applied sciences

MDPI

Review

Automatic Parsing and Utilization of System Log Features in Log Analysis: A Survey

Junchen Ma, Yang Liu, Hongjie Wan and Guozi Sun *

School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing 210049, China; junchenma@126.com (J.M.); yangliu200004@126.com (Y.L.); wan2njupt@163.com (H.W.)
* Correspondence: sun@njupt.edu.cn

Abstract: System logs are almost the only data that records system operation information, so they play an important role in anomaly analysis, intrusion detection, and situational awareness. However, it is still a challenge to obtain effective data from massive system logs. On the one hand, system logs are unstructured data, and, on the other hand, system log records cannot be directly analyzed and calculated by computers. In order to deal with these problems, current researchers digitize system logs through two key steps of log parsing and feature extraction. This paper classifies, analyzes, and summarizes the current log analysis research in terms of log parsing and feature extraction by investigating articles in recent years (including ICSE, TKDD, ICDE, IJCAI, ISSRE, ICDM, ICWS, ICSME, etc.). Finally, in combination with the existing research, the research prospects in the field are elaborated and predicted.

Keywords: log parsing; log mining; anomaly detection; artificial intelligence; deep learing; machine learing

Citation: Ma, J.; Liu, Y.; Wan, H.; Sun, G. Automatic Parsing and Utilization of System Log Features in Log Analysis: A Survey. *Appl. Sci.* **2023**, *13*, 4930. https://doi.org/10.3390/app13084930

Academic Editor: Christos Bouras

Received: 18 March 2023
Revised: 11 April 2023
Accepted: 12 April 2023
Published: 14 April 2023

1. Introduction

In the last decade, there has been significant growth and development in modern software systems. 24-h service software systems such as communications, short videos, government services, and search engines have become essential in our daily lives and can be found everywhere. Unlike traditional services, even a momentary downtime of these 24/7 software systems can result in immeasurable economic losses [1]. In July 2022, Microsoft reported that its Teams application could not be accessed normally due to internal storage service issues, which consequently affected the use of multiple Microsoft products integrated with Teams [2]. Similarily, in August of the same year, Google experienced the same situation, which prevented users from accessing its services for about an hour [3]. It can be seen that, with frequent updates of the system and the complexity of functions, those services that we think are very stable are also facing unpredictable system crashes at any time. On the other hand, today's network services are intricately interconnected with physical and digital services that are closely interwined. Thus, any downtime of basic services can have cascading effects on end customers. Amazon reported that some customers had to wait for three hours before their services were restored gradually.

Software system logs are the only data that record software operation information in system reliability assurance tasks; thus, they play a very important role in the industry [1] and include many services, such as intrusion detection [4–6], digital forensics [7,8], and situational awareness [9]. Compared with network data packets, system logs record more important information, such as memory and CPU information. Based on this, researchers use system log information to supplement intrusion detection based on network data packets [6]. Slightly different from this, Lou et al. [5] mined association rules from multi-source log data, such as security software logs and system logs, to build an intrusion detection system for cloud platforms. Digital forensics requires researchers to analyze attack

vectors and further actions from existing scenarios in order to deal with the huge challenge of the huge and complex log data in the cloud computing system. Dalezios et al. [8] adopted the Distributed Management Task Force's (DMTF) Cloud Auditing Data Federation (CADF) to standardize cloud system logs, which provided a valuable analysis method for subsequent log-based cloud computing digital forensics. In addition to dealing with large-scale log data, situational awareness [9] also puts forward higher requirements for log analysis, such as predicting whether an abnormal state will occur in the future based on existing log data. In summary, these increasingly diverse downstream requirements all rely on log analysis tasks. In the past, log analysis was often done by developers manually constructing rules for regularity screening and anomaly detection. This manual screening method was effective decades ago. However, today's widely used software systems are often very complex, thereby presenting significant challenges to manual processing, which means a double test of accuracy and efficiency. On the one hand, the composition of the log statement is complex, as shown in Figure 1. Without professional knowledge, the statement contains a large number of characters that are difficult to be understood directly, for example, '081109', '145', and 'dfs.DataNode$DataXceiver' represent Date, PID, and Component, respectively. It is difficult to manually select appropriate rules for log and anomaly detection without relying on professional knowledge. On the other hand, it is difficult to manually process the continuous log output.

```
Log1:081109 203519 145 INFO dfs.DataNode$DataXceiver: Receiving block
blk_-1608999687919862906 src: /10.250.14.224:42420 dest: /
10.250.14.224:50010
Log2:081109 203519 145 INFO dfs.DataNode$PacketResponder:
PacketResponder 1 for block blk_-1608999687919862906 terminating

Template1:Receiving block <*> src <*> dest <*>
Template2:PacketResponder <*> for block <*> terminating
```

Figure 1. In the figure, Log1 and Log2 are two HDFS logs; Template1 and Template2 are their corresponding log templates.

Therefore, it is no longer feasible to rely solely on manual log analysis. In the past 15 years, with the application of machine learning and distributed technology in the field of log analysis, automated log analysis has greatly improved the efficiency and accuracy of log analysis. Figure 2 shows the workflow of log anomaly detection in current times. Log parsing refers to the process of transforming semi-structured log data into structured statements. The structured log information obtained through this parsing is called a log template [10,11], as shown in Figure 1. However, on the one hand, computers cannot directly recognize and process this original language information. On the other hand, these log templates often contain diverse log features, so log features need to be encoded and represented, which is the role of feature extraction. Anomaly detection uses machine learning or deep learning methods to classify and predict logs in order to merge logs into abnormal or normal events.

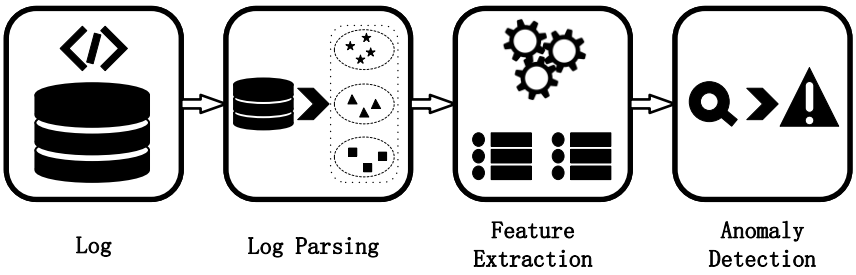

Figure 2. This picture shows the process of log analysis, including log mining, feature extraction, and anomaly detection.

In the last decade, the scale of software systems has expanded rapidly, and the field of log analysis is facing new challenges such as increasing log formats and log volumes. A large number of researchers have proposed to use machine learning and deep learning technologies to solve problems in the field of logs. For example, clustering methods are used for log parsing [12,13], and convolutional neural networks are used for anomaly detection [14], etc. At this time, researchers use a large number of log features based on statistics and sequence order. In the last few years, thanks to the rapid innovation of deep learning algorithms and the accumulation of a large amount of social text data, natural language processing technology has developed by leaps and bounds. Researchers began to use log semantic features for anomaly detection and proposed a variety of robust anomaly detection methods [15–18]. To this day, a large number of articles are still published in related fields every year. In this regard, we have investigated related articles from the past 8 years and made the following year distribution statistics. The statistical results are shown in Figure 3. By investigating the literature, especially the literature from the past three years, we found that most of the existing scientific research and review literature focus on the use and innovation of anomaly detection models, while less research has been published on the acquisition and use of log features. Therefore, this paper describes the log parsing method, as well as log feature classification and utilization, and introduces the research status in the field of log anomaly detection. The main contributions of this paper are as follows: (1) Review, classification, and comparison of the existing analysis methods, and judgement of the two evaluation indicators at the same time. (2) First-time classification of the existing log anomaly detection solutions based on the utilization of log features. We believe that this classification is more in line with the data mining process. (3) Suggestions made for further research and exploration in the field. The rest of this paper mainly answers the following questions:

RQ1: To date, what are the log parsing methods in academia, and what are the differences among them?

RQ2: What kinds of log features can be used in the feature extraction process?

RQ3: How does one apply log features to anomaly detection models, and what are the specific research applications?

RQ4: What suggestions are there for future research and development in the field of log anomaly detection?

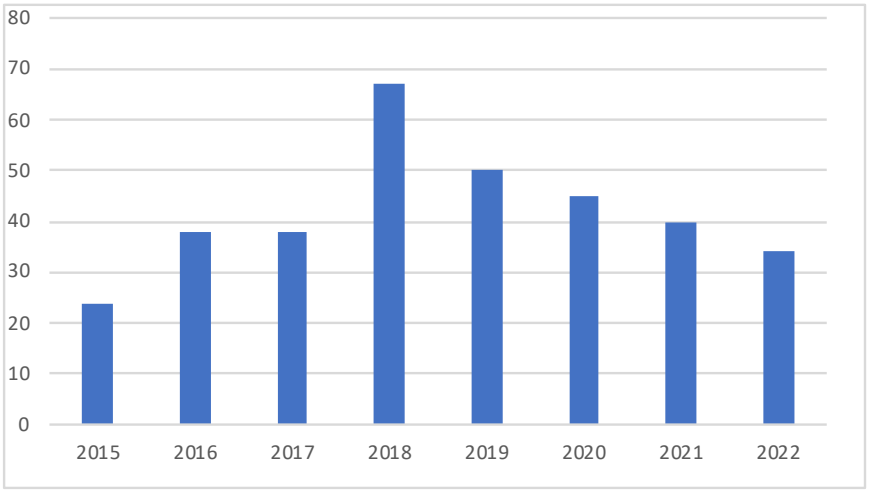

Figure 3. This picture shows the literature distribution in the field of log analysis in the last eight years.

