Loghub: A Large Collection of System Log Datasets for AI-driven Log Analytics

Jieming Zhu*, Shilin He*, Pinjia He†✉, Jinyang Liu‡, Michael R. Lyu‡

†School of Data Science, The Chinese University of Hong Kong, Shenzhen (CUHK Shenzhen), China
‡Department of Computer Science and Engineering, The Chinese University of Hong Kong, China
jiemingzhu@ieee.org slhe@link.cuhk.edu.hk hepinjia@cuhk.edu.cn {jyliu, lyu}@cse.cuhk.edu.hk

***Abstract*—Logs have been widely adopted in software system development and maintenance because of the rich runtime information they record. In recent years, the increase of software size and complexity leads to the rapid growth of the volume of logs. To handle these large volumes of logs efficiently and effectively, a line of research focuses on developing intelligent and automated log analysis techniques. However, only a few of these techniques have reached successful deployments in industry due to the lack of public log datasets and open benchmarking upon them. To fill this significant gap and facilitate more research on AI-driven log analytics, we have collected and released loghub, a large collection of system log datasets. In particular, loghub provides 19 real-world log datasets collected from a wide range of software systems, including distributed systems, supercomputers, operating systems, mobile systems, server applications, and standalone software. In this paper, we summarize the statistics of these datasets, introduce some practical usage scenarios of the loghub datasets, and present our benchmarking results on loghub to benefit the researchers and practitioners in this field. Up to the time of this paper writing, the loghub datasets have been downloaded for roughly 90,000 times in total by hundreds of organizations from both industry and academia. The loghub datasets are available at https://github.com/logpai/loghub.**

***Index Terms*—Log datasets, log analytics, log intelligence, benchmarks, anomaly detection**

I. Introduction

Logs have been widely adopted in software system development and maintenance [28], [30]. In industry, it is a common practice to record detailed software runtime information into logs, allowing developers and operating engineers to track system behaviors and perform post-mortem analysis.

In general, logs are a form of unstructured texts printed by logging statements (e.g., *logging.info()*, *printf()*, *Console.Writeline()*) in source code. A log message, as illustrated in the following example, records a specific system event with a set of fields: **timestamp** (the occurrence time of the event, *e.g., 2008-11-09 20:46:55,556*), **verbosity level** (the severity level of the event, *e.g., INFO*), and **message content** that describes the event in free text.

```
2008-11-09 20:46:55 INFO dfs.DataNode$PacketRespond:
Received block blk_3587508140051953248 of size 6710.
```

The rich information recorded by logs enables developers to conduct a variety of log-based analysis and management tasks, such as anomaly detection [72], [18], [27], [15], duplicate issue identification [13], [39], [58], usage statistics analysis [34], and program verification [2], [61]. For example, developers could inspect log messages and analyze whether the system behaves as expected. However, software systems are becoming large in scale and complex in structure. The volume of system logs is growing rapidly as well (e.g., 50 GB/hour [51]), making manual log analysis become labor-intensive and time-consuming. To address this problem, a line of research [72], [18], [27], [13], [39], [58], [34], [2], [61] has targeted at making automated log analysis possible based on artificial intelligence (AI) techniques. These studies demonstrate that the use of AI techniques can greatly facilitate log analysis tasks by extracting critical information of runtime behaviors.

Figure 1 illustrates an overall framework for AI-driven log analytics. In the development phase, developers can make logging decisions guided by *strategic logging practices* (i.e., "where to log" [17], [80] and "what to log" [36], [24]) mined from high-quality software repositories. At system runtime, logs are *collected and aggregated* in a streaming manner. To reduce the storage cost of system logs, *log compression* techniques [45] could be further applied. In the operation and maintenance phase, logs need to be parsed into structured events with *log parsing* techniques [25], [81], and then facilitate the modeling and mining for a variety of *log analysis* tasks (e.g., anomaly detection [27], problem identification [29]).

Along with this framework, many efforts have been devoted to improving AI techniques towards logging, log collection, log compression, log parsing, and log analysis. Many more methods are being proposed as well. However, there is still a large gap between research and practice. First, researchers in this field often work on their own log data. Logs are scarce data in public for research, because companies are often reluctant to release their production logs due to privacy concerns. Thus, an approach that works well on one type of log data may become ineffective on another type of logs. Second, it is difficult and time-consuming for researchers and practitioners to implement the approaches and accurately reproduce the results without a standard benchmark.

