A Large-Scale Evaluation for Log Parsing Techniques: How Far Are We?

Zhihan Jiang
The Chinese University of Hong Kong
Hong Kong, China
zhjiang@link.cuhk.edu.hk

Jinyang Liu
The Chinese University of Hong Kong
Hong Kong, China
jyliu@cse.cuhk.edu.hk

Junjie Huang
The Chinese University of Hong Kong
Hong Kong, China
junjayhuang@outlook.com

Yichen Li
The Chinese University of Hong Kong
Hong Kong, China
ycli21@cse.cuhk.edu.hk

Yintong Huo
The Chinese University of Hong Kong
Hong Kong, China
ythuo@cse.cuhk.edu.hk

Jiazhen Gu
The Chinese University of Hong Kong
Hong Kong, China
jiazhengu@cuhk.edu.hk

Zhuangbin Chen*
Sun Yat-sen University
Zhuhai, China
chenzhb36@mail.sysu.edu.cn

Jieming Zhu
Huawei Noah's Ark Lab
Shenzhen, China
jiemingzhu@ieee.org

Michael R. Lyu
The Chinese University of Hong Kong
Hong Kong, China
lyu@cse.cuhk.edu.hk

ABSTRACT

Log data have facilitated various tasks of software development and maintenance, such as testing, debugging and diagnosing. Due to the unstructured nature of logs, log parsing is typically required to transform log messages into structured data for automated log analysis. Given the abundance of log parsers that employ various techniques, evaluating these tools to comprehend their characteristics and performance becomes imperative. Loghub serves as a commonly used dataset for benchmarking log parsers, but it suffers from limited scale and representativeness, posing significant challenges for studies to comprehensively evaluate existing log parsers or develop new methods. This limitation is particularly pronounced when assessing these log parsers for production use. To address these limitations, we provide a new collection of annotated log datasets, denoted Loghub-2.0, which can better reflect the characteristics of log data in real-world software systems. Loghub-2.0 comprises 14 datasets with an average of 3.6 million log lines in each dataset. Based on Loghub-2.0, we conduct a thorough re-evaluation of 15 state-of-the-art log parsers in a more rigorous and practical setting. Particularly, we introduce a new evaluation metric to mitigate the sensitivity of existing metrics to imbalanced data distributions. We are also the first to investigate the granular performance of log parsers on logs that represent rare system events, offering in-depth details for software diagnosis. Accurately parsing such logs is essential, yet it remains a challenge. We believe this work could shed light on the evaluation and design of log parsers in practical settings, thereby facilitating their deployment in production systems.

CCS CONCEPTS

• **Software and its engineering** → *Maintaining software.*

KEYWORDS

benchmark, empirical study, log parsing, log analysis

ACM Reference Format:
Zhihan Jiang, Jinyang Liu, Junjie Huang, Yichen Li, Yintong Huo, Jiazhen Gu, Zhuangbin Chen, Jieming Zhu, and Michael R. Lyu. 2024. A Large-Scale Evaluation for Log Parsing Techniques: How Far Are We?. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '24), September 16–20, 2024, Vienna, Austria.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3650212.3652123

1 INTRODUCTION

Log data records software runtime information, which is essential for developers to understand the behaviors of software systems. The rich information encapsulated within log data empowers developers and maintainers to test programs [4, 7, 8, 41], identify bugs [3, 6, 14, 59] and diagnose softwares [44–46]. In general, log messages are semi-structured textual data, generated by logging statements written by developers in the source code, *e.g.*,'`logger.info("connected to host: {}", hostIp)`' in Java [29, 30, 33, 61]. At runtime, the variable `hostIp` may change in different executions, which can result in a sequence of log messages like '`connected to host: 172.16.254.1`' and '`connected to host: 172.16.254.2`'. Log parsing aims to convert such semi-structured log messages into structured events, which often serves as the first and foremost step to many log analysis tasks [18, 27, 32, 37, 60]. Specifically, log parsing extracts the constant parts (*i.e.*, *log templates*) and the changeable parts (*i.e.*, *log parameters*) from log messages. In the above example, the log template is '`connected to host: <*>`', and the log parameter indicates the concrete IP address of the host, *e.g.*,'`172.16.254.1`'.

Traditional approaches parse logs via matching raw log messages with their respective logging statements within the source

[*] Zhuangbin Chen is the corresponding author.

ISSTA '24, September 16–20, 2024, Vienna, Austria

ACM ISBN 979-8-4007-0612-7/24/09
https://doi.org/10.1145/3650212.3652123

code [5, 47, 48, 50]. However, this approach is usually impractical since software source code may not always be available, *e.g.*, commercial software. Thus, tremendous efforts have been devoted to data-driven approaches [17, 21, 23, 58]. These parsers directly process raw log messages without access to the source code.

Given the variety of log parsers employing different techniques, it is crucial to evaluate these tools to comprehend their characteristics and performance, providing guidance for production adoption in industry. To this end, Zhu et al. [62] released Loghub, which contains an extensive collection of log datasets generated by various systems. However, Loghub only provides annotated parsing ground truth for 2,000 lines of logs randomly sampled from each system, denoted as Loghub-2k, which has been extensively used to evaluate existing log parsers [25, 63] and develop new log parsers [9, 10, 31, 55].

Even though existing log parsers, such as Drain [17], Logram [9] and LogPPT [28], have reported state-of-the-art results on the Loghub-2k dataset, we observed that the parsing performance of these tools is compromised when being integrated into real-world software systems [15, 49]. Based on our experiences of deploying automated log parsing in production, we have found that existing parsers struggle to identify two types of log templates, *i.e.*, *infrequent log templates* (those that occur infrequently) and *parameter-intensive log templates* (those that involve many parameters). The former usually includes logs with severe logging levels (*e.g.*, error or fatal), which typically demand more attention due to their potential impact. The latter usually records the system runtime status and associated values. These two types of log templates are crucial for downstream analysis tasks, such as anomaly detection and debugging. Therefore, it is essential to ensure parsing accuracy for such log templates. However, the research results reported in previous studies [25, 63] may not necessarily apply in practical production settings, especially for these two types of log data.

This performance disparity primarily originates from three inherent limitations in existing benchmark studies. First, the widely-used Loghub-2k is of limited scale. It only encompasses 2,000 lines of log messages in each dataset, whereas real-world data often consist of millions of log lines [55, 56, 63]. As a result, the Loghub-2k may not be able to sufficiently represent the complex characteristics of log data obtained from production systems, particularly in terms of the frequency and parameter count of log templates. Second, evaluation metrics used in existing benchmarks (*e.g.*, group accuracy, GA [63]) are often *message-level* (*i.e.*, calculated based on the number of log messages), thus may produce misleadingly high-accuracy results. It is because the distribution of log templates' occurrences is usually highly imbalanced in production systems [25, 55]. The evaluation results could be dominated by the majority classes of log templates (*i.e.*, those contain many log messages). Therefore, such metrics may not be robust enough to datasets with diverse template distributions. Third, existing studies often report the performance of log parsers in processing the entire dataset. It is unclear how they perform when dealing with the above two types of log templates. The lack of fine-grained evaluation can lead to a limited understanding of how well a log parser handles these specific cases in practice.

To address these limitations, we propose a new log parsing benchmark tailored to evaluate log parsers in a more rigorous and practical setting. Specifically, (1) On top of the raw Loghub logs [20], we build a new version of large-scale annotated log datasets for log parsing, denoted as Loghub-2.0. The annotation is conducted adhering to a rigorous framework, which can significantly reduce manual efforts through log grouping and template matching. Loghub-2.0 aims to reflect the scale and distribution of log data observed in real-world scenarios. In detail, Loghub-2.0 contains 14 datasets from various software systems. Each dataset contains *3.6 million* lines of log messages on average. Each log line has been manually annotated with its corresponding log template and parameter(s). (2) We propose a more comprehensive benchmarking protocol to evaluate existing log parsers. The protocol includes a new *template-level* metric, *i.e.*, F1-score of Group Accuracy (FGA), to mitigate the sensitivity of the message-level metrics (*e.g.*, GA) to imbalanced data. Moreover, we make the first step to investigate the performance of log parsers on log templates with different frequencies and parameter counts, providing an essential reference regarding how well they would perform in production environments. (3) We conduct an extensive re-evaluation of *15* log parsers, including *13* statistic-based and *2* semantic-based log parsers, on Loghub-2.0 using the proposed benchmarking protocol. Our study provides researchers and practitioners with a more practical perspective on understanding the characteristics of these parsers. We summarize the key findings from the evaluation as follows.

Key Findings. (1) Compared with Loghub-2k, Loghub-2.0 exhibits more realistic data characteristics, especially in the context of log template frequencies and parameter counts. (2) All existing parsers demonstrate a significant degradation in performance on Loghub-2.0 compared to the Loghub-2k, with a greater degree of variance. This shows that our proposed datasets and benchmarking protocol can reveal the performance of log parsers under more complex and diverse conditions. (3) Achieving high overall performance on the entire datasets does not necessarily guarantee effective parsing of infrequent and parameter-intensive logs, which often deserve more attention in system maintenance. Thus, a comprehensive evaluation should consider different types of logs to ensure robust and reliable performance in practice. (4) *9* out of *15* parsers fail to process all the datasets in Loghub-2.0 within a reasonable time frame, highlighting the importance of improving parsing efficiency, especially for production deployment.

