial network for anomaly detection using permutation event modeling," *Information Systems Frontiers*, vol. 23, no. 2, pp. 285–298, 2021.

[67] C. Xiao, J. Huang, and W. Wu, "Detecting anomalies in cluster system using hybrid deep learning model," in *International Symposium on Parallel Architectures, Algorithms and Programming*. Springer, 2019, pp. 393–404.

[68] Y. Xie, L. Ji, and X. Cheng, "An attention-based gru network for anomaly detection from system logs," *IEICE TRANSACTIONS on Information and Systems*, vol. 103, no. 8, pp. 1916–1919, 2020.

[69] L. Yang, J. Chen, Z. Wang, W. Wang, J. Jiang, X. Dong, and W. Zhang, "Semi-supervised log-based anomaly detection via probabilistic label estimation," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1448–1460.

[70] S. Yen, M. Moh, and T.-S. Moh, "Causalconvlstm: Semi-supervised log anomaly detection through sequence modeling," in *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*. IEEE, 2019, pp. 1334–1341.

[71] K. Yin, M. Yan, L. Xu, Z. Xu, Z. Li, D. Yang, and X. Zhang, "Improving log-based anomaly detection with component-aware analysis," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2020, pp. 667–671.

[72] D. Yu, X. Hou, C. Li, Q. Lv, Y. Wang, and N. Li, "Anomaly detection in unstructured logs using attention-based bi-lstm network," in *2021 7th IEEE International Conference on Network Intelligence and Digital Content (IC-NIDC)*. IEEE, 2021, pp. 403–407.

[73] L. Zhang, W. Li, Z. Zhang, Q. Lu, C. Hou, P. Hu, T. Gui, and S. Lu, "Logattn: Unsupervised log anomaly detection with an autoencoder based attention mechanism," in *International Conference on Knowledge Science, Engineering and Management*. Springer, 2021, pp. 222–235.

[74] C. Zhang, X. Wang, H. Zhang, H. Zhang, and P. Han, "Log sequence anomaly detection based on local information extraction and globally sparse transformer model," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4119–4133, 2021.

[75] D. Zhang, D. Dai, R. Han, and M. Zheng, "Sentilog: Anomaly detecting on parallel file systems via log-based sentiment analysis," in *Proceedings of the 13th ACM Workshop on Hot Topics in Storage and File Systems*, 2021, pp. 86–93.

[76] Z. Zhao, W. Niu, X. Zhang, R. Zhang, Z. Yu, and C. Huang, "Trine: Syslog anomaly detection with three transformer encoders in one generative adversarial network," *Applied Intelligence*, pp. 1–10, 2021.

[77] P. Zhou, Y. Wang, Z. Li, X. Wang, G. Tyson, and G. Xie, "Logsayer: Log pattern-driven cloud component anomaly diagnosis with machine learning," in *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*. IEEE, 2020, pp. 1–10.

[78] B. Zhu, J. Li, R. Gu, and L. Wang, "An approach to cloud platform log anomaly detection based on natural language processing and lstm," in *2020 3rd International Conference on Algorithms, Computing and Artificial Intelligence*, 2020, pp. 1–7.

[79] I. H. Sarker, "Deep cybersecurity: a comprehensive overview from neural network and deep learning perspective," *SN Computer Science*, vol. 2, no. 3, pp. 1–16, 2021.

[80] Z. Zhang and M. Sabuncu, "Generalized cross entropy loss for training deep neural networks with noisy labels," *Advances in neural information processing systems*, vol. 31, 2018.

[81] Y. Cui, S. Ahmad, and J. Hawkins, "Continuous online sequence learning with an unsupervised neural network model," *Neural computation*, vol. 28, no. 11, pp. 2474–2504, 2016.

[82] R. Hadsell, D. Rao, A. A. Rusu, and R. Pascanu, "Embracing change: Continual learning in deep neural networks," *Trends in cognitive sciences*, vol. 24, no. 12, pp. 1028–1040, 2020.

[83] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE international conference on web services (ICWS)*. IEEE, 2017, pp. 33–40.

[84] M. Du and F. Li, "Spell: Streaming parsing of system event logs," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 2016, pp. 859–864.