The second chapter gives an answer to RQ1. We introduced several classic log parsing algorithms and several log parsing algorithms proposed in recent years and analyzed their internal correlation by classifying the underlying principles of these algorithms. Furthermore, we summarized the evaluation indicators of log parsing and compared the accuracy of the log parsing algorithms mentioned in the chapter. RQ2 and RQ3 can be

found in the third chapter. We classified the log features used in existing research, analyzed and compared the usability of different log features, and combined those findings with the literature; this paper introduces how existing research use these log features, as well as their characteristics and the performance of different methods. In Section 4, we introduce the research prospects and feasible directions in the field of log anomaly detection in the future.

2. Log Parsing

Log parsing is a key step in the log analysis process. Generally, a log message includes a message header and log content. Information, such as the timestamp and message level contained in the message header, is easier to extract, but the log content is often a natural language record, and it is difficult to extract useful information from it. Therefore, in today's software engineering, log parsing is still a challenging task. Here are two main challenges: (1) A large amount of log information will be generated when the complex and huge software architecture is running, which means the traditional way of manually constructing regular expressions will be too expensive; (2) Complex software functions and high-frequency business updates lead to more frequent updates of log templates, thereby greatly increasing the diversity of log templates [1]. Therefore, an efficient and accurate log parsing method will directly affect the scientificity and effectiveness of process mining. This chapter is divided into two sections: Section 2.1 classifies and compares the classic parsing algorithms and the log parsing algorithms proposed in the past two years from the perspective of the working principle of the parse, while Section 2.2 summarizes the existing accuracy indicators and gives a summarization and evaluation of the accuracy of these algorithms for a public dataset.

2.1. Classification of Log Parsing

2.1.1. Clustering

A log template is an aggregated representation of a group of similar logs, so a clustering algorithm can be used to cluster log messages. We introduce and compare three clustering-based methods below. LogMine [13] employs hierarchical clustering and an early abandoning technique, and it iteratively uses clustering and pattern recognition algorithms to generate hierarchical structures from the bottom up. The advantage of this method is that any level can be selected as the schema set. In addition, LogMine is scalable on MapReduce. Therefore, LogMine has been one of the important analysis tools used by the industry in the past period of time. However, with the actual deployment of a large number of parsing algorithms on distributed frameworks, due to its complex internal hierarchy, LogMine's performance gradually lags behind the latest algorithms. In LPV [19], researchers have used word-embedding technology to vectorize log events and to cluster log events according to the similarity of log parsing vectors. Then, through two steps of intra-cluster merging and inter-cluster merging, log templates are extracted from log event clusters. Similarly, QuickLogS also uses word-embedding for template clustering. The difference is that, in QuickLogs [20], researchers use more analytical rules based on professional knowledge and log message length, and they then convert the pre-processed log template into a fixed-dimensional vector, use SimHash to reduce the dimension of features to obtain the vector of each template, and merge similar log templates of the same length through Hamming distance and token position. Finally, the cosine similarity algorithm has been used to merge similar log templates of different lengths. This is the first application that uses SimHash log parsing to speed up parsing efficiency. In addition, the encoded features generated during the parsing process, such as LPV and QuickLogS, can be used in downstream feature extraction and anomaly detection, and they have good prospects for use in model systems that use related methods for anomaly detection.

2.1.2. Heuristic

A heuristic algorithm refers to a method for obtaining a feasible solution within an acceptable cost based on certain rules and experience. Since log events are usually

semi-structured data, it is usually feasible and convenient to use expert experience and relevant rules for parsing, so heuristic rules are used in many studies. The longest common subsequence refers to the problem of finding the longest subsequence in a set of sequences, which is very similar to the process of extracting common templates from log statements. Researchers have proposed a parsing method based on this similarity principle [21]. The Spell [21] algorithm applies a prefix tree for preprocessing, and uses reverse lookup to reduce the computational overhead of matching lookups. When a new log event appears, Spell first searches through the prefix tree to see if the existing log key matches. If there is no match, it uses the reverse lookup strategy to match the log key. What is worse, if there is still no match, then it will use loop traversal to search for a match. Therefore, when the log messages to be parsed are too long, Spell often still needs to use loop traversal to assist in the search, which makes its performance inefficient on datasets such as OpenStack. Drain [22] uses a fixed-depth parsing tree for log parsing. On the non-leaf nodes of the parsing tree, log events are processed according to different heuristic rules, and, then, the log templates obtained by parsing are stored in the leaf nodes of the parsing tree. Specifically, at different depths of the parse tree, heuristic matching is performed according to professional knowledge, log message length, log header token, and log token similarity. In the matching search process, if the similarity is lower than the set threshold, the current node will be split so as to update and expand the parse tree. The Drain algorithm shows a good parsing effect on a wide range of datasets, so it has received widespread attention in the academic community, and more and more heuristic algorithms based on parsing trees have been proposed. LTmatch [23] combines the ideas of the Drain and Spell algorithms; while using a tree structure for efficient access to log data, it compares and matches log templates based on the longest common subsequence. In addition, LTmatch also performs weighted matching on the words in the log to adapt to different log datasets. USTEP [24] also uses a tree structure for log parsing. Unlike Drain, USTEP can encode new log parsing rules without recording huge log information in its memory. In addition, when the number of USTEP leaf node templates exceeds the threshold, it will be classified into the parse tree, and new leaf nodes will be divided at the same time. Researchers have also designed USTEP-UP, which can share parsing information and knowledge monitoring so that it can be used for distributed log parsing. There are some differences between the industrial log dataset and the public academic dataset, such as separators, template nesting, etc., which prevent some excellent parsing algorithms from being used in the industry on a large scale. Drain+ [25] is an adjusted version made by researchers of Drain for real datasets. Drain+ generates delimiters based on the statistics of log samples for log message splitting instead of using a single delimiter. On the other hand, Drain+ merges log templates according to template similarity to reduce error analysis caused by the same template, but this results in large differences in log length. This kind of data actually exists in many real datasets. Paddy [26] maintained a dynamic dictionary as a data structure to improve the efficiency of matching log templates. In order to improve the accuracy of template matching, Paddy internally sorts the template set by similarity and length. However, a test on a public dataset showed that Paddy cannot deal with several GB of log data as efficiently as Drain can. MoLFI [27] is the only work known to us that employs evolutionary algorithms for log parsing, which treats the log parsing problem as a multi-objective optimization problem and uses a genetic algorithm to search for the optimal log template set. Its advantage is that it requires fewer parameters; accordingly, the analysis efficiency is low, and it is limited by the characteristics of the genetic algorithm. When faced with different datasets, it may fall into a local optimal solution, thus resulting in low analysis accuracy. In the previous literature [1], researchers expressed it as a separate category (such as evolutionary algorithms). We consider that evolutionary algorithms are also a heuristic algorithm, so we classify them into this category.

2.1.3. Utility Itemset Mining

Utility itemset mining is a technique commonly used in data mining, and it is used to solve specific problems related to identifying high value itemsets in transactional databases [28,29]. In the field of log parsing, the utility itemset consists of frequently occurring log words, parameters, time, and other information, with the most critical being the frequent words that make up the log template. Therefore, more specifically, the practical itemset mining in the field of log parsing mainly adopts frequent pattern mining.

The log parsing algorithm using frequent pattern mining is often based on the assumption that the words that appear frequently in the log are often part of the template. Similar to the early log parsing algorithms (SLCT [30], LFA [31]) that adopted frequent pattern mining, LogCluster [12] also relies on the frequency of words in log events. The difference is that LogCLuster allows variable length parameters, which combine multiple consecutive parameter tokens into one parameter token, so LogCLuster is better at handling log data with variable parameter lengths. When log statements containing the same parameters appear repeatedly within a period of time, LogCluster will include these frequently occurring parameters into the log template, which makes LogCluster inaccurate with some datasets. Unlike LogCluster, Logram [32] does not consider the number of occurrences of log tokens but instead considers the frequency of log n-grams. In Logram, log parsing is based on the assumption that frequent n-grams may be log templates. Logram first adopts a recursive method to continuously filter and extract *(n-1)-grams* from *n-grams* and then splices word pairs with a sequence relationship to form a log template. This kind of frequent pattern mining considering the context can avoid the problem of mis-parsing parameters. ULP [33] divides logs according to the length of log sentences and potential template tokens, pre-groups similar log events, and then performs word frequency analysis. Unlike Logram, ULP performs a separate word frequency analysis within each log group, rather than all log data, which makes it easier to distinguish log template parts. Prefix-Graph [34] automatically extracts frequent items of logs by constructing a prefix graph. In simple terms, this method first constructs a prefix tree for the log message collection, and each log message is represented as an edge from the root to the leaf node in the prefix tree. Then, its traverses the entire prefix tree and merges the same nodes of similar tree branches together to form a prefix graph until the prefix graph does not change. The common node sequence from the root to the leaf nodes is the log template. Based on the above process, Prefix-Graph shows excellent accuracy with various datasets. When dealing with datasets with many templates and long log messages, the prefix tree traversal and merging steps in the algorithm will bring a certain amount of time overhead.