To bridge this gap, this paper presents loghub, a large collection of system log datasets for AI-driven log analytics. Loghub contains a total of 19 log datasets (see Table I for details) generated by a wide range of systems, including distributed systems, super computers, operating systems, mo-

* The work was done when the authors were affiliated with CUHK.
✉ Pinjia He is the corresponding author.

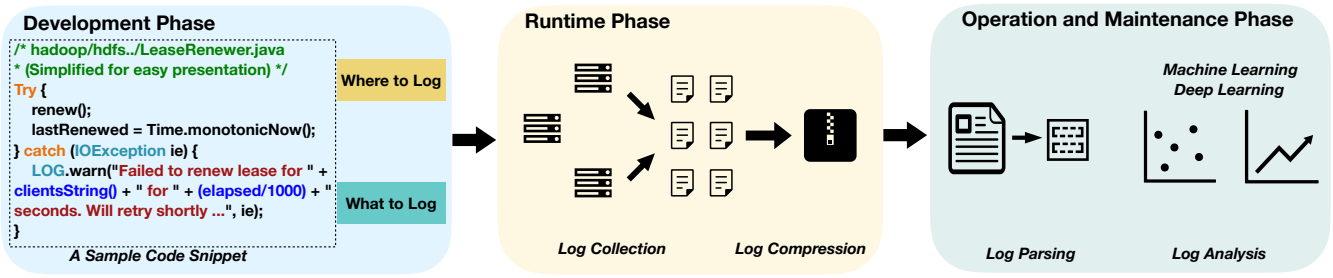

Fig. 1. Framework of AI-driven Log Analytics

bile systems, server applications, and standalone software. All these logs amount to about 77 GB in total. In particular, some of the logs are production data released from previous studies, while some others are collected from real systems in our lab environment. Among these log datasets, six of them are labeled (e.g., normal or abnormal, alerts or not alerts), which are amendable to studies for anomaly detection and duplicate issues identification. Additionally, other datasets could facilitate research on log parsing, log compression, and unsupervised methods for anomaly detection. Since the first release of these logs, they have been downloaded 90,000+ times by more than 450 organizations from both industry (35%) and academia (65%). We envision that loghub could serve as an open benchmarks towards research and practice for AI-driven log analytics. In summary, our work makes the following contributions:

- We collect and organize a large collection of log datasets (namely loghub) generated by a wide variety of systems. Loghub consists of 19 datasets, which are valuable for research and practice of AI-driven log analytics (§ II).
- We introduce practical usage scenarios of the loghub datasets (§ III). We also provide benchmarking results on three typical log analysis tasks using loghub and discuss remaining questions and challenges, shedding light on potential directions for future research and development in log analytics (§ IV).
- Our loghub datasets have been made available on Github. Since the release of loghub, they have made a measurable impact to the community, benefiting research in over 450 organizations from both industry and academia.

II. Loghub Datasets

Loghub maintains a collection of system logs, which are freely accessible for research. Some of the logs are production data released from previous studies, while some others are collected from real systems in our lab environment. Wherever possible, the logs are not sanitized, anonymized or modified in any way. All these logs amount to over 77 GB in total.

Table I presents an overview of the loghub datasets with some details about the description, time span, #Lines, data size, and the label information. Specifically, time span indicates the time range that the logs are collected. #Lines denotes the total number of log lines in a dataset. Data size shows the uncompressed log volume size. The "labeled" column indicates whether a dataset is labeled with anomaly information.

There are two categories of log datasets: *labeled* and *unlabeled*. Logs in labeled datasets contain labels for specific log analysis tasks (e.g., anomaly detection and duplicate issues identification). For example, in the labeled HDFS datasets, the labels indicate whether the system operations on an HDFS block is abnormal. Thus, developers could utilize the labeled HDFS dataset to evaulate their anomaly detection approaches. In loghub, 6 log datasets are labeled, while 13 log datasets are unlabeled. Note that unlabeled log datasets are also useful for the evaluation of log analytics tasks, such as log parsing, log compression, and unsupervised methods (e.g., word2vec). The details of each log dataset in loghub are introduced as follows.