The main contributions of this paper are summarized as follows:

- We propose a new collection of large-scale datasets for evaluating log parsing techniques, referred to as Loghub-2.0. This collection comprises 14 datasets, each with an average of 3.6 million log lines. The parsing labels of the log messages are manually annotated through a rigorous annotation framework, which ensures the efficiency and accuracy of the labeling process. This is a significant extension of the existing widely-used Loghub-2k, which contains only 2,000 lines of log messages per dataset.
- We propose a more comprehensive benchmarking protocol for log parsers, which emphasizes assessing parsing accuracy on logs with different characteristics. Moreover, a new template-level metric, *i.e.*, FGA, is proposed to address the sensitivity of existing metrics to imbalanced data.
- We re-evaluate 15 state-of-the-art log parsers by our benchmarking protocol and derive seven interesting findings, which could shed light on the design and evaluation of log parsers in a more practical setting. To benefit future research, we make datasets, source code, and experimental results publicly available [1].

2 BACKGROUND AND MOTIVATION

In this section, we first briefly introduce existing log parsers in the literature. Then, we talk about the observations that motivate us to revisit existing log parsing studies.

2.1 Existing Log Parsing Techniques

Many log parsing approaches have been proposed in the literature, mainly classified into the following four categories.

Frequency-based Parsing This type of methods [9, 43, 53, 54] is founded on the intuition that tokens, which frequently occur within a specific log dataset, generally represent the static elements of those logs. Consequently, the extraction of frequent patterns provides a straightforward approach for automated log parsing. In detail, these log parsers first traverse the provided log dataset to construct frequent itemsets. Subsequently, these itemsets are utilized to derive the corresponding log template for log messages.

Similarity-based Parsing These log parsers [13, 16, 42, 51, 52] conceptualize log parsing as clustering logs into distinct clusters predicated on their similarity, and logs in each cluster share the same log template. Various methods employ different clustering algorithms (*e.g.*, hierarchical clustering, density-based clustering) and definitions of similarity. Following the clustering process, log templates can be derived by extracting the common tokens from the logs within each respective cluster.

Heuristics-based Parsing Another category of log parsers [11, 17, 24, 39, 40, 58] employs a diverse range of heuristic algorithms or data structures, such as the longest common subsequence-based approach, parsing trees, evolutionary algorithm, among others. These log parsers are designed to leverage the unique characteristics of log data to distinguish the templates and parameters in log messages.

Semantic-based Parsing In recent years, numerous parsers [21, 28, 31, 35] have employed deep neural networks to understand the semantic meaning of logs, thereby improving parsing accuracy. In detail, these log parsers employ supervised methodologies, utilizing models such as bidirectional long short-term memory or pre-trained language models to learn the semantic information of log messages, thereby distinguishing between log templates and parameters through the completion of classification tasks.

2.2 Motivation

Given the fruitful log parsing studies, comprehensively evaluating existing log parsers is crucial in understanding their characteristics and guiding the selection of appropriate methods in practice. Zhu et al. [63] proposed the first benchmark of 13 log parsers by collecting multiple log datasets from various types of systems, including distributed systems, supercomputer systems, etc. Particularly, they randomly sampled 2,000 log messages for each dataset and manually annotated the template for these logs. This results in the widely-used collection of log parsing datasets, *i.e.*, Loghub-2k. Many new log parsing approaches [9, 28, 55] also evaluate their performance on Loghub-2k and demonstrate promising results.

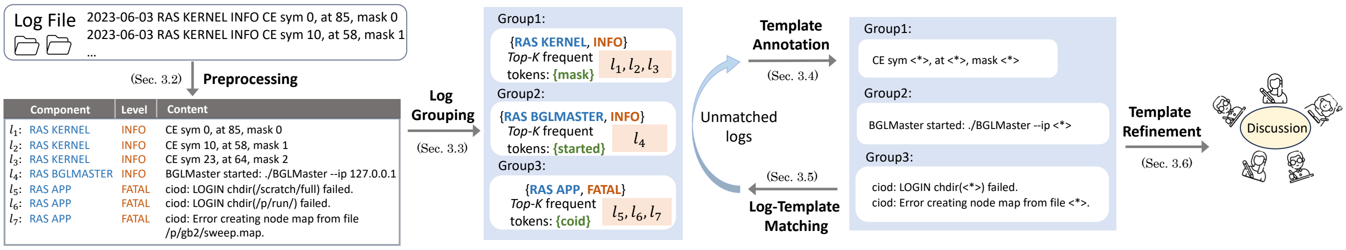
Despite the advantages of the dataset, we still observe some inherent limitations associated with it. First, a recent study [25] has pinpointed multiple errors in the annotated templates, which could potentially impact the assessment of log parsers. Therefore, they proposed several heuristic rules, such as regular expressions, to fix the incorrect templates in Loghub-2k. Second, we find that these log parsers demonstrate compromised effectiveness and efficiency in production deployment. This highlights the limitations of previous benchmark studies, as they do not fully capture the comprehensive performance of log parsers, especially in practical environments. To understand the aforementioned limitations, we have conducted a thorough investigation and identified three primary reasons:

- Loghub-2k is small in scale, with each dataset comprising only 2,000 lines of log messages. Considering that real-world systems often produce a large volume of log data (*e.g.*, tens of gigabytes per hour [19, 36, 55]), Loghub-2k may not be able to reflect the diverse and complex characteristics of log data observed in production environments. Given the data-driven nature of most existing log parsers, their performance could be affected by the limited scale of Loghub-2k and may not generalize well to real-world scenarios with much larger and diverse log datasets. Moreover, the annotation process of Loghub-2k does not follow a rigorous and standardized approach, leading to potential errors and inconsistencies in the annotated templates.
- Existing studies often lack a comprehensive set of metrics for evaluation. Most of them only employ message-level metrics such as group accuracy [63] and parsing accuracy [9]. Theoretically, these metrics tend to favor frequently occurring log templates. For instance, if a simple template involves a large number of log messages, then successfully parsing this template would yield good performance, irrespective of the results on the less frequent templates. In real-world scenarios, log data could be highly imbalanced. For example, some systems periodically print log messages to record routine information, such as system heartbeats (“`System uptime: 30 hours`”). Such logs might not be of interest to system operators. However, they could dominate the overall performance, masking potential errors in processing infrequent templates.
- The current evaluation of log parsers mainly reports the overall performance on entire datasets. This approach, however, lacks a fine-grained analysis of parsing performance for logs with different characteristics. We have identified two critical types of logs that play an essential role in production system maintenance. These include infrequent log templates, which represent rare system events that require particular attention, and parameter-intensive log templates, which provide informative details about system status and the associated entities. Investigating the performance on these logs helps understand the actual effectiveness of log parsers in real-world applications.

To address these limitations, we propose conducting a new benchmark study for existing log parsers in a more rigorous and practical setting. This entails the creation of a more diverse collection of log parsing datasets that are substantially larger in scale, as well as the design of a more comprehensive benchmarking protocol.

3 DATASET CONSTRUCTION

In this section, we introduce the construction process of the dataset collection, which intends to reflect the scale and characteristics of real-world log data and thus enables an accurate assessment of log parsers' capabilities in practical scenarios. Particularly, we propose a rigorous annotation framework designed to ensure the efficiency, accuracy and consistency of the labeling procedure.



```
2023-06-03 RAS KERNEL INFO CE sym 0, at 85, mask 0
2023-06-03 RAS KERNEL INFO CE sym 10, at 58, mask 1
...
```

	Component	Level	Content
l_1:	RAS KERNEL	INFO	CE sym 0, at 85, mask 0
l_2:	RAS KERNEL	INFO	CE sym 10, at 58, mask 1
l_3:	RAS KERNEL	INFO	CE sym 23, at 64, mask 2
l_4:	RAS BGLMASTER	INFO	BGLMaster started: ./BGLMaster --ip 127.0.0.1
l_5:	RAS APP	FATAL	ciod: LOGIN chdir(/scratch/full) failed.
l_6:	RAS APP	FATAL	ciod: LOGIN chdir(/p/run/) failed.
l_7:	RAS APP	FATAL	ciod: Error creating node map from file /p/gb2/sweep.map.

Group1:
{RAS KERNEL, INFO}
Top-K frequent tokens: {mask} l_1, l_2, l_3

Group2:
{RAS BGLMASTER, INFO}
Top-K frequent tokens: {started} l_4

Group3:
{RAS APP, FATAL}
Top-K frequent tokens: {coid} l_5, l_6, l_7

Group1:

```
CE sym <*>, at <*>, mask <*>
```

Group2:

```
BGLMaster started: ./BGLMaster --ip <*>
```

Group3:

```
ciod: LOGIN chdir(<*>) failed.
ciod: Error creating node map from file <*>.
```

Figure 1: The overall framework of data annotation

3.1 Overview

To construct the datasets, we select 14 log datasets in Loghub [20] that span different types of systems, including distributed systems, supercomputers, and operating systems. Although these datasets are collected from various types of systems on a large scale, they lack the essential labels for log parsing assessment. Thus, a necessary preliminary step in our study is annotating these datasets.