2.1.4. Others

LogStamp [35] converts the log parsing problem into a sequence labeling problem. Unlike other log parsing methods, LogStamp considers semantic information in log parsing. In order to improve the accuracy of the parsing, LogStamp trains the sequence tagger from two perspectives of word vector and sentence vector. Sentence embedding is used to reflect the different properties of the log, that is, to divide the log log template, and word embedding is used to train the log tagger, that is, to associate the word vector with the log template. This method works well when training with only a small amount of log data. Existing log parsing schemes are all based on comparing logs in the same dataset, and they cannot parse a single log. In this regard, Jared Rand et al. [36] explored a method for parsing a single log message without relying on any other data. To achieve the intended purpose, the researchers transformed the log parsing problem into a machine translation problem and conducted experiments on three machine translation architectures. As opposed to other methods, in LogPunk [37], researchers believe that matching log words is a factor that affects the parsing effect, so it classifies logs according to the punctuation marks in log messages and extracts log templates on this basis.

2.2. Accuracy of Parsing

2.2.1. Evaluation Indicators

There are many evaluation indicators for log parsers, such as accuracy, efficiency, etc. In addition, researchers have compared and verified the adaptability and incremental learning ability of the parsers [34]. For the log parser, the most important attribute is the accuracy of log parsing. It is no exaggeration to say that the accuracy of parsing is directly related to the completion of downstream tasks. As a machine learning task, traditional evaluation indicators such as *Precision*, *Recall*, and *F-measure* are applied to log parsing.

$$Recall = \frac{TP}{TP + FN} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{3}$$

TP refers to the number of logs that have been parsed into the correct log template, *FN* refers to the number of logs that have been parsed into the current log template by mistake, and *FP* refers to the number of logs that have been parsed into other error log templates. However, in this field, the accuracy indicators most commonly used by researchers in recent years are Parsing Accuracy (PA) and the Rand Index (RI).

In order to better quantify the accuracy of the log parser, researchers proposed the evaluation indicators of parsing accuracy [38]. They can be interpreted as the ratio of the number of accurately parsed log messages to all log messages, as shown in Formula (4). For example, for the log message sequence $[L_1, L_2, L_3]$, the parsed template sequence is $[T_1, T_2, T_3]$, but the actual template is $[T_1, T_5, T_6]$; in this case, $PA = 1/3$. Because the latter two log events match incorrectly, partially matched events are considered incorrect in the calculation of the parameter PA. There is no doubt that log parsing accuracy is more stringent than traditional parameters (such as *Recall*, *F-measure*, etc.) [38].

$$PA = \frac{\#Correct\_Templates}{\#Total\_Templates} \tag{4}$$

$$RI = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

Some researchers have selected the Rand Index as the evaluation index [34,35]. The Rand Index is a common method to evaluate the accuracy of two or more classification tasks. The Rand Index is calculated by Formula (5), which has similar calculation formula to precision and recall, but its specific meaning is different. *TP* refers to the number of log pairs that are classified into the same category and have the same template; *TN* refers to the number of log pairs that are divided into different categories and have different templates; *FP* refers to the number of log pairs that are divided into different categories and have the same template; *FN* refers to the number of log pairs that are classified into the same category and have different templates [34,35]. Compared to the recall and other evaluation indicators, the Rand Index can more comprehensively describe the accuracy of the analytical algorithm in the datasets. In addition, there are some studies that use self-defined accuracy indicators. The researchers in [33] considered the limitations of the accuracy of the expression of existing indicators, and they also made requirements for the accuracy of the parsed dynamic tokens. Such indicators are not within the scope of our research.

2.2.2. Accuracy

In this section, we will compare the accuracy of the above-mentioned parsing methods. Due to the limitation that some methods do not provide open source code, we prefer to use the accuracy data in their articles, and some null data cannot be filled. These data statistics can be used as an extension of the existing statistics [38]. All experiments were performed on a computer with 8 G memory, model Intel(R) Core(TM) i5-7300HQ, and CPU clocked at 2.50 GHz. Table 1 shows the accuracy of the four classical and six new parsing methods listed in this section on five typical datasets. Each line in Table 1 expresses the difference in the effect of one log parsing method on different logs. It can be seen that most of the parsing methods work well on HDFS logs for two reasons: the total number of log templates in HDFS datasets is small, and the length of the log sentence is short. On OpenStack logs with complex structures, almost all log parsing methods are less effective. Looking at the table vertically, one can make a preliminary judgment on the quality of different log parsing methods regarding the same dataset. Except for the OpenStack dataset, there are good parsing methods with parsing accuracies higher than 95%. The lower half of Table 1 shows the specific performance of two classic and two new methods on five typical datasets. From the table, we can draw the following conclusions: (1) It is relatively difficult to parse log datasets with long message lengths and a large number of templates. (2) Compared to the Rand Index, which is widely used in clustering, the parsing accuracy is a more stringent evaluation index, which explains the rationality of its widespread use. As mentioned above, the number of log templates and the length of log events are potential factors that affect the parsing, and Table 2 shows the basic information of the typical datasets used in Table 1. Although the HDFS dataset has a large amount of data, the number of vocabulary and the number of templates are the least, which allows the parser to easily identify log events classified into different templates. In addition, the length of the log event needs to be considered. The length of the log event reflects the complexity of the log event structure to a certain extent. The log statement of OpenStack contains 295 characters on average, which is much higher than other datasets. This may be related to why various parsers perform poorly on it. It can be seen that most log datasets can be better parsed by one or more existing log parsing methods. However, for log datasets with complex log structures such as OpenStack, accurate parsing is still difficult. Existing log parsing methods seem to be "length sensitive". For example, MoLFI uses an evolutionary algorithm to discover the optimal set of log templates, but OpenStack's complex log events contain too many interleaved variable parts (parameters, etc.). This makes it easy for the evolutionary algorithm to incorporate those frequently occurring variable parts into the log template, which results in greatly reduced parsing accuracy. After testing, it was only 0.213, which is marked as "*" in Table 1.

Table 1. Accuracy of some log parsing methods mentioned in this article.

Metrics	Log Parser	Year	HDFS	BGL	Zookeeper	OpenStack	Hadoop
PA	LogCluster	2015	0.546	0.835	0.732	0.696	0.563
PA	Spell	2016	0.999	0.787	0.964	0.764	0778
PA	Drain	2017	0.999	0.963	0.967	0.733	0.948
PA	MoLFI	2018	0.998	0.960	0.839	0.213 *	0.957
PA	Paddy	2020	0.940	0.963	0.986	**0.839**	0.952
PA	QuickLogS	2021	0.998	-	0.965	0.837	-
PA	USTEP	2021	0.998	**0.964**	**0.988**	0.764	0.951
PA	LTmatch	2021	**1**	0.933	0.987	0.835	**0.987**
PA	Drain+	2022	**1**	0.941	0.967	0.807	0.954
RI	Spell	2016	0.999	0.880	0.999	**0.917**	**0.999**
RI	Drain	2017	**1**	0.912	**1**	0.904	0.982
RI	Prefix-Graph	2021	0.989	**0.993**	**1**	0.890	**0.999**
RI	LogStamp	2022	**1**	0.98	0.95	-	0.95

"*" This method is not suitable for this dataset, and the explanation is given in this paper.

Table 2. Basic information of the test datasets used in Table 1.

Datasets	# Words	# Templates	Len_{MAX}	Len_{AVE}	Size	# Messgae
HDFS	132	30	320	139	1.58 GB	11,175,629
BGL	1053	619	928	156	74.32 MB	4,747,963
Zookeeper	261	95	387	1387	10.4 MB	74,380
OpenStack	282	51	450	295	61.4 MB	207,820
Hadoop	880	298	592	121	48.2 MB	394,308

"#" This symbol represents the number of counts.

For datasets such as OpenStack with long log event lengths and complex log event formats, the potential feasible methods are as follows: (1) Multi-layer grouping of log events—group log events by length, and, then, divide the grouped content layer by layer so as to reduce the impact of "length sensitivity"; (2) Do not use the method of relying on sequence comparison—for example, use the idea of LogStamp to sequence label each word in the log event, and combine parts marked as templates into log templates; (3) Take preprocessing measures for the logs to be parsed. Log events are semi-structured texts, which contain a large number of parameters and time fields that carry less information, and some information can be discarded in the preprocessing stage. In fact, just like other machine learning application scenarios, a good preprocessing step can greatly improve the parsing effect.

In addition, we also noticed that most of the parsing methods proposed in recent years are online log analysis solutions, such as Prefix-Graph, Drain+, USTEP, etc., which is related to the huge potential of online log parsing methods. Online log parsing methods can extract available information from real-time log streams, thus providing the possibility for downstream intrusion detection and situational awareness.

3. Feature Extraction and Utilization

The parsed log template sequence often has tens of thousands or even hundreds of thousands of logs. In order to facilitate log feature extraction and subsequent processing, the log sequence that results after log parsing need to be partitioned. In other words, the log is divided into blocks of limited size according to the log event occurrence time or a specific identifier (such as blk_id in the HDFS log). According to the method of grouping, it can be divided into time grouping and session grouping. Time grouping divides logs into sequences based on a self-defined time span and can be divided into fixed time windows and sliding time windows according to whether the time spans overlap. Session grouping is based on labels (such as blk_id) in log events, and log events are divided into multiple groups of log sequences according to labels [1].