A. Distributed Systems

HDFS. HDFS is the Hadoop Distributed File System designed to run on commodity hardware. Due to the popularity of HDFS, it has been widely studied in recent years. We provide three sets of HDFS logs in loghub: HDFS-v1, HDFS-v2, and HDFS-v3. HDFS-v1 is generated in a 203-nodes HDFS using benchmark workloads, and manually labeled through handcrafted rules to identify the anomalies. The logs collected from the work [73] are sliced into traces (i.e., log sequences) according to block IDs. Then each trace associated with a specific block ID is assigned a ground truth label: *normal* or *abnormal*. Additionally, HDFS-v1 also provide the specific anomaly type information, while futher allows research on duplicate issues identification. HDFS-v2 is collected by aggregating logs from the HDFS cluster in our lab environment, which comprises one name node and 32 data nodes. The logs are aggregated at the node level. The logs have a huge size (over 16 GB) and are provided as-is without further modification or labeling. HDFS-v3 is an open dataset from trace-oriented monitoring [79], which is collected through instrumenting the HDFS system using MTracer [78] in a real IaaS environment. The logs are collected under different workloads (e.g., multiple scales of clusters, different kinds of user requests, various workload levels). In addition to some normal trace logs, abnormal logs are also collected via fault injection.

TABLE I
SUMMARY OF LOGHUB DATASETS

Dataset	Description	Time Span	#Lines	Data Size	Labeled
Distributed systems					
HDFS_v1	Hadoop distributed file system log	38.7 hours	11,175,629	1.47GB	√
HDFS_v2	Hadoop distributed file system log	N.A.	71,118,073	16.06GB	
HDFS_v3	Instrumented HDFS trace log	N.A.	14,778,079	2.96GB	√
Hadoop	Hadoop mapreduce job log	N.A.	394,308	48.61MB	√
Spark	Spark job log	N.A.	33,236,604	2.75GB	
Zookeeper	ZooKeeper service log	26.7 days	74,380	9.95MB	
OpenStack	OpenStack infrastructure log	N.A.	207,820	58.61MB	√
Super computers					
BGL	Blue Gene/L supercomputer log	214.7 days	4,747,963	708.76MB	√
HPC	High performance cluster log	N.A.	433,489	32.00MB	
Thunderbird	Thunderbird supercomputer log	244 days	211,212,192	29.60GB	√
Operating systems					
Windows	Windows event log	226.7 days	114,608,388	26.09GB	
Linux	Linux system log	263.9 days	25,567	2.25MB	
Mac	Mac OS log	7.0 days	117,283	16.09MB	
Mobile systems					
Android_v1	Android framework log	N.A.	1,555,005	183.37MB	
Android_v2	Android framework log	N.A.	30,348,042	3.38GB	
HealthApp	Health app log	10.5 days	253,395	22.44MB	
Server applications					
Apache	Apache web server error log	263.9 days	56,481	4.90MB	
OpenSSH	OpenSSH server log	28.4 days	655,146	70.02MB	
Standalone software					
Proxifier	Proxifier software log	N.A.	21,329	2.42MB	

Hadoop. Hadoop is a big data processing framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. Due to the increasing importance of Hadoop in industry, it has been widely studied in the literature. The logs are generated from a Hadoop cluster with 46 cores across five machines in [41]. Each machine has Intel(R) Core(TM) i7-3770 CPU and 16 GB RAM. Two testing applications are executed: *WordCount* and *PageRank*. Firstly, the applications are run without injecting any failure. Then, in order to simulate service failures in the production environment, the following deployment failures are injected: (1) *machine down*: during application runtime, turn off one server to simulate the machine failure; (2) *network disconnection*: disconnect one server from the network to simulate the network connection failure; and (3) *disk full*: during application runtime, manually fill up one server's hard disk to simulate the disk full failure. The labels of different failures are provided, making the data amenable to duplicate issues identification research.

Spark. Apache Spark is a unified analytics engine for big data processing, with built-in modules for streaming, SQL, machine learning and graph processing. Currently, Spark has been widely deployed in industry. This dataset was collected by aggregating logs from the running Spark system in our lab environment, which comprises a total of 32 machines. The logs are aggregated at the machine level. The logs have a huge size (over 2 GB) and are provided as-is without further modification or labelling, which involve both normal and abnormal application runs.

Zookeeper. ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. The log dataset was collected by aggregating logs from the ZooKeeper service in our lab environment, which comprises

a total of 32 machines, covering a time period of 26.7 days.