The annotation process is carried out by a team of five skilled data annotators. This team consists of three Ph.D. students with a minimum of two years of experience in system maintenance research, alongside two industry engineers, both of whom have at least five years of experience in software development and management. Given the immense size of each dataset (*e.g.*, millions of log messages), manual labeling for each log message is infeasible. Therefore, we design a rigorous annotation framework to assist the annotation process, which guarantees both labeling efficiency and accuracy through log grouping and template matching.

Fig. 1 shows the overview of the annotation framework, which includes five steps: *preprocessing*, *log grouping*, *template annotation*, *log-template matching*, and *template refinement*. To begin, we first preprocess the raw logs to obtain meaningful log contents. Then, we apply a hierarchical approach to coarsely partition the logs into distinct groups. The logs sharing the same template are highly likely to be divided into the same group, facilitating efficient annotation procedures. Within each group, all annotators carefully identify all log templates. In this process, we arrange log messages within each group in lexicographical order to place similar log messages together, which enables us to quickly annotate all log templates instead of labeling each log message. After the annotation, we employ regular expressions to construct the matching between log messages and the labeled log templates. If any log messages remain unmatched, we review and rectify the templates, subsequently repeating the matching process until all log messages are matched. Finally, we conduct template refinement to calibrate the results of all annotators to ensure the accuracy and uniformity of the annotations across all annotators and datasets. The details of each step are explained as follows.

3.2 Preprocessing

Following previous work [17, 55, 63], we first apply predefined regular expressions to extract different fields of log messages. Typical fields include timestamp, logging level, component, and content. We then undertake a cleaning process for the logs. This process specifically targets logs whose content does not include any alphabetical characters, such as logs comprised solely of numerical figures or punctuation marks. Such logs are cleaned due to their lack of parseable content. We also remove log lines with duplicate content temporarily to reduce manual efforts in the following steps.

3.3 Log Grouping

After the preprocessing stage, we are still facing a substantial number of log messages (*e.g.*, millions of log messages in the HDFS dataset), making it unfeasible to manually annotate each message. Inspired by [34], we adopt a hierarchical approach to coarsely divide log messages into multiple groups. Our goal is to group together log messages sharing the same template, which enables us to label their template in one pass. To this end, we first partition logs based on their logging level and component name, which are extracted during the preprocessing step. These two properties provide a straightforward means to initially identify logs that belong to the same template [34]. Second, we use more advanced information to group logs, *i.e.*, the most frequently occurring tokens of a log message. Specifically, we employ delimiters such as spaces and punctuation to tokenize each log message into multiple tokens and calculate the frequency of each token in the dataset. For each log message, we calculate the K most frequent tokens and then group together those log messages that share the common top-K frequent tokens. The underlying rationale is that the template part of log messages remains stable, while the parameter part can dynamically change during runtime. As a result, the most frequent K tokens in log messages can effectively serve as robust evidence to determine their belonging to the same group. The value of K for each dataset has been determined based on their characteristics, with the value ranging from one to three. Particularly, we maintain a collection of stop words to be excluded from the top-K frequent tokens, ensuring that these common words do not interfere with the grouping process. In addition to the stop words provided by the Scipy library [2], we have also manually added certain words to the collection, such as `root`, `true`, etc. Finally, we obtain multiple coarse-grained groups of log messages where the log messages in each group share the same logging level, component, and top-K tokens.

3.4 Template Annotation

The objective of the log grouping step is to partition the logs into coarse-grained groups, ensuring, as far as possible, that logs sharing the same template are divided into the same group. Therefore, it is possible that log messages with different templates are grouped together. To address this issue, we employ manual template annotation to derive ground-truth log templates from each group.

To accelerate the manual annotation process, we sort the log messages within each group in lexicographical order to place similar log messages together. Annotators can then quickly recognize the templates of logs based on their structures and similarities. Consequently, annotators are required to annotate only the ground-truth log templates, eliminating the need for sequential labeling of each log message. This approach is based on the observation that the quantity of potential log templates is typically several orders of magnitude smaller than the total number of log messages [25, 55, 63], which renders manual labeling a feasible task.

In detail, all annotators conduct the manual annotation process independently, whose individual results will be consolidated to generate the final results (to be detailed in Sec. 3.6). To ensure the accuracy and consistency of the annotation, we adhere to the parameter categories proposed by Li et al. [31] to determine whether a token is a parameter. We also ask the annotators to apply the same heuristic rules proposed by Khan et al. [25] to ensure a more consistent template format, *e.g.*, replace double spaces with a single space. If a token is identified as a parameter, we will replace it with `"<*>"`, and the static parts remain unchanged to form the corresponding templates. Since similar log messages have been grouped and sorted together, we can efficiently bypass numerous log messages that evidently share the same template when handling each group, and only the identified templates are recorded. Finally, in the rare cases where different groups still share identical templates, we compare the templates derived from different groups, eliminate duplicate templates, and merge some templates as necessary. This procedure can eliminate potential errors in the log grouping step, thereby ensuring the accuracy of the annotations.

3.5 Log-Template Matching

While we generate log templates in the manual annotation step, we do not record the explicit matching between each log message and its corresponding template. This is to avoid complicating the annotation process, and later in the possible deduplication and merging procedures, it could lead to ambiguous relations and challenges in maintaining clarity and accuracy. Instead, we resort to the technique of regular expressions to automatically construct the matching between logs and templates.

Specifically, we convert each template into a regular expression by substituting `"<*>"` with `"(.*)"`, which enables each parameter position to match strings of any length. Subsequently, for each log message, we attempt to match it against every log template, halting when a match is found. Although this step requires pairwise matching between a large number of log messages and log templates, it can be completed within a reasonable time given that the number of log templates is typically much smaller than the total count of log messages (as shown in Table 1). We also further speed up this matching process by implementing it in a parallel manner.

Additionally, in this matching process, one log message could match multiple templates. For examples, two templates T_1: `"auth failure; logname=<*> uid=<*> ruser=<*>"` and T_2: `"auth failure; logname=<*> uid=<*>"` can exist in the same dataset. All log messages generated from template T_1 can be matched by T_2 since the last `<*>` are allowed to match multiple tokens. To address this issue, when a log message matches multiple regular expressions, we give priority to the templates with longer static parts for annotation. The intuition is that when two different templates are capable of matching the same log message, the template that can match more non `"<*>"` characters suggests a higher probability that this log message belongs to that particular template. In the rare cases where the two are the same, we choose templates with fewer `"<*>"` to generate more compact and simple templates. By applying this rule, the log messages of T_1 will be correctly matched with T_1.

In instances where specific log messages fail to match any regular expression, we will revert to the template annotation step, carefully review these log messages, make necessary template corrections, and subsequently repeat the matching process. Ultimately, each log message should successfully match one regular expression that corresponds to its annotated template.

3.6 Template Refinement

The last step is template refinement, which aims to consolidate all five annotators' results and correct potential errors. After carefully comparing the templates from five annotators, we identify the following inconsistent cases that occur most frequently. All discrepancies are addressed through discussions to ensure accurate and uniform annotation.

- One annotator may produce more templates than others. In this case, it is possible that some of his annotated templates are too specific. For example, some variables (*e.g.*,`root/True/temp`) are incorrectly identified as constant. We then regard such cases as parameters following [25].
- The same template may have different formats, *e.g.*,"`1165 bytes (1.13 KB) sent`" may be labeled as "`<*> bytes (<*> KB) sent`" or "`<*> bytes <*> sent`". In this case, we chose the former one to retain the original format of the log messages.

Additionally, we quantitatively assess the annotation consistency of five annotators. This is measured by determining the proportion of templates where the annotations of the five annotators are identical. The average consistency score across all datasets attains a value of 0.926, indicating a high agreement in the annotation step. Ultimately, all five annotators reached a consensus on all annotated templates, which is then adopted as the final annotation.

3.7 Annotation Results

The data annotation process finally produces a collection of large-scale log paring datasets from diverse systems, named Loghub-2.0. The detailed statistic of Loghub-2.0 is presented in Table 1. Compared with Loghub-2k, the average number of annotated log messages has seen a substantial increase, escalating by a factor of 1875, from 2,000 to 3,601,187. Furthermore, the average number of annotated log templates has increased by 204.2%, from 81.9 to 249.1, encompassing a broader range of templates. The large scale of the new dataset collection, Loghub-2.0, enables detailed evaluations of log parsing techniques, potentially exposing their performance in more realistic and large-scale scenarios.

4 STUDY DESIGN

In this section, we introduce the design of our benchmark study for log parsers. Based on the large scale and diversity of Loghub-2.0, we aim to gain a more in-depth understanding of the log parsers' effectiveness and suitability for real-world applications. To this end, we first design three research questions to guide the study. Then, we select a new set of metrics for comprehensive performance assessment, which includes a new template-level metric that we design and existing popular metrics. Finally, we elaborate on the selected log parsers for evaluation and the experiment setup.