Generally speaking, most parsing methods will separate keywords describing the running status of the log, such as ERROR, INFO, etc. However, due to the high complexity of the software system, it is difficult to accurately judge the current running status of the system only by relying on these keywords, so there is an urgent need for an automated method to obtain the features in the log and make use of these features. In order to automatically analyze the log, it is necessary to convert the log template that results after log parsing into numerical information that the machine can understand. According to the investigation of He et al., the existing research mainly utilizes two types of numerical information for logs: digital features and graphical features [1]. The content of this chapter is arranged as follows. The first and second subsections introduce the current main log features and their utilization, respectively. The third subsection makes statistics and comparisons between the current anomaly detection methods.

3.1. Digital Features

Digital features are values or log strings that can be directly extracted from logs, and they are the most commonly used type of log features in log feature extraction and utilization. A variety of log features can be extracted directly or indirectly from log

partitions, some of which can be used alone for downstream anomaly detection, which can be called main features, such as log count features, log index features, log event features, etc. The other part of log features is often used to cooperate with the main features to improve the accuracy of anomaly detection in a certain dataset or certain aspects, such as log time features, log parameter features, etc. Our survey results show that the main features extracted from log templates have been used more in anomaly detection research, while the features based on the variable parts of logs have been less used by researchers and can be called secondary features. In the following, we classify and introduce existing methods from the perspective of log digitization feature extraction and utilization.

3.1.1. Log Count Features

Log counting refers to counting the number of log events in a log block to generate log vector information. The dimension of the vector often represents the number of event types in the entire log dataset. For the log template sequence shown in Figure 4, assuming that there are only four different types of log templates in the current dataset, the log count vector is $[5, 4, 0, 2]$, because there were five occurrences of Event 1, four occurrences of Event 2, one occurrence of Event 4, and Event 3 did not occur. This log feature allows the establishment of quantitative relationships between the occurrences of different log events over time. Since it is relatively easy to extract log count features and can be used to achieve certain results in anomaly detection, it has been used in many works. He et al. [39] combined multiple log count features into a log count matrix and used the matrix as a feature to use and compare six machine learning methods such as Decision Tree, SVM, and PCA. Xie et al. [40] divided log events into dependent and independent events and judged anomalies by detecting whether the log sequence violated the dependency pattern of the log. Zhao et al. [41] found that there is a number relationship between different log templates: the value of one dimension in the counting feature increases, and the value of the other dimension decreases. Therefore, the application of multivariable time series anomaly detection technology can improve the accuracy of anomaly detection.

3.1.2. Log Index Features

In contrast to the log count feature, the log index feature is not based on the frequency of log events. Instead, it is based on the encoding of the log templates index. Each log event template can be encoded into an index number, and the index numbers sorted by execution time are combined to form the log index feature. Each log event template can be encoded into an index number, and the index numbers sorted by execution time are combined to form the log index feature. Furthermore, for the log template sequence shown in Figure 4, its log index vector may be $[1, 1, 4, 2, 1, 1, 2, 2, 4, 2, 1]$, as each number represents the index of its corresponding template. Obviously, this method can not only record the number distribution of log events, but can also preserve the positional relationship of log events, which makes the log index feature more widely used. Du et al. [14] used LSTM to capture the potential log sequence relationship of log index vectors to predict whether the next log event deviated from the normal log event sequence. Lu et al. [42] proposed a CNN-based anomaly detection method by using a parser to find log constants that frequently appeared in log events, named it log-key, and then constructed a one-to-one mapping of the log-key and index. Next, they sorted the index according to the time spent to construct an index vector, encoded the index vector into an index matrix, and finally used three convolutional layers of different sizes of filters for feature extraction. This method can well capture the log features of different ranges in the log index. Similar to Lu et al., Yen et al. [43] also exploited the feature extraction ability of a CNN, used convolution filters of different sizes for feature extraction, and then used LSTM to capture the relationship between sequences for anomaly detection.

It should be noted that, due to the unevenness of the log partition, the vector lengths generated by the log index feature are often different, so shorter and longer log vectors need to be processed, but deleting the vector length can easily cause data loss, which

makes it difficult for the above methods to achieve a better average effect on multiple sets of datasets.

```
Template1:Receiving block <*> src <*> dest <*>
Template1:Receiving block <*> src <*> dest <*>
Template4:Deleting block <*> file <*>
Template2:PacketResponder <*> for block <*> terminating
Template1:Receiving block <*> src <*> dest <*>
Template1:Receiving block <*> src <*> dest <*>
Template2:PacketResponder <*> for block <*> terminating
Template2:PacketResponder <*> for block <*> terminating
Template4:Deleting block <*> file <*>
Template2:PacketResponder <*> for block <*> terminating
Template1:Receiving block <*> src <*> dest <*>
```

Figure 4. Log template sequence after log parsing. The "*" in the figure represents the variable part in the log template.

3.1.3. Log Event Features

The log count features and the log index features mainly focus on the distribution of log events in time and space, which is a statistical feature. However, with changes in system functions, modification of log statements, and other actual operations that lead to changes in the syntax or types of log events, the above log features cannot guarantee normal operation while maintaining the existing coding structure. Furthermore, such statistical features cannot describe the semantic information contained in the log. With the development of deep learning models and natural language processing technologies, the feature extraction of log events using related technologies has become the most common method in today's research.

Bertero et al. [44] regarded log sentences as ordinary natural language texts, rather than formatted templates, and applied Google's word2vec for word embedding to obtain log event features. Word2vec can obtain the contextual features of log sentences very well, but it has difficulty in obtaining log semantics. Meng et al. [15] proposed a template vectorization representation method, Template2Vec, by adding log synonyms and antonyms, which could extract semantic information in logs simply and effectively. It is also the first feature extraction method that considers log semantic information. This method can match new generated, unseen log templates into existing log templates based on semantic features, which allows the model to adapt to the changing actual production environment without frequent modification, and Meng et al. utilized both log count features and log semantic features to detect the statistical and logical relationships of log events. LogRobust [16] obtains word embedding through FastText, uses TD-IDF for vector weighting, converts each log event into a fixed-dimensional semantic vector to obtain the semantic information embedded in the sentence, and uses Bi-LSTM for semantic learning. This method can adapt to basically similar, but not identical, log statements generated with iterative updates of the system. Li et al. [18] used Bert for log feature extraction. In contrast to other feature extraction methods, this method adds a self-attention layer to the encoder to obtain information implicit in log sentences. Huang et al. [17] split the compound long words in the log into multiple sub-words and at the same time used Bert to generate word vectors. In this way, the log template sentence was transformed into a list of word vectors. Wang et al. [45] believed that calculating the log event vector from the word vector brought huge computational overhead, so they proposed that LogEvent2Vec directly encode the log event vector from the log event. Specifically, for the log sequence $[L_{i+1}, L_{i+2}, \ldots, L_{i+L}, \ldots, L_{i+M}]$, Wang et al. used $[L_{i+L-k}, \ldots, L_{i+L-1}, L_{i+L+1}, \ldots, L_{i+L+k}]$ as the input of word2vec, and the goal was to obtain the log event features of the log event L_{i+L}. This method of encoding log features according to the log context can simultaneously represent the log sequence

features and log semantic features. However, its actual effect remains to be further verified. Ying et al. [46] counted the log word frequency in their research, and calculated the weight $Weight_{Log}^{Word}$ of each word in each log event through TF-IDF; for example, by converting the event L into a word weight vector $[Weight_L^{Word_1}, Weight_L^{Word_2}, \ldots, Weight_L^{Word_n}]$, each dimension value represents the weight representation of a word in the dictionary in the log L.

3.1.4. Log Sequence Features

A log sequence refers to a set of consecutive log records in a log file. The log event features encoded by a log event cannot record enough information, and, in particular, the relationship feature before the log event cannot be recorded. On the other hand, in order to meet different anomaly detection methods, it is also necessary to construct log sequence features. For example, the time series information of log event features cannot meet the needs of random forest classification. Just as sentence vectors or log event vectors are obtained from word vectors, log sequence features are generated in a similar manner. For example, by using TF-IDF to transform log sequence features into a weighted average representation of log event features, the weight of rare log events is higher than that of common log events.

Lv et al. [47] considered that some words in the log are invalid words (such as "to"), and the word embedding technology word2vec cannot filter these pairs of meaningless words. Furthermore, because word2vec uses words as the coding unit, these frequent invalid words will increase the vector space. In order to reduce the waste of computing resources, a method called log2vec was proposed. Log2vec performs part-of-speech tagging on words, and vocabularies tagged as invalid words are set to zero vectors by the encoder. Then, the log event vector is generated by weighting the corresponding log template words, and, similarly, the log sequence vector is generated by averaging the log event vector. Yang et al. [48] explored the method of constructing log sequence features by modeling the word–word interaction information in a log sequence. This encoding method using word-by-word matching can deal well with new logs in the evolution of the system. In LogEvent2Vec [45], the log event is used as the input of the encoder to obtain the correlation between log events, and, on this basis, the log sequence vector is obtained by averaging the log event vector. It should be noted that, although this log sequence feature is aggregated from log event features, it does not mean that it is more suitable for anomaly detection. Since it is limited by machine learning classifiers, the accuracy is often lower than that of deep models, and the method of using log sequence features is not as effective as the latter. One possible way to improve accuracy is to reduce the size of the log event window [49].