OpenStack. OpenStack is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a datacenter. This dataset was provided by [15] and was generated on CloudLab [11], a flexible, scientific infrastructure for research on cloud computing. Both normal logs and abnormal cases with failure injection are provided, making the data amenable to anomaly detection research.

B. Supercomputers

BGL. BGL is an open dataset of logs collected by [55] from a BlueGene/L supercomputer system at Lawrence Livermore National Labs (LLNL) in Livermore, California, with 131,072 processors and 32,768 GB memory [37]. The logs contain alert and non-alert messages identified by alert category tags. In the first column of the log, "-" indicates non-alert messages while others are alert messages. The label information is amenable to alert detection and prediction research.

HPC. HPC is an open dataset from the work [48], containing logs collected from System 20 of the high performance computing cluster at the Los Alamos National Laboratories, which has 49 nodes with 6,152 cores and 128 GB memory per node.

Thunderbird. Thunderbird is an open dataset of logs provided by [55], which was collected from a Thunderbird supercomputer system at Sandia National Labs (SNL) in Albuquerque, with 9,024 processors and 27,072 GB memory. The logs contain alert and non-alert messages identified by alert category tags. In the first column of the log, "-" indicates non-alert messages while others are alert messages. The label information is amenable to alert detection and prediction research.

C. Operating Systems

Windows. This log dataset was collected by aggregating a number of logs from a lab computer running Windows 7. The original logs were located at *C:/Windows/Logs/CBS*. CBS (Component Based Servicing) is a componentization architecture in Windows, which works at the package/update level. The CBS architecture is far more robust and secure than the installers in previous operating systems. Users benefit from a more complete and controlled installation process that allows updates, drivers and optional components to be added while simultaneously mitigating against instability issues caused by improper or partial installation. The logs have a huge size (over 27 GB) and span a period of 226.7 days.

Linux. Linux logs are usually located at */var/log/*. The dataset was collected from */var/log/messages* on a Linux server over a period of 263.9 days, as part of the Public Security Log Sharing Site project [10].

Mac. We collected the MacOS logs from */var/log/system.log* on a personal Macbook after 7 days of use. The log records the user activities on the Mac OS.

D. Mobile Applications

Android. Android is a popular open-source mobile operating system and has been used by many smart devices. However, Android logs are rarely available in public for research purposes. We provide some Android log files, which were collected on Android smartphones with heavily instrumented modules installed. The Android architecture comprises of five levels, including the Linux Kernel, Libraries, Application Framework, Android Runtime, and System Applications. We provide two dataset versions of Andriod logs printed by the Application Framework: Andriod-v1 and Andriod-v2. The Andriod-v1 dataset is a sampled small log file from Andriod-v2, while in Andriod-v2, the logs cover two types of issues, and each type has over 10 duplicate issue logs. However, due to the high complexity of Android's multi-threading system, it is difficult to pinpoint the abnormal log points.

HealthApp. HealthApp is a mobile application for Andriod devices. We collected the application logs from an Android smartphone after more than 10 days of use.

E. Server Applications

Apache. Apache HTTP Server is one of the most popular web servers. Apache servers usually generate two types of logs: access logs and error logs. This dataset provides an error log for the purpose of research on anomaly detection and diagnosis. The log file was collected from a Linux system running Apache Web server, as part of the Public Security Log Sharing Site project [10].

OpenSSH. OpenSSH is the premier connectivity tool for remote login with the SSH protocol. We collected the log from an OpenSSH server in our lab over a period of 28 days.

F. Standalone Software

Proxifier. Proxifier is a software program, allowing network applications that do not support working through proxy servers to operate through a SOCKS or HTTPS proxy and chains. We collected the Proxifier logs from a desktop computer in our lab.

III. Usage of Loghub Datasets

In this section, we present some common usage scenarios of the loghub datasets.