Table 1: Statistics of Loghub-2.0

System	Dataset	# Templates (Loghub-2k)	# Templates (Loghub-2.0)	# Annotated Logs (Loghub-2.0)
Distributed systems	Hadoop	114	236	179,993
	HDFS	14	46	11,167,740
	OpenStack	43	48	207,632
	Spark	36	236	16,075,117
	Zookeeper	50	89	74,273
Super-computer systems	BGL	120	320	4,631,261
	HPC	46	74	429,987
	Thunderbird	149	1,241	16,601,745
Operating systems	Linux	118	338	23,921
	Mac	341	626	100,314
Server application	Apache	6	29	51,977
	OpenSSH	27	38	638,946
Standalone software	HealthApp	75	156	212,394
	Proxifier	8	11	21,320
Average		**81.9**	**249.1**	**3,601,187**

4.1 Research Question

RQ1: What are the differences between Loghub-2.0 and Loghub-2k? In this RQ, we aim to explore whether there are significant differences in the characteristics of Loghub-2.0 and Loghub-2k, which could potentially impact the performance of log parsers. Specifically, We focus on examining two important characteristics: *frequencies of log templates* and *parameter counts in log templates*.

RQ2: How does the performance of log parsers differ when applied to Loghub-2.0 compared to Loghub-2k? In this RQ, our focus lies in conducting a comprehensive re-evaluation of log parsers using Loghub-2.0, encompassing both effectiveness and efficiency aspects. We also explore any potential limitations associated with the widely-used Loghub-2k. To this end, we carefully compare the evaluation results obtained from Loghub-2.0 with those from Loghub-2k, enabling us to draw insightful conclusions.

RQ3: What is the performance of log parsers on logs with varying characteristics? Inspired by our observations in Sec. 2.2, we investigate the performance of log parsers on logs with diverse template frequencies and parameter counts. This is pivotal, as certain logs with distinctive characteristics may hold significant importance in production environments. Particularly, such an evaluation becomes feasible only with the use of the labeled datasets in Loghub-2.0, attributable to its large scale and diversity.

4.2 Evaluation Metrics

We employ two categories of metrics, *i.e.*, message-level and template-level metrics, to evaluate log parsers. *Message-level metrics* account for the quantities of messages belonging to each template, thereby favoring templates with a higher volume of log messages. On the other hand, *template-level metrics* evenly consider each template, regardless of the number of log messages each template corresponds to. In our benchmark protocol, we adopt two message-level metrics *i.e.*, Group Accuracy (GA) and Parsing Accuracy (PA) and two template-level metrics *i.e.*, F1-score of Group Accuracy (FGA) and the F1-score of Template Accuracy (FTA) [25]. In particular, FGA, proposed by us, is the template-level version of GA. Below, we elaborate on the metrics used in our study.

4.2.1 *Message-Level Metrics.*

Following existing studies, we utilize two popular message-level metrics, *i.e.*, GA and PA.

Group Accuracy (GA). GA is first used by Zhu et al. [63], which assesses the ability to correctly group log messages belonging to the same template. It is defined as *the proportion of correctly grouped log messages to the total number of log messages.* A log message is regarded as correctly grouped if and only if its template corresponds to the same set of log messages as the ground truth does.

Parsing Accuracy (PA). PA utilized by Dai et al. [9] assesses the ability to correctly extract the template parts and parameter parts for each log message, which is essential for various log analysis tasks, such as anomaly detection using parameter values [12, 22, 26]. It is defined as *the ratio of correctly parsed log messages over the total number of log messages*, where a log message is considered to be correctly parsed if and only if all tokens of static templates and dynamic variables are correctly identified.

4.2.2 *Template-Level Metrics.*

Despite the wide use of message-level metrics [31, 57, 58], they consider the number of log messages and thus are sensitive to imbalanced templates. For example, in a dataset where 95% of log messages belong to only 1% of the templates, a log parser could achieve a GA or PA of 0.95 by accurately grouping or parsing these 1% of templates, regardless of any parsing errors for the remaining 99% of templates. In practice, certain infrequently occurring templates, such as error-level log messages, may hold crucial significance, while frequently appearing templates, like heartbeat messages, might be of less importance. Thus, template-level metrics, which do not consider the number of log messages of each template, should also be incorporated to comprehensively evaluate the performance of log parsers.

F1-score of Group Accuracy (FGA). We propose FGA, which focuses on the proportion of correctly grouped templates rather than log messages. Thus, it can be considered as calculating GA at the template level. Specifically, FGA is the harmonic mean of PGA (Precision of Group Accuracy) and RGA (Recall of Group Accuracy). Let N_p be the number of templates that are generated by a log parser, and N_c be the number of templates that are correctly parsed by the log parser. The correctness here has the same definition as in GA, *i.e.*, a log template is considered as correctly parsed if and only if the set of log messages belonging to this template matches the set indicated in the ground truth. N_g is the actual correct number of templates in the ground truth. Based on these notations, we can define PGA as $\frac{N_c}{N_p}$ and RGA as $\frac{N_c}{N_g}$. Then, we can calculate FGA as their harmonic mean, *i.e.*, $\frac{2 \times PGA \times RGA}{PGA + RGA}$

F1-score of Template Accuracy (FTA). FTA is the harmonic mean of RTA (Recall of Template Accuracy) and PTA (Precision of Template Accuracy) proposed by Khan et al. [25]. FTA has a different definition of "correct identification" from FGA, and we define a new notation $\hat{N_c}$ to represent the number of templates that are correctly identified by a log parser. For FTA, one template is regarded as correctly identified if and only if these two conditions hold: (1) the parsed template's corresponding set of log messages share the same ground-truth template; (2) all the tokens of the template are the same as those of the ground-truth template. Then, we can define PTA as $\frac{\hat{N_c}}{N_p}$ and RTA as $\frac{\hat{N_c}}{N_g}$. And FTA can be calculated as their harmonic mean, *i.e.*, $\frac{2 \times PTA \times RTA}{PTA + RTA}$. FTA focuses more on the ability to identify concrete constant and parameter parts for a particular log message in comparison to FGA.

4.3 Evaluation Setup

For our evaluation, we carefully select 15 state-of-the-art log parsers from the literature. Thirteen of them have been previously evaluated by Khan et al. [25]. Different from their work, we exclude LKE [13] from our evaluation, which involves the computation of pair-wise distances, rendering it impractical for large-scale scenarios. These log parsers are all statistic-based, using techniques that are based on frequency (*i.e.*, LFA [43], LogCluster [54], Logram [9], SLCT [53]), similarity (*i.e.*, LenMa [51], LogMine [16], LogSig [52]), and heuristics (*i.e.*, AEL [24], Drain [17], IPLoM [39], MoLFI [40], SHISO [42], Spell [11]). For implementation, we directly reuse the source codes released by previous work [25, 63]. Moreover, we have incorporated two semantic-based log parsers, namely UniParser [35] and LogPPT [28], into our benchmarking study. We implement the UniParser model following the details provided in its corresponding paper and reuse the source code of LogPPT.

In our evaluation, we apply the same preprocessing rules (*e.g.*, regular expressions) and fine-tune the parameter settings through multiple runs of each log parser. For log parsers that exhibit variability in their parsing results due to inherent randomness, *e.g.*, MoLFI and LogPPT, we perform the evaluation five times. By reporting the median result, we aim to mitigate potential biases arising from such randomness and present a more reliable assessment of their performance. All experiments were conducted on a server equipped with an Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz, 256GB RAM, and an NVIDIA GeForce GTX3090, running Ubuntu 16.04.7 LTS.

5 STUDY RESULTS

5.1 RQ1: Differences between Loghub-2.0 and Loghub-2k

In this RQ, we aim to investigate the difference in data characteristics between Loghub-2.0 and Loghub-2k. As previously indicated in Table 1, Loghub-2.0 significantly surpasses Loghub-2k in terms of the sizes of log messages and templates, with approximately 1,900 times more messages and 3 times more templates on average. This substantial increase might imply a significant distinction in feature distribution across these two dataset collections.

As discussed in Sec. 2.2, there are two pivotal characteristics inherent to log datasets: the frequency of templates and the parameter count of templates. Specifically, the frequency of a log template refers to the number of log messages belonging to a specific log template. The parameter count of a log template is the number of different dynamic parts within each log template, *i.e.*, the number of "<*>" symbols in a log template. We calculate the distribution of these two characteristics for each dataset in Loghub-2k and Loghub-2.0, and plot the corresponding cumulative distribution function diagrams. Due to space constraints, we only present three representative datasets in Loghub-2.0, *i.e.*, Spark, Linux, and OpenSSH. The figures for all 14 datasets are available at our repository [1].

The distribution of templates' frequencies The figures on the left-hand side of Fig. 2 depict the distribution of template frequencies across three datasets. On the one hand, Loghub-2.0 exhibits a broader range of template frequencies, *e.g.*, in Spark of Loghub-2.0, the template frequencies range from 1 to over 10^6, while in the Loghub-2k, the range is narrower, ranging from 1 to around 10^3. On the other hand, the long-tail distribution of Loghub-2.0 is more pronounced than that of Loghub-2k, indicating more imbalanced template frequencies. For example, in the Spark dataset of Loghub-2.0, only 10% of the templates have frequencies exceeding 10^4, yet these few templates constitute the majority of the logs.