3.1.5. Log Time Features

Generally, in the process of converting log original events into log templates will separate the original statements into multiple parts. As shown in Figure 5, a parsed original HDFS log statement is divided into seven different parts: Date, Time, Pid, Level, Component, Template, and ParameterList. In addition to the log template repeatedly mentioned above, there is a large amount of information that has not been used, including time characteristics such as date and time.

Most current sequence prediction methods are based on time-independent event sequences, such as predicting natural language sequences. However, for many real log events, the time span between events contains key sequence information. Time series are often event-dependent (a single time series has no practical significance). Li et al. [50] proposed two time encoding methods for this: (1) time masking of event embedding and (2) event-time joint embedding. Both methods can improve the accuracy of time series forecasting, which provides a precondition for applying time coding in log analysis.

Log event features and log sequence features have the ability to describe the running status of the software system, but it is relatively difficult to find some other problems (such as performance detection [51]). The traditional method to detect performance anomalies is

to perform intrusive detection when the system is running, which will affect the system performance. When the system has a performance problem, the time interval of some events in the system log will change significantly. Based on this, the researchers in [14] considered using the non-intrusive method of log exception detection to deal with the potential performance problems of the system. In order to capture time interval anomalies, the researchers Li et al. [18] calculated the time difference of two logs $\Delta t_i = time_i - time_{i-1} (\Delta t_1 = 0)$ and then combined the time differences between all logs to obtain the time difference sequence $\Delta T = [\Delta t_1, \Delta t_2, \ldots, \Delta t_k]$. However, the time difference sequence is a one-dimensional time series and can only contain limited information. Therefore, Li et al. and Xiao et al. [52] used one-hot encoding and a custom linear layer to upgrade the feature dimension of the time difference sequence to make it serve downstream tasks. However, the logs generated by modern distributed systems are usually interwoven by multiple processes, which makes it challenging to effectively extract and utilize temporal features.

```
Log3:081109 203615 148 INFO dfs.DataNode$PacketResponder: PacketResponder 1
for block blk_38865049064139660 terminating

 Date       Time      Pid       Level              Component
081109     203615    148       INFO       dfs.DataNode$PacketResponder
           Template
PacketResponder <*> for block <*> terminating
           ParameterList
['1', 'blk_38865049064139660']
```

Figure 5. Components of an HDFS log. The "*" in the figure represents the variable part in the log template.

3.1.6. Log Parameter Features

Most log parsers also separate the parameter values in the log statement while parsing the log statement into a log template, as shown in the parameter list ['1', '*blk*_38865049064139660'] in Figure 5. These parameters may be software function return values, block information, etc. These log parameters often record important system data, which makes it feasible to use log parameters to encode auxiliary models for anomaly detection. In DeepLog [14], log time intervals and parameter values are extracted simultaneously, and an LSTM model is constructed for parameter value vector sequences of different log event types, which simplifies the problem to multi-parameter anomaly detection. However, log parameters in some cases have numerical units or practical meanings, such as 'took 1 seconds to upload' and 'took 800 ms to upload'. Although the numerical values of the parameter values are quite different, the practical meanings they express are relatively close. Therefore, it is not possible to simply use standards such as average values to measure parameter value information in order to realize the interaction between parameter values and log templates. Huang et al. [17] combined parameter value encoding and log event encoding with the aim to integrate log template information into parameter value encoding. LogAD [41] not only models the value and variation of parameters, but also pays attention to the distribution characteristics of parameters, such as the city and browser type that request access.

3.1.7. Others

In addition to the log characteristics described above, the log also contains many other kinds of information, such as log level, process ID, component name, etc. Log levels include INFO, DEBUG, ERROR, WARN, etc. Even in different types of system logs, there are relatively few differences in the identification of log levels, so it is easier to obtain a universal log level encoder. The occurrence of the log levels ERROR and WARN has a high probability of judging that there is an error in the program operation, even if the system program can solve some errors by itself. Multiple process is a necessary technology for system software with complex functions. A set of log statements is often a collection of logs

generated by multiple process tasks, so the process ID and component name can be used as one of the characteristics for identifying and dividing logs [52].

3.2. Graphical Features

Graph analysis has been widely used in different research domains [53–55], and graph embedding techniques have attracted much attention. Perozzi et al. [56] used Random Walk to sample nodes in the graph, and obtained the co-occurrence relationship between nodes in the graph to learn the vector representation of nodes, which is similar to the idea of word2vec encoding of a natural language. In contrast to the random walk strategy based on the depth-first traversal algorithm, Tang et al. [57] adopted a breadth-first traversal strategy and defined the second-order similarity between graph nodes at the same time, which made the graph embedding method suitable for weighted graphs. Grover et al. [58] took the work of both into account, combined a depth-first search and a width-first search to sample vertices, and optimized the maximum occurrence probability of their neighboring vertices and given vertices. Simultaneously, neighboring nodes had the same encoding, which is different from Tang et al. Researchers have been working on graph structure in log mining. Hacker et al. [59] proposed a log graphical method based on a one-dimensional Markov random field, which can represent conditional probability relationships before words and phrases in log events. The existence of specific words depends on the arrangement of the preceding words. In order to obtain the logical (interdependent) and sequential relationships of system events, the researchers mined log features by constructing directed graphs that describe system relationships or system behavior. Compared to digital features, graphical features can better describe the intrinsic relationship between different log events. Therefore, graphical features are often used not only for anomaly detection, but also for intrusion analysis, intrusion detection, anomaly classification, and diagnosis. However, these functions are not entirely within the scope of our research. Therefore, in this section, a brief introduction is given to the methods of using graphic features in the existing work.

Since the objects in the log have relatively complex logical and sequential relationships, it is a great challenge to construct a log graph. Zhao et al. [60] extracted log objects and constructed 1:1, 1:n, n:m logical relationship structure diagrams between different objects to reconstruct the execution flow from logs afterwards to analyze the performance and behavior of distributed server stacks. Milajerdi et al. [61] built a high-level scenario graph based on log entities and the information flow between entities to detect advanced persistent threats. Liu et al. [62] classified log events into three log graphs by setting ten discriminant rules. These rules were the following: (1) Construct a log heterogeneous graph within a day based on the causality and sequence relationships within a day. (2) Construct a multi-day log heterogeneous graph based on the logical relationship of events within multiple days. (3) Finally, construct an object-based log heterogeneous graph according to user login and browsing behaviors, as well as the logical relationship among them. On this basis, the problem of unbalanced data categories and diversified attack methods was solved by using a graph-based random walk strategy. Yang et al. [63] constructed a continuous-time dynamic graph based on the specific operations and execution time in the log, which can be used for fine-grained anomaly discovery and tracking. Zhang et al. [64] studied the use of graph structures to describe system behavior sequences and combined them with microservice logs to train deep graph models to detect abnormal behavior in microservice systems.

3.3. Comparison of Feature Utilization

Table 3 compares some existing anomaly detection methods, especially those proposed in the last two years. The main focus is on the log features used, *Robustness*, *Rationale*, *Accuracy*, *Recall*, and *F-measure* values for the dataset. Because the HDFS is the most widely used in the field of log anomaly detection, we used the anomaly detection performance of the HDFS as the measure. He et al. [39] used log count features for anomaly detection on three unsupervised (LogCluster, PCA, Invarants Mining) and three supervised learning

(Logistic Regression, Decision, SVM) algorithms with the aim to evaluate the performance of the six methods mentioned above. Although the log count feature is a simple statistical feature, it only expresses the distribution of log events in space, which often makes its robustness poor. However, in both work LogAD [41] and LogDP [40], anomaly detection was performed by modeling the relationship between the number of occurrences of different log events, and good results were achieved with various datasets. DeepLog [14] models the log index sequence, which enables DeepLog to have the ability to express sequence features. LogCNN [42] also exploits log index features. Compared with the log count feature, the log index feature can express the connection of log events in time. However, the above methods cannot handle unknown log data well, and their robustness is poor. LogAnomaly [15], LogRobust [16], HitAnomaly [17], LogBert [65], and other methods use NLP technology to extract log features, and these methods have good robustness. However, anomaly detection methods that only use a single type of log feature have trouble efficiently dealing with multiple types of anomalous events. Detection methods using multiple log features (Deeplog [14], SwissLog [18], AllInfolog [52], etc.) have a better chance of detecting potential performance problems that are difficult to find in the system (such as system blocking, etc.).

Table 3. Feature utilization, robustness, and effect comparison of log anomaly detection methods.