A. Overview

The loghub datasets have been made available for five years. During this time, we conducted a survey via Zenodo, a dataset hosting website, to gather information regarding the organization and usage scenarios of dataset downloading requests. Since its release, loghub has attracted the attention of not only large companies such as IBM, Microsoft, Huawei, Nvidia, MasterCard, Adobe, BMW, and Samsung, but also some startup companies focusing on building log analysis products, including Elastic.co, Splunk, Rapid7, Element AI, White Ops, Unomaly.com, and Ascend.io[1]. Many universities

[1]See https://github.com/logpai/loghub/wiki/Loghub-download-list.

have also requested the loghub dataset. To date, loghub has been downloaded 90,000+ times by more than 450 organizations from both industry (35%) and academia (65%). After analyzing the information collected from the data requests received, we manually categorize the log usage scenarios and present the distribution in Figure 2. It is worth noting that due to the limited information provided by users, we could only provide a rough categorization for them. We denote it as an unknown category if the user input is not clear. We can see that loghub datasets can potentially facilitate 23 different categories of research and education purposes. The top 5 usage scenarios are anomaly detection, log analysis, security, log parsing, and education. In particular, log analysis may cover the spectrum of some other log related tasks, but this has not been clearly specified by users. Education indicates the use cases about course projects and thesis projects. Here, we do not intend to expand all usage scenarios shown in the figure, but the variety of them have already confirmed the practical importance of the loghub datasets.

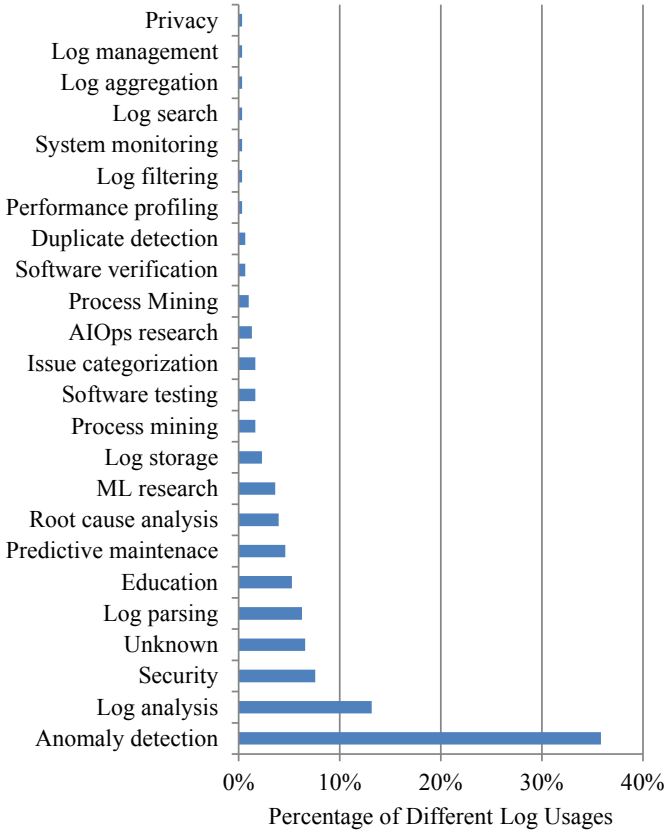

Fig. 2. Summary of Tentative Industry Adoption of Loghub

In the following, we present the details of four common usage scenarios that have been widely studied in the literature and describe how loghub can be used in these tasks, including log parsing, log compression, anomaly detection, and duplicate issue identification.

B. Log Parsing

Most of the AI-based log analysis approaches require structured input data, such as a list of system events with event IDs or a matrix. However, software logs are often unstructured texts, containing several fields and natural language descriptions written by developers. Thus, log parsing [81] is a crucial step in AI-driven log analytics that transforms unstructured log messages into structured system events.

However, as the volume of logs increases rapidly, traditional parsing approaches that largely rely on manual parsing rules construction becomes labor-intensive and inefficient. To address this problem, recent research has proposed various data-driven log parsers [65], [54], [66], [19], [63], [21], [52], [62], [33], [48], [23], [14], [67], [71], [32], which automatically label an unstructured log message with corresponding system event ID. Typically, log parsing is modeled as a clustering problem, where log messages describing the same system event should be clustered into the same group. The common tokens in all the log messages in the same group is regarded as the system event or event template. The research problem of log parsing is how to accurately and efficiently separate the unstructured log messages into different clusters by designing similarity metrics for log messages and novel clustering approaches. By clustering log messages into groups, log parsers can summarize the corresponding system events and match each log message with an event ID. The structured logs (i.e., log messages with event ID) could be easily transformed into a matrix or directly utilized by log analysis algorithms.

To evaluate the accuracy and efficiency of log parsing approaches, we need: (1) a large volume of logs and (2) logs generated by a variety of systems. Loghub contains 19 log datasets collected from 6 categories of systems. Besides, all the logs amount to over 77 GB. Thus, datasets in loghub can be employed in the experiments to evaluate the parsing accuracy and efficiency of different log parsing approaches [81].