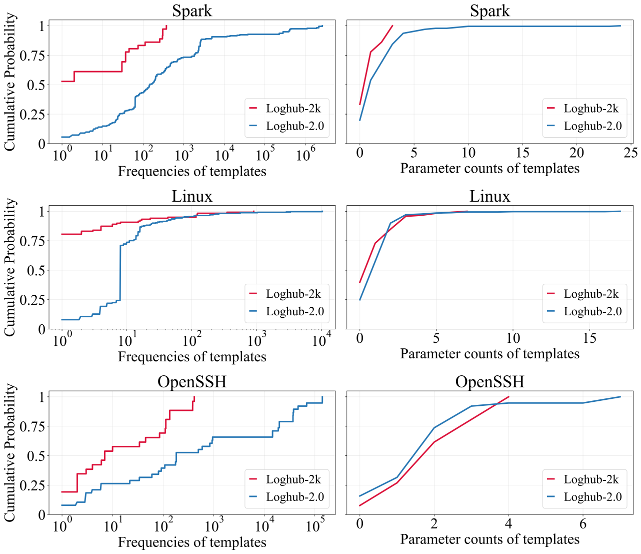


Figure 2: Distribution comparison of template frequencies and parameter counts in Loghub-2.0 and Loghub-2k

The distribution of templates' parameter counts The three figures on the right-hand side of Fig. 2 present the distribution of templates' parameter counts. We can observe that Loghub-2.0 covers a wider array of templates, each with a significantly higher number of parameters compared to those in Loghub-2k. For example, the maximum number of parameters of Spark's log templates is 3 in Loghub-2k. However, this number rises to 24 in the case of Loghub-2.0. A similar trend is observed for both Linux and OpenSSH, suggesting that Loghub-2.0 has more complex log templates. This complexity presents a greater challenge for a log parser in accurately identifying an increased number of parameters.

> **Finding 1.** The distributions of log template frequencies and parameter counts in Loghub-2k and Loghub-2.0 exhibit significant differences. Loghub-2.0, in particular, exhibits a more pronounced imbalance in template frequencies. Additionally, Loghub-2.0 contains a larger number of templates, and each has a larger parameter count on average compared to those in Loghub-2k.

5.2 RQ2: Performance differences of log parsers on Loghub-2.0 and Loghub-2k

Given the differences in characteristics observed between Loghub-2k and Loghub-2.0 in RQ1, the potential impact of such differences on the performance of log parsers remains unclear. To address this, we apply the selected 15 log parsers to Loghub-2.0 and compare the performance with Loghub-2k [25, 63] in terms of effectiveness and efficiency. Specifically, we apply the same experimental settings (*e.g.*, preprocessing and parameter tuning) described in Sec. 4.3 to all the evaluated log parsers. To evaluate their effectiveness, we report metrics including GA, PA, FGA, and FTA for both Loghub-2.0 and Loghub-2k. Additionally, we record the parsing time for each parser, which is measured from the beginning of loading log data to the completion of parsing. Following existing work [25, 63], we set a timeout of 12 hours. If a parser cannot finish parsing a dataset within this timeframe, we terminate the process and mark it as "timed out". Any parser that surpasses this time limit might not be suitable for practical deployment in a production environment, which often handles massive amounts of log data on a daily basis [38, 55, 63]. Due to the space limitation, interested readers can refer to our repository [1] for more detailed evaluation results.

5.2.1 Effectiveness. Fig. 3 presents a box plot illustrating the effectiveness of all log parsers on Loghub-2k and Loghub-2.0. Each box encapsulates the distribution of experimental results across all datasets in terms of a specific metric. In addition, we also denote the number of datasets each log parser can finish processing within 12 hours in the parentheses next to the parser's name.

According to Fig. 3, we can make the following observations. (1) For most log parsers, whether applied to Loghub-2k or Loghub-2.0, the average FGA is typically lower than GA. This shows that FGA is a more strict metric because it fairly treats all types of templates without considering their frequencies. (2) When comparing GA and FGA across Loghub-2k and Loghub-2.0, we can find that the decrease of FGA in Loghub-2.0 is more obvious than that in Loghub-2k. For example, for AEL, the discrepancy between the average GA and FGA on Loghub-2k is about 0.1. However, on Loghub-2.0, this value escalates to roughly 0.3. This indicates that the template distribution in Loghub-2.0 is more imbalanced than Loghub-2k, which validates our finding in RQ1. (3) Similarly, FTA generally decreases when compared with PA, either in the Loghub-2k or Loghub-2.0, suggesting that PA is also dominated by major classes.

Finding 2. Message-level metrics, such as GA and PA, usually produce higher evaluation results compared to template-level metrics like FGA and FTA, due to their sensitivity to imbalanced log data. The differences between these metrics are more noticeable in the large-scale Loghub-2.0, which displays greater imbalances.

In addition, it is obvious that the performance of all log parsers across all metrics displays a significant difference between the Loghub-2k and Loghub-2.0. Specifically, (1) When comparing Loghub-2k with the more imbalanced Loghub-2.0, we generally observe an increase in PA (*e.g.*, in AEL and Drain) and a slight decrease in GA for most parsers (*e.g.*, LenMA and LFA). This is attributed to the fact that GA demands precise grouping of log messages belonging to the same templates, while PA is calculated based on the accuracy of parsing individual log messages. The task of accurate grouping becomes more challenging within the larger Loghub-2.0, leading to a noticeable increase in PA and a slight decrease in GA when using Loghub-2.0. (2) All log parsers display a significant drop in template-level metrics on Loghub-2.0 compared to their performance on Loghub-2k. For instance, Drain, which achieves the highest FGA metric on Loghub-2k, sees its average FGA score dropping from 0.75 to approximately 0.55. Similarly, LogPPT, though achieving the highest FTA on Loghub-2k, experiences a reduction in its average FTA from roughly 0.64 to 0.5. (3) Furthermore, it is noteworthy that the variances of the four metrics across different datasets for most log parsers (*e.g.*, AEL, Drain, LenMa and LogMine) have significantly increased, visually represented by the expanded range of the box plot. This implies that existing parsers struggle to achieve consistent effectiveness across different large-scale systems.

Finding 3. The evaluation results obtained on the Loghub-2k do not consistently hold when the log parsers are applied to the large-scale Loghub-2.0. On Loghub-2.0, existing parsers experience a performance drop and an increase in the variance of all metrics.

Additionally, semantic-based log parsers, such as UniParser and LogPPT, have consistently demonstrated notably higher PA and FTA scores compared to other log parsers on both Loghub-2k and Loghub-2.0 datasets. This suggests that semantic information can facilitate accurately identifying the template of individual log messages. However, their GA and FGA scores are generally lower than those of other log parsers. This can be attributed to their disregard for global information, such as statistical frequency. As a result, these parsers are more prone to categorizing logs from the same templates into different groups, leading to lower GA and FGA scores. Furthermore, although semantic-based log parsers have achieved commendable performance on Loghub-2k, the performance metrics also decrease dramatically when applied to Loghub-2.0. The primary reason is that Loghub-2k contain too few log messages and log templates, making it easy for these models to learn the features of the entire dataset. For instance, in LogPPT, the default number of prompts for tuning is 32, however, many datasets in Loghub-2k have even fewer than 32 templates, resulting in LogPPT achieving near-perfect accuracy on these datasets. In contrast, since the number of log messages and log templates in Loghub-2.0 significantly increases, it becomes challenging for these models to generalize to more unseen log messages based on limited training samples.

Finding 4. Semantic-based log parsers are more capable of parsing individual logs, evidenced by their higher PA and FTA. However, they exhibit lower grouping-related metrics, due to their neglect of global information. Moreover, the performance of these log parsers may decline on larger and more diverse datasets in Loghub-2.0, particularly when the number of annotated samples available for training is limited.

5.2.2 Efficiency. For the Loghub-2k, all log parsers can successfully parse all 14 datasets. However, when these log parsers are applied to the larger-scale datasets of Loghub-2.0, most of them (9 out of 15) are unable to complete the parsing process for all 14 datasets within 12 hours. Due to the space limit, we have uploaded the detailed time cost for each parser processing each dataset to our replication repository [1]. As depicted in Fig. 3, only six parsers (*i.e.*, Drain, IPLoM, LFA, LogCluster, LogSig, UniParser, and LogPPT) successfully complete parsing on all 14 datasets of Loghub-2.0. Certain parsers, such as LenMa and LogMine, despite demonstrating superior performance, are unable to process larger datasets efficiently. Considering the substantial demands for log parsing throughput in real-world systems, *e.g.*, millions of logs per hour [55], log parsers that are unable to complete the parsing process within a reasonable timeframe (*i.e.*, 12 hours) may have limited applicability in practical scenarios. Moreover, semantic-based log parsers like LogPPT require GPU computational resources. When computing with a CPU, their time consumption is considerably higher compared to