Methods	Year	Features						Robustness	Principle	HDFS		
		Count	Time	Event	Index	Sequence	Parameter			*Precision*	*Recall*	F_1
LogCluster	2016	✓							Cluster	0.99	0.77	0.87
SVM etc.	2017	✓							SVM etc.	-	-	-
IM etc.	2017	✓							IM etc.	-	-	-
DeepLog	2017		✓		✓		✓		LSTM	0.93	0.94	0.93
LogCNN	2018				✓				CNN	-	-	-
CausalConvLSTM	2019				✓				CNN/LSTM	0.9	1	0.94
LogAnomaly	2019	✓		✓					LSTM	0.96	0.94	0.95
LogGAN	2020		✓		✓				GAN/LSTM	1	0.36	0.53
LogRobust	2020			✓				✓	Bi-LSTM	0.98	1	0.99
SwissLog	2020		✓	✓				✓	BERT/LSTM	0.97	1	0.99
HitAnomaly	2020			✓			✓	✓	Transformer	1	0.97	0.98
LogEvent2vec	2020					✓			RF etc.	-	-	-
ConAnomaly	2021			✓				✓	LSTM	1	0.98	0.98
LogBert	2021			✓				✓	BERT	0.87	0.78	0.82
LAnoBERT	2021			✓				✓	BERT	-	-	0.96
LogAD	2021	✓		✓	✓		✓	✓	LSTM etc.	-	-	-
KNN-Based	2021			✓				✓	KNN	0.97	0.99	0.98
LogDP	2021	✓						✓	MLP	0.98	0.99	0.99
Sprelog	2021			✓				✓	RoBERTa	-	-	-
PLELog	2021			✓				✓	GRU	0.95	0.96	0.96
AllInfoLog	2022		✓	✓			✓	✓	RoBERTa/Bi-LSTM	0.99	0.99	0.99
DeepTraLog	2022			✓				✓	SVDD/GGNNs	-	-	-

"-" This method does not use HDFS logs or does not show this indicator.

We can see that, with the increasingly powerful performance of deep models, using log event features combined with deep neural networks seems to have become a popular anomaly detection method in recent years. In addition, researchers have done some research on the different issues of log analysis. To solve the problem that the existing distributed anomaly detection models are difficult to migrate, Guo et al. [66] built a distributed log detection system using broad federated learning. In order to reduce the impact of homologous encryption and other schemes on the overall efficiency of the system, Guo et al. designed a lightweight CNN model, which did not guarantee the accuracy of anomaly detection. Chen et al. [67] proposed LogTransfer, which is a transfer learning log anomaly detection method that uses transfer learning from tagged log datasets and then detects anomalies on other log datasets that are similar but have different log events, considering that existing anomaly detection methods tend to be less robust and are only good at one type or specific log dataset. This improves the performance of the scheme on a wide range

of real-world datasets. In the system log, the number of normal log sequences was far greater than the number of abnormal log sequences. This sample imbalance problem greatly affected the accuracy of the anomaly detection. Xia et al. [68] proposed to alleviate the class imbalance of log datasets by using generative adversarial networks. In their research, the feature extraction link also used log index features, captured features through the generator, generated credible log sample data, and finally used the discriminator to identify whether the sample was generated by the generator so as to improve the sample quality of the generator. However, this method is less robust on real-world datasets. Yang et al. [69] proposed a semi-supervised anomaly detection method called PLELog, which uses probabilistic label estimation to reduce the anomaly detection error caused by uneven sample labels.

4. Prospects

Open datasets can be more accurately parsed by at least one log parser, but there is currently no log parser for multitype datasets. Even though log parsers such as Drain+ and Prefix-Graph show good results for a variety of datasets, they still have trouble dealing with data with a complex structure of log events, long variable parts, and nested log structures. We propose three possible solutions for this: (1) Group log events one by one; (2) Do not use text similarity comparison to classify; (3) Use preprocessing to remove some insignificant numerical features. However, these schemes may reduce the efficiency of parsing. Especially for online log parsers that are used in system monitoring and auditing, it is important to balance the accuracy and efficiency of the parser. On the other hand, some studies [70,71] abandon log parsing and extract the semantics of preprocessed log event statements directly. We think this is meaningful and promising, because the accuracy of log anomaly detection is limited to some extent by the accuracy of log parsing. As mentioned earlier, existing log parsers cannot guarantee the robustness and accuracy of parsing. In addition, using an unresolved log anomaly detection scheme can help produce a lighter anomaly detection framework, which will help break down barriers between anomaly detection for different log datasets.

In the concept of parsing accuracy, not exactly matching a template is considered an error, and, in some cases, we think this is too restrictive, such as tokens ‘blk*’ and ‘*’. Therefore, we strongly encourage researchers to try other evaluation metrics, such as using the adjusted Rand coefficient, which may better compare the difference between the parsed template grouping and the real template grouping.

At present, distributed systems are gradually developing toward heterogeneous computing, cloud computing, and mixed loads. The question of how to analyze logs in distributed systems while ensuring data privacy may become a difficult problem. In terms of log processing, some researchers have proposed using a distributed framework to analyze the system logs of distributed systems [72]. However, for a large-scale distributed system, even summarizing the parsed log templates from different terminals will consume a lot of time and resources, and there are also data security issues in the process. On the other hand, the difference in log statements between different distributed systems may be very large, which makes it difficult for a model built on a certain distributed system to be used on other distributed systems. This is something to be solved in the industry in the future. Although some researchers [66,67] have engaged in related work, there are still broad research prospects. For example, by combining graph feature technology to extract the relationship between log events and performing sequence flow relational transfer learning on log event relationships, this makes it possible to achieve multi-functional security detection and analysis across datasets and detection purposes.

LOGPAI [73] and other publicly available data [74] provide researchers with valuable log datasets and several available toolkits. However, many log parsing and anomaly detection researches do not open source their work, which makes it difficult to compare the pros and cons of different methods in the same experimental framework, and it is also difficult for future researchers to make feedback and improve existing work. In addition,

there is currently a lack of anomaly detection frameworks that can be deployed and used online, which may bring great convenience to those who need them.

It is a feasible direction to apply log analysis work to situational awareness and digital forensics, but it also poses new challenges to log analysis. To put it simply, most of the existing log anomaly detection schemes detect abnormal behaviors that have already occurred, and such delayed detection often cannot be used for situational awareness. Situational awareness needs to predict possible future events or trends based on past logs or other data. Situational awareness is often applied in the environments of multi-device, multi-system, and multi-task, so the parsing, sequencing, standardization, and feature aggregation of multi-category logs are also one of the problems to be solved urgently. Similarly, digital forensics also needs to face problems that involve multiple devices. Unlike situational awareness, digital forensics does not require the prediction of future statuses and trends. The key issue is to analyze the relationship between data entities in different types of information carriers to achieve information traceability and tracking, so graphical features may be the main applied research method.

5. Summary

In recent years, with the explosive growth of software system scales, system logs have become an important medium for recording software system behavior and analyzing system health. This paper mainly discusses two steps in automatic log parsing: log parsing and feature extraction. In the log parsing section, this paper compared the parsing methods in recent years on a variety of datasets. Although the parser principles are different, they can achieve good accuracy with most datasets. For complex log datasets that are difficult to parse accurately, three potential feasible solutions have been proposed. Feature extraction is an important step in log analysis. This paper introduced and compared the existing log feature extraction schemes, described the log feature utilization of researchers in recent years, and labeled the robustness and results of different methods. Finally, the potential research directions were described.

By surveying recent articles, this paper categorizes, analyzes, and summarizes the current research on log analysis in terms of log parsing and feature extraction. This paper is intended to enable researchers outside the field to have a certain understanding of log analysis and log feature extraction and utilization, as well as to fill in the gaps for new researchers in the field.

Author Contributions: Conceptualization, J.M. and H.W.; methodology, J.M.; software, J.M.; validation, J.M.; formal analysis, J.M.; investigation, J.M.; resources, J.M.; data curation, J.M.; writing—original draft preparation, J.M.; writing—review and editing, J.M. and Y.L.; supervision, G.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors would like to thank the Nanjing University of Posts and Telecommunications (NJUPT) for all the support provided for this research work.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

DMTF	Distributed Management Task Force
CADF	Cloud Auditing Data Federation
PID	Process Identity Document
PA	Parsing Accuracy

RI	Rand Index
HDFS	Hadoop Distributed File System
BGL	Blue Gene/L
PCA	Principal Component Analysis
SVM	Support Vector Machine
LSTM	Long Short-Term Memory
CNN	Convolutional Neural Networks
TF-IDF	Term Frequency–Inverse Document Frequency
Bi-LSTM	Bi-Directional Long Short-Term Memory
NLP	Natural Language Processing
IM	Invariant Mining
GAN	Generative Adversial Network
BERT	Bidirectional Encoder Representation from Transformers
RF	Random Forest
KNN	K-Nearest Neighbor
MLP	Multilayer Perceptron
RoBERTa	Robustly Optimized BERT Pretraining Approach
GRU	Gate Recurrent Unit
SVDD	Support Vector Data Description
GGNNs	Gated Graph Neural Networks