C. Log Compression

Logs can be used in various system maintenance tasks, and thus they often need to be stored for a long time (e.g., a year or more) in practice. As the the explosion of log size in recent years, archiving system logs is consuming a large amount of storage space, which leads to high cost of electrical power. General compression approaches do not work well on log compression because they do not consider the inherent structure of log messages. To achieve higher compression ratio, a new line of research [22], [57], [12], [40], [16] has proposed compression approaches specialized for log data.

Log compression can be modeled as a frequent pattern mining problem. Existing approaches focus on finding inherent structure information of log messages (e.g., repetitive text). In particular, these approaches provide different strategies to detect repetitive text, such as utilizing the common format of logs generated by a specific system [12]. The research problem of log compression is how to achieve efficient and lossless compression with high compression rate.

To evaluate the accuracy and efficiency of log compression approaches, similar to the evaluation of log parsers, we need a large volume of logs collected from diverse systems. Thus, all the datasets in loghub can facilitate the evaluation of log compression approaches, as demonstrated in [45].

D. Anomaly Detection

Modern systems have become large-scale in size and complex in structure. An increasing number of these systems are expected to run on a 24×7 basis serving millions of users globally. Any non-trivial downtime of them could lead to enormous revenue loss [64], [53]. Thus, to enhance the reliability of modern systems, a line of recent research [72], [47], [3], [18], [27], [41], [15], [7], [44], [31] has focused on log-based anomaly detection approaches that report potential abnormal system behaviors by analyzing system runtime logs.

The anomaly detection problem is usually modeled as a binary classification problem. The input is a list of structured system events or a matrix, while the output is a list of labels indicating whether an instance (e.g., an event or a time period) is abnormal. There are mainly two categories of log-based anomaly detection approaches: *unsupervised* [72], [47], [41] and *supervised* [3], [18], [15]. The research problem of log-based anomaly detection is how to accurately detect the anomalies based on system logs. Towards this end, F-measure (i.e., F1 score) [49], a commonly-used evaluation metric for classification algorithms, is employed.

To evaluate the accuracy of log-based anomaly detection approaches, we need log datasets that contain anomaly labels for instances (e.g., whether a time period is regarded as anomaly). Loghub contains 6 labeled log datasets, which can be used in the experiments to evaluate the accuracy of diverse anomaly detection approaches [27].

E. Duplicate Issues Identification

To enhance system reliability, one important task for developers is to handle user-reported operational issues efficiently. An operational issue is a system problem reported by users. When a user of Amazon EC2 finds that her node becomes extremely slow, she will report the node slowness as an issue to Amazon. To handle an operational issue, developers need to mainly inspect the runtime logs to understand the system operations, which is time-consuming. Thus, to facilitate the issue handling process, recent research has proposed duplicate issues identification techniques [13], [39], [58], [43] to alleviate unnecessary manual effort.

The duplicate issue identification problem can be modeled as a clustering problem, while the duplicate issues are clustered into the same group based on the corresponding log messages. If the log messages of two issues demonstrate similar patterns (e.g., occurrence frequency, order), the two issue will be clustered in to the same group. The research problem of log-based duplicate issues identification is how to accurately separate the log messages (i.e., log sequences) of different issues into clusters.

TABLE II
SUMMARY OF LOG PARSING TOOLS.

Log Parser	Technique	Mode	Industrial Use
SLCT	Frequent pattern mining	Offline	N.A.
AEL	Heuristics	Offline	RIM
IPLoM	Iterative partitioning	Offline	N.A.
LKE	Clustering	Offline	Microsoft
LFA	Frequent pattern mining	Offline	N.A.
LogSig	Clustering	Offline	N.A.
SHISO	Clustering	Online	N.A.
LogCluster	Frequent pattern mining	Offline	CCDCOE
LenMa	Clustering	Online	N.A.
LogMine	Clustering	Offline	N.A.
Spell	Longest common subsequence	Online	N.A.
Drain	Parsing tree	Online	N.A.
MoLFI	Evolutionary algorithms	Offline	N.A.

To evaluate the accuracy of log-based duplicate issues identification approaches, log datasets that contain the issue categories are needed. Loghub contains 3 labeled log datasets (i.e., HDFS-v1, Hadoop, Android-v2) that provide this information. Thus, loghub could be used for this task.