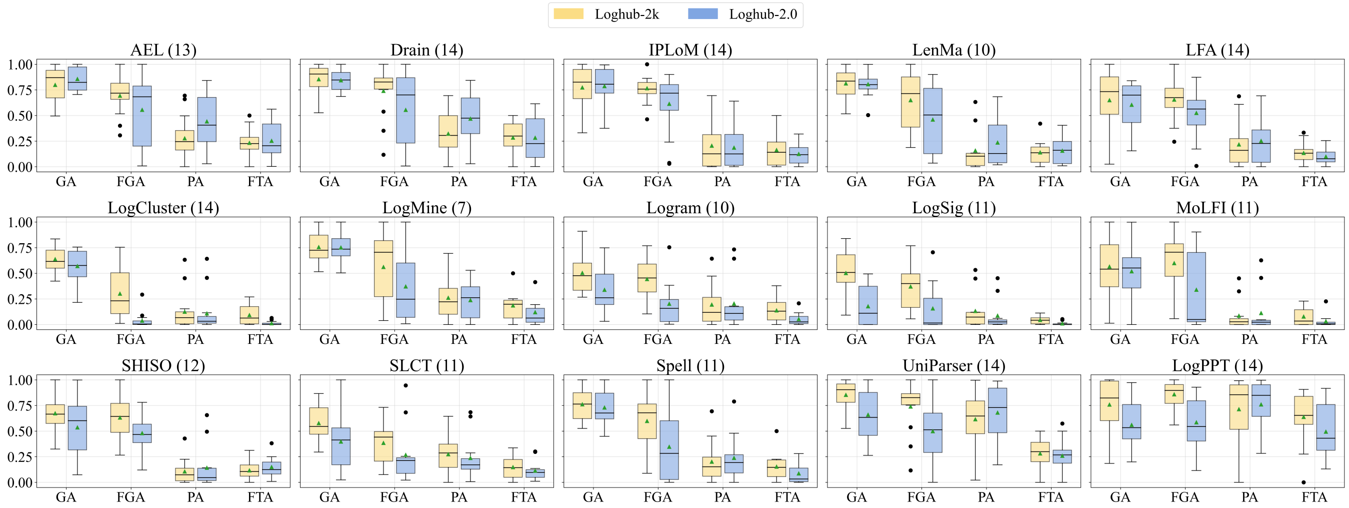


Figure 3: The evaluation results of all log parsers on Loghub-2k and Loghub-2.0

other efficient statistic-based log parsers like Drain. This potentially hampers their adoption in scenarios with resource constraints.

> **Finding 5.** 9 out of 15 log parsers are unable to process all 15 datasets of Loghub-2.0 within a reasonable 12-hour timeframe. Moreover, semantic-based methods typically demand more computational resources than statistic-based parsers.

5.3 RQ3: Performances of log parsers on logs with different characteristics

Previous studies [25, 63] typically report the overall performance of log parsers on the entire full set of a dataset. However, this may not fully characterize their effectiveness on logs with diverse characteristics, especially those that demand more attention in real-world system maintenance. To address this limitation, our benchmarking study adopts a more granular approach by evaluating log parsers on specific logs with varying characteristics. We primarily focus on the characteristics of template frequency and parameter count. In specific, log templates with a lower frequency often represent rare events, hidden problems, and potential failures. Besides, log templates with more parameters might be more informative to on-site engineers for analysis. Thus, it is crucial to accurately parse these two types of log messages.

To this end, we first apply log parsers to parse the entire dataset. We then select subsets of logs with different characteristics and calculate the performance on each subset. This approach ensures that the input data for each parser are consistent with that in RQ2, instead of merely parsing the selected subset of logs. Due to the space limitation, we only present the results of five representative log parsers that are capable of parsing the majority of datasets in Loghub-2.0. The complete results can be found in our repository [1].

5.3.1 Performance with different template frequencies. As mentioned in RQ1, Loghub-2.0 exhibits a higher imbalance in template frequencies. Considering the data-driven nature of log parsers, their performance could be affected. Hence, we investigate the performance on template frequencies by looking into the relative infrequent and frequent logs. In particular, we evaluate log parsers on the templates with top and bottom k% frequencies, where k is set as 5, 10, and 20, respectively. Then, we report the average scores of these four metrics. Fig. 4 illustrates the results when k=10, while the results for k=5 and 20 can be found in our repository [1].

As illustrated in Fig. 4, all log parsers exhibit worse GA and FGA on frequent logs than on infrequent ones. Taking LogPPT as an example, its GA and FGA scores approach 0.95 for infrequent templates, while dropping below 0.5 for frequent ones. We explain the performance drop as follows. Considering grouping accuracy requires a parser to correctly group all logs that belong to a certain template, the grouping will become more challenging as more log messages should be included.

In contrast, all log parsers demonstrate lower average PA for infrequently occurring log templates compared to frequent ones. This is expected for statistic-based log parsers, as the less frequency of a template provides less information and evidence (*e.g.*, count discrepancy between static and dynamic tokens) for the parser, resulting in decreased parsing accuracy. For semantic-based log parsers, such as LogPPT, which typically require a sample of logs for training, the sampling process reduces the likelihood of selecting infrequent templates. This, in turn, decreases the parsing accuracy of infrequent templates compared to high-frequency templates.

> **Finding 6.** Existing log parsers exhibit varying effectiveness when dealing with templates of different frequencies. They typically achieve lower GA and FGA for frequent templates, as the grouping is more challenging for templates with more log messages. Besides, they achieve lower PA and FTA for infrequent templates, as less evidence (*e.g.*, training data) is available to guide the accurate parsing of each log message.

5.3.2 Performance with different parameter counts. Our study in RQ1 demonstrates that parameter counts of templates can vary in a large range (*e.g.*, 0 to 25 for the Spark dataset). Therefore, we also evaluate parsing effectiveness for templates with different numbers of parameters. To achieve this, we classify the logs in each dataset within Loghub-2.0 into three categories based on their parameter count: logs with no parameters, logs with one to four parameter

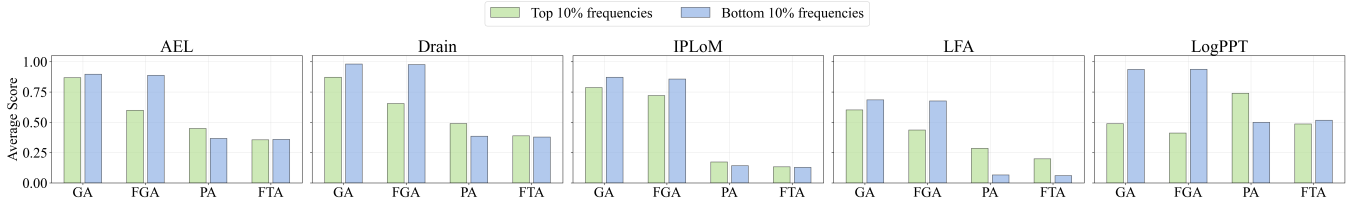


Figure 4: The evaluation results of log parsers on logs with different frequencies

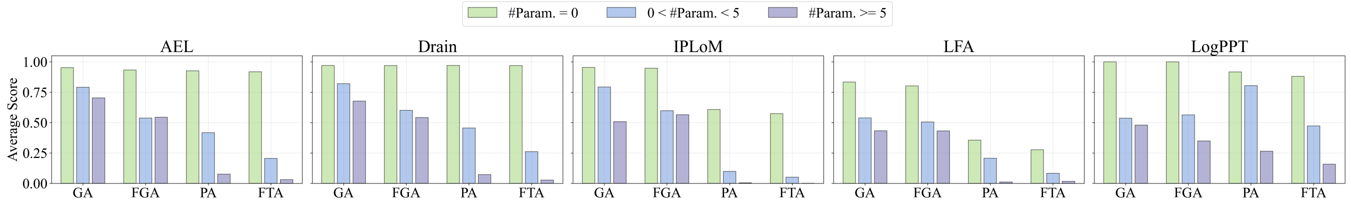


Figure 5: The evaluation results of log parsers on logs with different parameter counts

count, and logs with five or more parameters. We then utilize the same approach in Sec. 5.3.1 to calculate the average of four metrics for each log parser on each category, respectively.

According to the results illustrated in Fig. 5, we can observe that all log parsers exhibit a significant decline across all four performance metrics as the parameter counts increase. More specifically, these parsers perform exceptionally well on logs without parameters, significantly surpassing their overall performance on all logs as presented in Fig. 3. For example, Drain achieves an average score exceeding 0.95 on all four metrics on logs without parameters, much higher than overall performance on the complete set of logs. When dealing with logs with more than five parameters, all log parsers exhibit notably poor performance. For example, LogPPT only attains an average FGA of 0.16, while the average FGA of other methods does not surpass 0.03. This suggests that despite many log parsers demonstrating relatively high performance on entire datasets, their performance on logs with more parameters remains less than satisfactory, potentially resulting in distracting parsing errors in real-world applications.

Finding 7. Despite the high scores achieved by the log parsers on the entire datasets, their parsing effectiveness remains unsatisfactory when dealing with parameter-intensive log templates.

5.4 Summary of all research questions

We can make the following summaries of all research questions: (1) The proposed collection of large-scale datasets for log parsing, Loghub-2.0, exhibits significantly different characteristics of log data compared to the commonly used Loghub-2k. Loghub-2.0 presents greater challenges for existing log parsers due to its larger scale and more complex characteristics. (2) Our evaluation results indicate that Drain is the most performant parsers that are more capable of grouping log messages, as evidenced by the highest average GA and FGA. On the other hand, semantic-based methods (*e.g.*, UniParser and LogPPT) exhibit stronger abilities in distinguishing each token as either constant or dynamic parts. However, these methods compromise their effectiveness in grouping log messages with the same templates. This is because classification errors in tokens can easily lead to incomplete groups. (3) Despite the encouraging results shown in the Loghub-2k, the parsing performance remains unsatisfactory when applied to Loghub-2.0. This is particularly noticeable when parsing infrequent logs and parameter-intensive logs. (4) Moreover, the efficiency of the majority of log parsers fails to meet the demands of large-scale application scenarios.