References

1. He, S.; He, P.; Chen, Z.; Yang, T.; Su, Y.; Lyu, M.R. A survey on automated log analysis for reliability engineering. *ACM Comput. Surv. (CSUR)* **2021**, *54*, 1–37. [CrossRef]
2. Broken Connection in Recent Deployment Causes Microsoft Teams Outage. Available online: https://techwireasia.com/2022/07/broken-connection-in-recent-deployment-causes-microsoft-teams-outage/ (accessed on 21 July 2022).
3. Google Outage Analysis: 9 August 2022. Available online: https://www.thousandeyes.com/blog/google-outage-analysis-august-9-2022 (accessed on 19 August 2022).
4. Hadem, P.; Saikia, D.K.; Moulik, S. An SDN-based intrusion detection system using SVM with selective logging for IP traceback. *Comput. Netw.* **2021**, *191*, 108015. [CrossRef]
5. Lou, P.; Lu, G.; Jiang, X.; Xiao, Z.; Hu, J.; Yan, J. Cyber intrusion detection through association rule mining on multi-source logs. *Appl. Intell.* **2021**, *51*, 4043–4057. [CrossRef]
6. Lin, Y.D.; Wang, Z.Y.; Lin, P.C.; Nguyen, V.L.; Hwang, R.H.; Lai, Y.C. Multi-datasource machine learning in intrusion detection: Packet flows, system logs and host statistics. *J. Inf. Secur. Appl.* **2022**, *68*, 103248. [CrossRef]
7. Awuson-David, K.; Al-Hadhrami, T.; Alazab, M.; Shah, N.; Shalaginov, A. BCFL logging: An approach to acquire and preserve admissible digital forensics evidence in cloud ecosystem. *Future Gener. Comput. Syst.* **2021**, *122*, 1–13. [CrossRef]
8. Dalezios, N.; Shiaeles, S.; Kolokotronis, N.; Ghita, B. Digital forensics cloud log unification: Implementing CADF in Apache CloudStack. *J. Inf. Secur. Appl.* **2020**, *54*, 102555. [CrossRef]
9. Cinque, M.; Della Corte, R.; Pecchia, A. Contextual filtering and prioritization of computer application logs for security situational awareness. *Future Gener. Comput. Syst.* **2020**, *111*, 668–680. [CrossRef]
10. Lupton, S.; Washizaki, H.; Yoshioka, N.; Fukazawa, Y. Literature Review on Log Anomaly Detection Approaches Utilizing Online Parsing Methodology. In Proceedings of the 2021 28th Asia-Pacific Software Engineering Conference (APSEC), Taipei, Taiwan, 6–9 December 2021; pp. 559–563.
11. Zhang, T.; Qiu, H.; Castellano, G.; Rifai, M.; Chen, C.S.; Pianese, F. System Log Parsing: A Survey. *IEEE Trans. Knowl. Data Eng.* **2023**, *early access*. [CrossRef]
12. Vaarandi, R.; Pihelgas, M. Logcluster—A data clustering and pattern mining algorithm for event logs. In Proceedings of the 2015 11th International Conference on Network and Service Management (CNSM), Barcelona, Spain, 9–13 November 2015; pp. 1–7.
13. Hamooni, H.; Debnath, B.; Xu, J.; Zhang, H.; Jiang, G.; Mueen, A. Logmine: Fast pattern recognition for log analytics. In Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, Indianapolis, IN, USA, 24–28 October 2016; pp. 1573–1582.
14. Du, M.; Li, F.; Zheng, G.; Srikumar, V. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 1285–1298.
15. Meng, W.; Liu, Y.; Zhu, Y.; Zhang, S.; Pei, D.; Liu, Y.; Chen, Y.; Zhang, R.; Tao, S.; Sun, P.; et al. LogAnomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In Proceedings of the IJCAI, Macao, China, 10–16 August 2019; Volume 19, pp. 4739–4745.
16. Zhang, X.; Xu, Y.; Lin, Q.; Qiao, B.; Zhang, H.; Dang, Y.; Xie, C.; Yang, X.; Cheng, Q.; Li, Z.; 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, Tallinn, Estonia, 26–30 August 2019; pp. 807–817.

17. Huang, S.; Liu, Y.; Fung, C.; He, R.; Zhao, Y.; Yang, H.; Luan, Z. Hitanomaly: Hierarchical transformers for anomaly detection in system log. *IEEE Trans. Netw. Serv. Manag.* **2020**, *17*, 2064–2076. [CrossRef]
18. Li, X.; Chen, P.; Jing, L.; He, Z.; Yu, G. Swisslog: Robust and unified deep learning based log anomaly detection for diverse faults. In Proceedings of the 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE), Coimbra, Portugal, 12–15 October 2020; pp. 92–103.
19. Xiao, T.; Quan, Z.; Wang, Z.J.; Zhao, K.; Liao, X. Lpv: A log parser based on vectorization for offline and online log parsing. In Proceedings of the 2020 IEEE International Conference on Data Mining (ICDM), Sorrento, Italy, 17–20 November 2020; pp. 1346–1351.
20. Fang, L.; Di, X.; Liu, X.; Qin, Y.; Ren, W.; Ding, Q. QuickLogS: A Quick Log Parsing Algorithm based on Template Similarity. In Proceedings of the 2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), Shenyang, China, 20–22 October 2021; pp. 1085–1092.
21. Du, M.; Li, F. Spell: Streaming parsing of system event logs. In Proceedings of the 2016 IEEE 16th International Conference on Data Mining (ICDM), Barcelona, Spain, 12–15 December 2016; pp. 859–864.
22. He, P.; Zhu, J.; Zheng, Z.; Lyu, M.R. Drain: An online log parsing approach with fixed depth tree. In Proceedings of the 2017 IEEE International Conference on Web Services (ICWS), Honolulu, HI, USA, 25–30 June 2017; pp. 33–40.
23. Wang, X.; Zhao, Y.; Xiao, H.; Wang, X.; Chi, X. Ltmatch: A method to abstract pattern from unstructured log. *Appl. Sci.* **2021**, *11*, 5302. [CrossRef]
24. Vervaet, A.; Chiky, R.; Callau-Zori, M. Ustep: Unfixed search tree for efficient log parsing. In Proceedings of the 2021 IEEE International Conference on Data Mining (ICDM), Auckland, New Zealand, 7–10 December 2021; pp. 659–668.
25. Fu, Y.; Yan, M.; Xu, J.; Li, J.; Liu, Z.; Zhang, X.; Yang, D. Investigating and improving log parsing in practice. In Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Singapore, 14–18 November 2022; pp. 1566–1577.
26. Huang, S.; Liu, Y.; Fung, C.; He, R.; Zhao, Y.; Yang, H.; Luan, Z. Paddy: An event log parsing approach using dynamic dictionary. In Proceedings of the NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, 20–24 April 2020; pp. 1–8.
27. Messaoudi, S.; Panichella, A.; Bianculli, D.; Briand, L.; Sasnauskas, R. A search-based approach for accurate identification of log message formats. In Proceedings of the 26th Conference on Program Comprehension, Gothenburg, Sweden, 27 May–3 June 2018; pp. 167–177.
28. Fournier-Viger, P.; Lin, J.C.W.; Kiran, R.U.; Koh, Y.S.; Thomas, R. A survey of sequential pattern mining. *Data Sci. Pattern Recognit.* **2017**, *1*, 54–77.
29. Luna, J.M.; Fournier-Viger, P.; Ventura, S. Frequent itemset mining: A 25 years review. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* **2019**, *9*, e1329. [CrossRef]
30. Vaarandi, R. A data clustering algorithm for mining patterns from event logs. In Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM 2003) (IEEE Cat. No. 03EX764), Kansas City, MO, USA, 3 October 2003; pp. 119–126.
31. Nagappan, M.; Vouk, M.A. Abstracting log lines to log event types for mining software system logs. In Proceedings of the 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010), Cape Town, South Africa, 2–3 May 2010; pp. 114–117.
32. Dai, H.; Li, H.; Chen, C.S.; Shang, W.; Chen, T.H. Logram: Efficient Log Parsing Using *n* n-Gram Dictionaries. *IEEE Trans. Softw. Eng.* **2020**, *48*, 879–892. [CrossRef]
33. Sedki, I.; Hamou-Lhadj, A.; Ait-Mohamed, O.; Shehab, M.A. An Effective Approach for Parsing Large Log Files. In Proceedings of the 2022 IEEE International Conference on Software Maintenance and Evolution (ICSME), Limassol, Cyprus, 3–7 October 2022; pp. 1–12.
34. Chu, G.; Wang, J.; Qi, Q.; Sun, H.; Tao, S.; Liao, J. Prefix-Graph: A Versatile Log Parsing Approach Merging Prefix Tree with Probabilistic Graph. In Proceedings of the 2021 IEEE 37th International Conference on Data Engineering (ICDE), Chania, Greece, 19–22 April 2021; pp. 2411–2422.
35. Tao, S.; Meng, W.; Cheng, Y.; Zhu, Y.; Liu, Y.; Du, C.; Han, T.; Zhao, Y.; Wang, X.; Yang, H. Logstamp: Automatic online log parsing based on sequence labelling. *ACM SIGMETRICS Perform. Eval. Rev.* **2022**, *49*, 93–98. [CrossRef]
36. Rand, J.; Miranskyy, A. On automatic parsing of log records. In Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER), Madrid, Spain, 25–28 May 2021; pp. 41–45.
37. Zhang, S.; Wu, G. Efficient Online Log Parsing with Log Punctuations Signature. *Appl. Sci.* **2021**, *11*, 11974. [CrossRef]
38. Zhu, J.; He, S.; Liu, J.; He, P.; Xie, Q.; Zheng, Z.; Lyu, M.R. Tools and benchmarks for automated log parsing. In Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), Montreal, QC, Canada, 25–31 May 2019; pp. 121–130.
39. He, S.; Zhu, J.; He, P.; Lyu, M.R. Experience report: System log analysis for anomaly detection. In Proceedings of the 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), Ottawa, ON, Canada, 23–27 October 2016; pp. 207–218.
40. Xie, Y.; Zhang, H.; Zhang, B.; Babar, M.A.; Lu, S. LogDP: Combining Dependency and Proximity for Log-Based Anomaly Detection. In Proceedings of the Service-Oriented Computing: 19th International Conference, ICSOC 2021, Virtual Event, 22–25 November 2021; Springer: Berlin/Heidelberg, Germany, 2021; pp. 708–716.