[85] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[86] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[87] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[88] C. Manning, P. Raghavan, and H. Schütze, "Introduction to information retrieval," *Natural Language Engineering*, vol. 16, no. 1, pp. 100–103, 2010.

[89] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *37th annual IEEE/IFIP international conference on dependable systems and networks (DSN'07)*. IEEE, 2007, pp. 575–584.

[90] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and 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.

[91] S. Garfinkel, P. Farrell, V. Roussev, and G. Dinolt, "Bringing science to digital forensics with standardized forensic corpora," *digital investigation*, vol. 6, pp. S2–S11, 2009.

[92] C. Eoghan and G. R. I. Golden, "Dfrws 2009 forensics challenge," http://old.dfrws.org/2009/challenge/index.shtml, accessed: 2022-05-13.

[93] G. Arcas, H. Gonzales, and J. Cheng, "The honeynet project 2011 challenge 7 – forensic analysis of a compromised server," https://www.honeynet.org/challenges/forensic-challenge-7-analysis-of-a-compromised-server/, accessed: 2022-05-13.

[94] S. He, J. Zhu, P. He, and M. R. Lyu, "Loghub: a large collection of system log datasets towards automated log analytics," *arXiv preprint arXiv:2008.06448*, 2020.

[95] A. Chuvakin, "Public security log sharing site," https://log-sharing.dreamhosters.com/, Nov. 2010, accessed: 2021-10-18.

[96] R. Marty, A. Chuvakin, and S. Tricaud, "The honeynet project 2010 challenge 5 – log mysteries," https://www.honeynet.org/challenges/forensic-challenge-7-analysis-of-a-compromised-server/, accessed: 2022-05-13.

[97] Z. Zheng, L. Yu, W. Tang, Z. Lan, R. Gupta, N. Desai, S. Coghlan, and D. Buettner, "Co-analysis of ras log and job log on blue gene/p," in *2011 IEEE International Parallel & Distributed Processing Symposium*. IEEE, 2011, pp. 840–851.

[98] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Largescale system problem detection by mining console logs," *Proceedings of SOSP'09*, 2009.

[99] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection," in *2010 USENIX Annual Technical Conference (USENIX ATC 10)*, 2010.

[100] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, "Failure prediction in ibm bluegene/l event logs," in *Seventh IEEE International Conference on Data Mining (ICDM 2007)*. IEEE, 2007, pp. 583–588.

[101] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural computation*, vol. 13, no. 7, pp. 1443–1471, 2001.

[102] D. M. Tax and R. P. Duin, "Support vector data description," *Machine learning*, vol. 54, no. 1, pp. 45–66, 2004.

[103] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 eighth ieee international conference on data mining*. IEEE, 2008, pp. 413–422.

[104] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen, "Fingerprinting the datacenter: automated classification of performance crises," in *Proceedings of the 5th European conference on Computer systems*, 2010, pp. 111–124.

[105] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer, "Failure diagnosis using decision trees," in *International Conference on Autonomic Computing, 2004. Proceedings.* IEEE, 2004, pp. 36–43.

[106] D. Preuveneers, V. Rimmer, I. Tsingenopoulos, J. Spooren, W. Joosen, and E. Ilie-Zudor, "Chained anomaly detection models for federated learning: An intrusion detection case study," *Applied Sciences*, vol. 8, no. 12, p. 2663, 2018.

[107] A. B. Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. García, S. Gil-López, D. Molina, R. Benjamins *et al.*, "Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai," *Information fusion*, vol. 58, pp. 82–115, 2020.

[108] M. Landauer, F. Skopik, M. Wurzenberger, W. Hotwagner, and A. Rauber, "Have it your way: generating customized log datasets with a model-driven simulation testbed," *IEEE Transactions on Reliability*, vol. 70, no. 1, pp. 402–415, 2020.

[109] M. Landauer, F. Skopik, M. Frank, W. Hotwagner, M. Wurzenberger, and A. Rauber, "Maintainable log datasets for evaluation of intrusion detection systems," *arXiv preprint arXiv:2203.08580*, 2022.

[110] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A sense of self for unix processes," in *Proceedings 1996 IEEE Symposium on Security and Privacy*. IEEE, 1996, pp. 120–128.