IV. BENCHMARKING ON LOGHUB DATASETS

In this section, we demonstrate the use of loghub dataset via benchmarking typical log analysis tasks including log parsing, log compression, and log-based anomaly detection. From the results of this benchmarking, we derive critical unresolved questions and challenges inherent to each task. Our aspiration is that the academic community may leverage the insights drawn from the large-scale dataset, loghub, thereby fostering further advancements in the field.

A. Benchmarking for Log Parsing

In the following, we describe a case study of benchmarking existing log parsing algorithms using loghub.

1) Existing Log Parsing Algorithms: We have evaluated 13 log parsing algorithms as shown in Table II, which can be categorized into four types: *frequent pattern-based*, *clustering-based*, *heuristics-based* methods and *others*.

Typical *frequent pattern-based* methods include SLCT [65], LFA [54] and LogCluster [66]. They require the entire set of log messages ready before parsing, i.e., offline mode. They have a similar rationale that first identify frequent patterns (e.g., tokens or token-position pairs) then group all log messages based on these patterns to clusters. The templates are finally extracted from each cluster. *Clustering-based* methods include LKE [19], LogSig [63], SHISO [52], LenMa [62] and LogMine [21]. These methods rely on a core clustering algorithm (e.g., hierarchical clustering) to group log messages to clusters, then extract a template from each cluster. Among these clustering-based methods, SHISO and Lenma have an online mode that can process each log message without seeing

TABLE III
ACCURACY OF EXISTING LOG PARSING APPROACHES.

Dataset	SLCT	AEL	IPLoM	LKE	LFA	LogSig	SHISO	LogCluster	LenMa	LogMine	Spell	Drain	MoLFI	Best
HDFS	0.545	**0.998**	**1***	**1***	0.885	0.850	**0.998**	0.546	**0.998**	0.851	**1***	**0.998**	**0.998**	**1**
Hadoop	0.423	0.538	**0.954**	0.670	**0.900**	0.633	0.867	0.563	0.885	0.870	0.778	**0.948**	**0.957***	**0.957**
Spark	0.685	**0.905**	**0.920**	0.634	**0.994***	0.544	**0.906**	0.799	0.884	0.576	**0.905**	**0.920**	0.418	**0.994**
Zookeeper	0.726	**0.921**	**0.962**	0.438	0.839	0.738	0.660	0.732	0.841	0.688	**0.964**	**0.967***	0.839	**0.967**
OpenStack	0.867	0.758	0.871*	0.787	0.200	0.200	0.722	0.696	0.743	0.743	0.764	0.733	0.213	0.871
BGL	0.573	0.758	**0.939**	0.128	0.854	0.227	0.711	0.835	0.69	0.723	0.787	**0.963***	**0.960**	**0.963**
HPC	0.839	**0.903***	0.824	0.574	0.817	0.354	0.325	0.788	0.830	0.784	0.654	0.887	0.824	**0.903**
Thunderb.	0.882	**0.941**	0.663	0.813	0.649	0.694	0.576	0.599	**0.943**	**0.919**	0.844	**0.955***	0.646	**0.955**
Windows	0.697	0.690	0.567	**0.990**	0.588	0.689	0.701	0.713	0.566	**0.993**	**0.989**	**0.997***	0.406	**0.997**
Linux	0.297	0.673	0.672	0.519	0.279	0.169	0.701	0.629	0.701*	0.612	0.605	0.690	0.284	0.701
Mac	0.558	0.764	0.673	0.369	0.599	0.478	0.595	0.604	0.698	0.872*	0.757	0.787	0.636	0.872
Android	0.882	0.682	0.712	**0.909**	0.616	0.548	0.585	0.798	0.880	0.504	**0.919***	**0.911**	0.788	**0.919**
HealthApp	0.331	0.568	0.822*	0.592	0.549	0.235	0.397	0.531	0.174	0.684	0.639	0.780	0.440	0.822
Apache	0.731	**1***	**1***	**1***	**1***	0.582	**1***	0.709	**1***	**1***	**1***	**1***	**1***	**1**
OpenSSH	0.521	0.538	0.802	0.426	0.501	0.373	0.619	0.426	**0.925***	0.431	0.554	0.788	0.500	**0.925**
Proxifier	0.518	0.518	0.515	0.495	0.026	**0.967***	0.517	**0.951**	0.508	0.517	0.527	0.527	0.013	**0.967**
Average	0.637	0.754	0.777	0.563	0.652	0.482	0.669	0.665	0.721	0.694	0.751	0.865*	0.605	N.A.