41. Zhao, N.; Wang, H.; Li, Z.; Peng, X.; Wang, G.; Pan, Z.; Wu, Y.; Feng, Z.; Wen, X.; Zhang, W.; et al. An empirical investigation of practical log anomaly detection for online service systems. In Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, 23–28 August 2021; pp. 1404–1415.
42. Lu, S.; Wei, X.; Li, Y.; Wang, L. Detecting anomaly in big data system logs using convolutional neural network. In Proceedings of the 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), Athens, Greece, 12–15 August 2018; pp. 151–158.
43. Yen, S.; Moh, M.; Moh, T.S. Causalconvlstm: Semi-supervised log anomaly detection through sequence modeling. In Proceedings of the 2019 18th IEEE International Conference On Machine Learning Furthermore, Applications (ICMLA), Boca Raton, FL, USA, 16–19 December 2019; pp. 1334–1341.
44. Bertero, C.; Roy, M.; Sauvanaud, C.; Trédan, G. Experience report: Log mining using natural language processing and application to anomaly detection. In Proceedings of the 2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE), Toulouse, France, 23–26 October 2017; pp. 351–360.
45. Wang, J.; Tang, Y.; He, S.; Zhao, C.; Sharma, P.K.; Alfarraj, O.; Tolba, A. LogEvent2vec: LogEvent-to-vector based anomaly detection for large-scale logs in internet of things. *Sensors* **2020**, *20*, 2451. [CrossRef]
46. Ying, S.; Wang, B.; Wang, L.; Li, Q.; Zhao, Y.; Shang, J.; Huang, H.; Cheng, G.; Yang, Z.; Geng, J. An improved KNN-based efficient log anomaly detection method with automatically labeled samples. *ACM Trans. Knowl. Discov. Data (TKDD)* **2021**, *15*, 1–22. [CrossRef]
47. Lv, D.; Luktarhan, N.; Chen, Y. ConAnomaly: Content-based anomaly detection for system logs. *Sensors* **2021**, *21*, 6125. [CrossRef]
48. Yang, H.; Zhao, X.; Sun, D.; Wang, Y.; Huang, W. Sprelog: Log-Based Anomaly Detection with Self-matching Networks and Pre-trained Models. In Proceedings of the Service-Oriented Computing: 19th International Conference, ICSOC 2021, Virtual Event, 22–25 November 2021; Springer: Berlin/Heidelberg, Germany, 2021; pp. 736–743.
49. Ryciak, P.; Wasielewska, K.; Janicki, A. Anomaly detection in log files using selected natural language processing methods. *Appl. Sci.* **2022**, *12*, 5089. [CrossRef]
50. Li, Y.; Du, N.; Bengio, S. Time-dependent representation for neural event sequence prediction. *arXiv* **2017**, arXiv:1708.00065.
51. Rak, T.; Żyła, R. Using Data Mining Techniques for Detecting Dependencies in the Outcoming Data of a Web-Based System. *Appl. Sci.* **2022**, *12*, 6115. [CrossRef]
52. Xiao, R.; Chen, H.; Lu, J.; Li, W.; Jin, S. AllInfoLog: Robust Diverse Anomalies Detection Based on All Log Features. *IEEE Trans. Netw. Serv. Manag.* **2022**, *early access*. [CrossRef]
53. Backes, M.; Humbert, M.; Pang, J.; Zhang, Y. walk2friends: Inferring social links from mobility profiles. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 1943–1957.
54. Dai, H.; Dai, B.; Song, L. Discriminative embeddings of latent variable models for structured data. In Proceedings of the International Conference on Machine Learning, PMLR, New York, NY, USA, 20–22 June 2016; pp. 2702–2711.
55. Xu, X.; Liu, C.; Feng, Q.; Yin, H.; Song, L.; Song, D. Neural network-based graph embedding for cross-platform binary code similarity detection. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 363–376.
56. Perozzi, B.; Al-Rfou, R.; Skiena, S. Deepwalk: Online learning of social representations. In Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 24–27 August 2014; pp. 701–710.
57. Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; Mei, Q. Line: Large-scale information network embedding. In Proceedings of the 24th International Conference on World Wide Web, Florence, Italy, 18–22 May 2015; pp. 1067–1077.
58. Grover, A.; Leskovec, J. node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; pp. 855–864.
59. Hacker, T.; Pais, R.; Rong, C. A markov random field based approach for analyzing supercomputer system logs. *IEEE Trans. Cloud Comput.* **2017**, *7*, 611–624. [CrossRef]
60. Zhao, X.; Rodrigues, K.; Luo, Y.; Yuan, D.; Stumm, M. Non-Intrusive Performance Profiling for Entire Software Stacks Based on the Flow Reconstruction Principle. In Proceedings of the Osdi, Savannah, GA, USA, 2–4 November 2016; pp. 603–618.
61. Milajerdi, S.M.; Gjomemo, R.; Eshete, B.; Sekar, R.; Venkatakrishnan, V. Holmes: Real-time apt detection through correlation of suspicious information flows. In Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 19–23 May 2019; pp. 1137–1152.
62. Liu, F.; Wen, Y.; Zhang, D.; Jiang, X.; Xing, X.; Meng, D. Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, London, UK, 11–15 November 2019; pp. 1777–1794.
63. Yang, W.; Gao, P.; Huang, H.; Wei, X.; Liu, W.; Zhu, S.; Luo, W. RShield: A refined shield for complex multi-step attack detection based on temporal graph network. In Proceedings of the Database Systems for Advanced Applications: 27th International Conference, DASFAA 2022, Virtual Event, 11–14 April 2022; Proceedings, Part III; Springer: Berlin/Heidelberg, Germany, 2022; pp. 468–480.

64. Zhang, C.; Peng, X.; Sha, C.; Zhang, K.; Fu, Z.; Wu, X.; Lin, Q.; Zhang, D. DeepTraLog: Trace-log combined microservice anomaly detection through graph-based deep learning. In Proceedings of the 44th International Conference on Software Engineering, Pittsburgh, PA, USA, 21–29 May 2022; pp. 623–634.
65. Guo, H.; Yuan, S.; Wu, X. Logbert: Log anomaly detection via bert. In Proceedings of the 2021 International Joint Conference on Neural Networks (IJCNN), Shenzhen, China, 18–22 July 2021; pp. 1–8.
66. Guo, Y.; Wu, Y.; Zhu, Y.; Yang, B.; Han, C. Anomaly detection using distributed log data: A lightweight federated learning approach. In Proceedings of the 2021 International Joint Conference on Neural Networks (IJCNN), Shenzhen, China, 18–22 July 2021; pp. 1–8.
67. Chen, R.; Zhang, S.; Li, D.; Zhang, Y.; Guo, F.; Meng, W.; Pei, D.; Zhang, Y.; Chen, X.; Liu, Y. Logtransfer: Cross-system log anomaly detection for software systems with transfer learning. In Proceedings of the 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE), Coimbra, Portugal, 12–15 October 2020; pp. 37–47.
68. Xia, B.; Yin, J.; Xu, J.; Li, Y. Loggan: A sequence-based generative adversarial network for anomaly detection based on system logs. In Proceedings of the Science of Cyber Security: Second International Conference, SciSec 2019, Nanjing, China, 9–11 August 2019; Revised Selected Papers 2; Springer: Berlin/Heidelberg, Germany, 2019; pp. 61–76.
69. Yang, L.; Chen, J.; Wang, Z.; Wang, W.; Jiang, J.; Dong, X.; Zhang, W. Semi-supervised log-based anomaly detection via probabilistic label estimation. In Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), Madrid, Spain, 25–28 May 2021; pp. 1448–1460.
70. Le, V.H.; Zhang, H. Log-based anomaly detection without log parsing. In Proceedings of the 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), Melbourne, Australia, 15–19 November 2021; pp. 492–504.
71. Lee, Y.; Kim, J.; Kang, P. LAnoBERT: System log anomaly detection based on BERT masked language model. *arXiv* **2021**, arXiv:2111.09564.
72. Vervaet, A. MoniLog: An Automated Log-Based Anomaly Detection System for Cloud Computing Infrastructures. In Proceedings of the 2021 IEEE 37th International Conference on Data Engineering (ICDE), Chania, Greece, 19–22 April 2021; pp. 2739–2743.
73. He, S.; Zhu, J.; He, P.; Lyu, M.R. Loghub: A large collection of system log datasets towards automated log analytics. *arXiv* **2020**, arXiv:2008.06448.
74. Landauer, M.; Skopik, F.; Höld, G.; Wurzenberger, M. A User and Entity Behavior Analytics Log dataset for Anomaly Detection in Cloud Computing. In Proceedings of the 2022 IEEE International Conference on Big Data (Big Data), Naples, Italy, 5–7 December 2022; pp. 4285–4294.