6 DISCUSSION

6.1 Implications

Based on our findings, we have identified the following implications, which we believe could benefit future research on log parsing.

Consider both levels of metrics in combination. While most existing tasks utilize message-level metrics such as GA and PA to assess performance, these measures are often dominated by log templates with high frequencies in large-scale application scenarios, thereby yielding higher scores. In contrast, template-level metrics are resistant to the imbalanced frequencies of templates and thus can accurately reflect the parsing performance on datasets with diverse template distributions. Hence, these two types of metrics may be contemplated in conjunction, and one can be prioritized over the other based on specific requirements. For instance, if the focus is more on the parsing accuracy of frequent log templates and one can tolerate errors in infrequent templates, then message-level metrics are more appropriate, and vice versa.

Evaluate the performance across logs with different characteristics. Although certain log parsers have exhibited high overall performance on specific datasets, their parsing performance is still lacking when handling infrequent and parameter-intensive log templates. Considering the importance of these logs, as underscored in Section 2.2, it is crucial to concentrate specifically on performance within these logs. The evaluation protocol we propose can unearth the performance of log parsers on these log templates more comprehensively. Consequently, future work should pay attention to this aspect when designing new log parsers, thereby enhancing their applicability in real-world scenarios.

Place greater emphasis on efficiency. As discussed in Sec. 5.2.2, many existing log parsers fail to meet the performance requirements of large-scale application scenarios, a fact not represented in Loghub-2k. Considering the large volume of logs in practical settings, it is imperative that future log parsers are designed to meet the performance demands of specific applications.

Try to combine semantic and statistical information. According to finding 4, semantic-based log parsers possess superior capabilities in distinguishing parameters from templates, which also substantiates the significance of semantic information in the process of log parsing. However, their grouping abilities are compromised due to the neglect of global information. This is inevitable, given that these log parsers exclusively process each log message in isolation. A potential avenue for future research could involve the combination of semantic and statistical information in logs, thereby simultaneously enhancing the parsing and grouping capabilities.

6.2 Threats to Validity

Annotation errors The primary threat of this study is the potential annotation errors in Loghub-2.0, which is inevitable without the source code. To mitigate this issue as much as possible, we have designed a stringent annotation framework with a team comprising five members with significant experience in log analysis research.

Limited log parsers The selection of log parsers is limited, as not all existing log parsers are open-sourced due to industry confidentiality reasons [55]. Nevertheless, the selected parsers have included state-of-the-art log parsers published at top-tier conferences and covered all existing categories of technology. Furthermore, we have made our dataset available and implemented our benchmark protocol in a unified and user-friendly manner. This allows for the easy comparison of additional log parsers with existing ones.

Implementation and settings To mitigate the bias of implementation and settings, we have adopted the source code of several log parsers from the widely-used benchmark [28, 63]. For the newly incorporated log parsers, we have either used the source code provided by the original authors or carefully replicated them to ensure the fidelity of the results. Additionally, we have tuned the parameters of each log parser to optimize the results.

7 CONCLUSION

In this paper, we conduct a more rigorous and practical large-scale evaluation for log parsing techniques. We propose a log template annotation framework that ensures both efficiency and accuracy, and have annotated a new collection of large-scale datasets for log parsing, which more accurately reflects the scale and distribution of log data in real-world situations. Our proposed benchmarking protocol, inclusive of a new template-level metric and an evaluation of the performance of log parsers on logs with varying characteristics, offers a more comprehensive and in-depth analysis of log parsers' performance. Furthermore, our re-evaluation of selected log parsers using Loghub-2.0 uncovers valuable findings of the limitations of existing log parsers and benchmarks. We believe that our work, together with the open-source dataset Loghub-2.0 and benchmark, could benefit future research in the field of log analysis.

ACKNOWLEDGMENT

The work described in this paper was supported by the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CUHK 14206921 of the General Research Fund). We extend our sincere gratitude to the anonymous reviewers for their constructive feedback.

REFERENCES

[1] 2023. The replication repository of our evaluation artifacts. https://github.com/logpai/Loghub-2.0 [Online; accessed 1 Dec 2023].

[2] 2023. Scipy. https://scipy.org/ [Online; accessed 1 July 2023].

[3] Anunay Amar and Peter C Rigby. 2019. Mining historical test logs to predict bugs and localize faults in the test logs. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 140–151.

[4] James H Andrews. 1998. Testing using log file analysis: tools, methods, and issues. In *Proceedings 13th IEEE International Conference on Automated Software Engineering (Cat. No. 98EX239)*. IEEE, 157–166.

[5] Vincent Bushong, Russell Sanders, Jacob Curtis, Mark Du, Tomas Cerny, Karel Frajtak, Miroslav Bures, Pavel Tisnovsky, and Dongwan Shin. 2020. On matching log analysis to source code: A systematic mapping study. In *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*. 181–187.

[6] An Ran Chen, Tse-Hsun Chen, and Shaowei Wang. 2021. Pathidea: Improving information retrieval-based bug localization by re-constructing execution paths using logs. *IEEE Transactions on Software Engineering (TSE)* 48, 8 (2021), 2905–2919.

[7] Boyuan Chen, Jian Song, Peng Xu, Xing Hu, and Zhen Ming Jiang. 2018. An automated approach to estimating code coverage measures via execution logs. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 305–316.

[8] Zhichao Chen, Junjie Chen, Weijing Wang, Jianyi Zhou, Meng Wang, Xiang Chen, Shan Zhou, and Jianmin Wang. 2023. Exploring better black-Box test case prioritization via log analysis. *ACM Transactions on Software Engineering and Methodology* 32, 3 (2023), 1–32.

[9] Hetong Dai, Heng Li, Che-Shao Chen, Weiyi Shang, and Tse-Hsun Chen. 2020. Logram: Efficient Log Parsing Using n n-Gram Dictionaries. *IEEE Transactions on Software Engineering (TSE)* 48, 3 (2020), 879–892.

[10] Hetong Dai, Yiming Tang, Heng Li, and Weiyi Shang. 2023. PILAR: Studying and Mitigating the Influence of Configurations on Log Parsing. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 818–829.

[11] Min Du and Feifei Li. 2016. Spell: Streaming parsing of system event logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 859–864.

[12] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 1285–1298.

[13] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. 2009. Execution anomaly detection in distributed systems through unstructured log analysis. In *2009 ninth IEEE international conference on data mining (ICDM)*. IEEE, 149–158.

[14] Qiang Fu, Jieming Zhu, Wenlu Hu, Jian-Guang Lou, Rui Ding, Qingwei Lin, Dongmei Zhang, and Tao Xie. 2014. Where do developers log? an empirical study on logging practices in industry. In *Companion Proceedings of the 36th International Conference on Software Engineering*. 24–33.

[15] Ying Fu, Meng Yan, Jian Xu, Jianguo Li, Zhongxin Liu, Xiaohong Zhang, and Dan Yang. 2022. 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 (FSE)*. 1566–1577.

[16] Hossein Hamooni, Biplob Debnath, Jianwu Xu, Hui Zhang, Guofei Jiang, and Abdullah Mueen. 2016. Logmine: Fast pattern recognition for log analytics. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management (CIKM)*. 1573–1582.

[17] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. 2017. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE international conference on web services (ICWS)*. IEEE, 33–40.

[18] Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R Lyu. 2021. A survey on automated log analysis for reliability engineering. *ACM computing surveys (CSUR)* 54, 6 (2021), 1–37.

[19] Shilin He, Xu Zhang, Pinjia He, Yong Xu, Liqun Li, Yu Kang, Minghua Ma, Yining Wei, Yingnong Dang, Saravanakumar Rajmohan, et al. 2022. An empirical study of log analysis at Microsoft. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE)*. 1465–1476.

[20] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. 2020. Loghub: A large collection of system log datasets towards automated log analytics. *arXiv preprint arXiv:2008.06448* (2020).

[21] Yintong Huo, Yuxin Su, Cheryl Lee, and Michael R Lyu. 2021. Semparser: A semantic parser for log analysis. *arXiv preprint arXiv:2112.12636* (2021).

[22] Tong Jia, Lin Yang, Pengfei Chen, Ying Li, Fanjing Meng, and Jingmin Xu. 2017. Logsed: Anomaly diagnosis through mining time-weighted control flow graph in logs. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*. IEEE, 447–455.

[23] Zhihan Jiang, Jinyang Liu, Zhuangbin Chen, Yichen Li, Junjie Huang, Yintong Huo, Pinjia He, Jiazhen Gu, and Michael R Lyu. 2023. Llmparser: A llm-based log parsing framework. *arXiv preprint arXiv:2310.01796* (2023).

[24] Zhen Ming Jiang, Ahmed E Hassan, Parminder Flora, and Gilbert Hamann. 2008. Abstracting execution logs to execution events for enterprise applications (short paper). In *2008 The Eighth International Conference on Quality Software*. IEEE, 181–186.

[25] Zanis Ali Khan, Donghwan Shin, Domenico Bianculli, and Lionel Briand. 2022. Guidelines for assessing the accuracy of log message template identification techniques. In *Proceedings of the 44th International Conference on Software Engineering (ICSE)*. 1095–1106.

[26] Zanis Ali Khan, Donghwan Shin, Domenico Bianculli, and Lionel Briand. 2023. Impact of Log Parsing on Log-based Anomaly Detection. *arXiv preprint arXiv:2305.15897* (2023).

[27] Van-Hoang Le and Hongyu Zhang. 2022. Log-based anomaly detection with deep learning: How far are we?. In *Proceedings of the 44th international conference on software engineering (ICSE)*. 1356–1367.

[28] Van-Hoang Le and Hongyu Zhang. 2023. Log Parsing with Prompt-based Few-shot Learning. *arXiv preprint arXiv:2302.07435* (2023).

[29] Yichen Li, Yintong Huo, Zhihan Jiang, Renyi Zhong, Pinjia He, Yuxin Su, and Michael R Lyu. 2023. Exploring the Effectiveness of LLMs in Automated Logging Generation: An Empirical Study. *arXiv preprint arXiv:2307.05950* (2023).

[30] Yichen Li, Yintong Huo, Renyi Zhong, Zhihan Jiang, Jinyang Liu, Junjie Huang, Jiazhen Gu, Pinjia He, and Michael R Lyu. 2024. Go Static: Contextualized Logging Statement Generation. *arXiv preprint arXiv:2402.12958* (2024).

[31] Zhenhao Li, Chuan Luo, Tse-Hsun Chen, Weiyi Shang, Shilin He, Qingwei Lin, and Dongmei Zhang. 2023. Did We Miss Something Important? Studying and Exploring Variable-Aware Log Abstraction. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*.

[32] Jinyang Liu, Junjie Huang, Yintong Huo, Zhihan Jiang, Jiazhen Gu, Zhuangbin Chen, Cong Feng, Minzhi Yan, and Michael R Lyu. 2023. Scalable and Adaptive Log-based Anomaly Detection with Expert in the Loop. *arXiv preprint arXiv:2306.05032* (2023).

[33] Jiahao Liu, Jun Zeng, Xiang Wang, Kaihang Ji, and Zhenkai Liang. 2022. Tell: log level suggestions via modeling multi-level code block information. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. 27–38.

[34] Jinyang Liu, Jieming Zhu, Shilin He, Pinjia He, Zibin Zheng, and Michael R Lyu. 2019. Logzip: Extracting hidden structures via iterative clustering for log compression. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 863–873.

[35] Yudong Liu, Xu Zhang, Shilin He, Hongyu Zhang, Liqun Li, Yu Kang, Yong Xu, Minghua Ma, Qingwei Lin, Yingnong Dang, et al. 2022. Uniparser: A unified log parser for heterogeneous log data. In *Proceedings of the ACM Web Conference 2022 (WWW)*. 1893–1901.

[36] Steven Locke, Heng Li, Tse-Hsun Peter Chen, Weiyi Shang, and Wei Liu. 2021. LogAssist: Assisting log analysis through log summarization. *IEEE Transactions on Software Engineering (TSE)* 48, 9 (2021), 3227–3241.

[37] Junchen Ma, Yang Liu, Hongjie Wan, and Guozi Sun. 2023. Automatic Parsing and Utilization of System Log Features in Log Analysis: A Survey. *Applied Sciences* 13, 8 (2023), 4930.

[38] Shiqing Ma, Juan Zhai, Yonghwi Kwon, Kyu Hyung Lee, Xiangyu Zhang, Gabriela Ciocarlie, Ashish Gehani, Vinod Yegneswaran, Dongyan Xu, and Somesh Jha. 2018. {Kernel-Supported} {Cost-Effective} Audit Logging for Causality Tracking. In *2018 USENIX Annual Technical Conference (USENIX ATC)*. 241–254.

[39] Adetokunbo AO Makanju, A Nur Zincir-Heywood, and Evangelos E Milios. 2009. Clustering event logs using iterative partitioning. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*. 1255–1264.

[40] Salma Messaoudi, Annibale Panichella, Domenico Bianculli, Lionel Briand, and Raimondas Sasnauskas. 2018. A search-based approach for accurate identification of log message formats. In *Proceedings of the 26th Conference on Program Comprehension*. 167–177.

[41] Salma Messaoudi, Donghwan Shin, Annibale Panichella, Domenico Bianculli, and Lionel C Briand. 2021. Log-based slicing for system-level test cases. In *Proceedings of the 30th ACM SIGSOFT international symposium on software testing and analysis (ISSTA)*. 517–528.

[42] Masayoshi Mizutani. 2013. Incremental mining of system log format. In *2013 IEEE International Conference on Services Computing*. IEEE, 595–602.

[43] Meiyappan Nagappan and Mladen A Vouk. 2010. Abstracting log lines to log event types for mining software system logs. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR)*. IEEE, 114–117.

[44] Meiyappan Nagappan, Kesheng Wu, and Mladen A Vouk. 2009. Efficiently extracting operational profiles from execution logs using suffix arrays. In *2009 20th International Symposium on Software Reliability Engineering*. IEEE, 41–50.

[45] Karthik Nagaraj, Charles Killian, and Jennifer Neville. 2012. Structured comparative analysis of systems logs to diagnose performance problems. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. 353–366.

[46] Paolo Notaro, Soroush Haeri, Jorge Cardoso, and Michael Gerndt. 2023. LogRule: Efficient Structured Log Mining for Root Cause Analysis. *IEEE Transactions on Network and Service Management* (2023).

[47] Antonio Pecchia, Marcello Cinque, Gabriella Carrozza, and Domenico Cotroneo. 2015. Industry practices and event logging: Assessment of a critical software development process. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE)*, Vol. 2. IEEE, 169–178.

[48] Daan Schipper, Maurício Aniche, and Arie van Deursen. 2019. Tracing back log data to its log statement: from research to practice. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 545–549.

[49] Issam Sedki, Abdelwahab Hamou-Lhadj, Otmane Ait-Mohamed, and Naser Ezzati-Jivan. 2023. Towards a Classification of Log Parsing Errors. In *2023 IEEE/ACM 31st International Conference on Program Comprehension (ICPC)*. IEEE, 84–88.

[50] Weiyi Shang. 2012. Bridging the divide between software developers and operators using logs. In *2012 34th international conference on software engineering (ICSE)*. IEEE, 1583–1586.

[51] Keiichi Shima. 2016. Length matters: Clustering system log messages using length of words. *arXiv preprint arXiv:1611.03213* (2016).

[52] Liang Tang, Tao Li, and Chang-Shing Perng. 2011. LogSig: Generating system events from raw textual logs. In *Proceedings of the 20th ACM international conference on Information and knowledge management (CIKM)*. 785–794.

[53] Risto Vaarandi. 2003. A data clustering algorithm for mining patterns from event logs. In *Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM)(IEEE Cat. No. 03EX764)*. Ieee, 119–126.

[54] Risto Vaarandi and Mauno Pihelgas. 2015. Logcluster-a data clustering and pattern mining algorithm for event logs. In *2015 11th International conference on network and service management (CNSM)*. IEEE, 1–7.

[55] Xuheng Wang, Xu Zhang, Liqun Li, Shilin He, Hongyu Zhang, Yudong Liu, Lingling Zheng, Yu Kang, Qingwei Lin, Yingnong Dang, et al. 2022. SPINE: a scalable log parser with feedback guidance. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE)*. 1198–1208.

[56] Kundi Yao, Mohammed Sayagh, Weiyi Shang, and Ahmed E Hassan. 2021. Improving state-of-the-art compression techniques for log management tools. *IEEE Transactions on Software Engineering (TSE)* 48, 8 (2021), 2748–2760.

[57] Siyu Yu, Ningjiang Chen, Yifan Wu, and Wensheng Dou. 2023. Self-supervised log parsing using semantic contribution difference. *Journal of Systems and Software* 200 (2023), 111646.

[58] Siyu Yu, Pinjia He, Ningjiang Chen, and Yifan Wu. 2023. Brain: Log Parsing with Bidirectional Parallel Tree. *IEEE Transactions on Services Computing (TSC)* (2023).

[59] Ding Yuan, Haohui Mai, Weiwei Xiong, Lin Tan, Yuanyuan Zhou, and Shankar Pasupathy. 2010. Sherlog: error diagnosis by connecting clues from run-time logs. In *Proceedings of the fifteenth International Conference on Architectural support for programming languages and operating systems*. 143–154.

[60] Tianzhu Zhang, Han Qiu, Gabriele Castellano, Myriana Rifai, Chung Shue Chen, and Fabio Pianese. 2023. System Log Parsing: A Survey. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* (2023).

[61] Chen Zhi, Jianwei Yin, Shuiguang Deng, Maoxin Ye, Min Fu, and Tao Xie. 2019. An exploratory study of logging configuration practice in java. In *2019 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 459–469.

[62] Jieming Zhu, Shilin He, Pinjia He, Jinyang Liu, and Michael R Lyu. 2023. Loghub: A large collection of system log datasets for ai-driven log analytics. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 355–366.

[63] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R Lyu. 2019. Tools and benchmarks for automated log parsing. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 121–130.

Received 14-DEC-2023; accepted 2024-03-02