the entire log set. LKE, LogSig and LogMine only have an offline mode. *Heuristics-based* methods include AEL [33], IPLoM [48] and Drain [26]. Specifically, AEL sorts log messages into groups by checking the frequency of static and variable tokens. IPLoM uses a step-by-step division method to group messages based on length, token location, and mapping connection. Drain uses a fixed-depth tree model to display log messages and quickly extracts common patterns. These methods use log features effectively and often yield good results. *Other* methods include Spell [14] and MoLFI [50]. Spell applies an algorithm based on the longest common subsequence for continuous log parsing. MoLFI formulates log parsing as a problem of multi-objective optimization, and resolves it through evolutionary algorithms.

2) Benchmarking on Loghub: To evaluate the accuracy of different log parsing algorithms. We define the metric of parsing accuracy (PA) as follows:

$$PA = \frac{\#\ of\ corrected\ parsed\ logs}{\#\ of\ total\ logs} \quad (1)$$

After parsing, every log message transforms into an event template, and each template relates to a cluster of log messages sharing the same template. A log message is considered correctly parsed if and only if its event template matches the same cluster of log messages as the groudtruth does. For instance, parsing a log sequence [E1, E2, E2] to [E1, E4, E5] results in a Parsing Accuracy (PA) of 1/3, as the second and third messages are not grouped similarly.

We benchmark these log parsing algorithms in Table II using the datasets in Loghub, and the experimental results are shown in Table III. For each dataset, the best accuracy is highlighted using an asterisk "*" and shown in the column "Best", and the PA higher than 0.9 are marked as boldface. We can summarize the following observations: (1) Over 90% accuracy is achieved by at least one log parser on most datasets, with 8 out of 13 log parsers showing the best accuracy on at least two datasets. Some parsers can even handle HDFS and Apache datasets with 100% accuracy due to their simpler event templates. (2) Despite these successes, complex log types like OpenStack, Linux, Mac, and HealthApp still present challenges due to their intricate structures and numerous event templates, such as the 341 templates in Mac logs. This calls for further improvements in parsing these complex log data. (3) Regarding the overall effectiveness of log parsers, Drain stands out with the highest average accuracy across different datasets, showing high precision on 9 out of 16 datasets. IPLoM, AEL, and Spell also perform well, maintaining high accuracy on 6 datasets. Conversely, LogSig, LFA, MoLFI, and LKE have the lowest average accuracy.

3) Remaining Questions and Challenges: Based on the benchmarking results, the following questions and challenges can be summarized: (1) No single log parser can effectively handle all types of datasets. This requires the development of a more generalized log parsing algorithm capable of handling diverse types of log data. (2) Dealing with complex datasets containing a substantial number of templates poses a significant challenge for most existing methods. A practical log parser should possess the ability to accurately handle such logs. (3) Existing log parsers can only differentiate between log templates and log parameters. It would potentially enhance the diagnostic procedures of on-site engineers if log parameters could be classified into various fine-grained types, such as status codes and block IDs. (4) The majority of log parsers rely on identifying frequent patterns to determine templates. However, log message occurrence distributions in practical scenarios can be diverse, resulting in instances where

TABLE IV
COMPRESSION EFFECTIVENESS OF EXISTING LOG COMPRESSION APPROACHES.

Tools	HDFS		Spark		Android		Windows		Thunderbird	
	Size	CR	Size	CR	Size	CR	Size	CR	Size	CR
Raw data	1,618	1	3,011	1	3,707	1	27,648	1	30,720	1
Cowic	373.6	4.3	707.4	4.3	1196.7	3.1	2794.0	9.9	8418.1	3.6
LogArchive	114.2	14.2	102.1	29.5	278.7	13.3	271.5	101.8	1146.4	26.8
gzip	149	10.9	175	17.2	439	8.4	1,638	16.9	1,946	15.8
logzip (gzip)	72	22.5	112	26.9	229	16.2	108	256.0	926	33.2
Improvement	51.7%	2.1x	36.0%	1.6x	47.8%	1.9x	93.4%	15.1x	52.4%	2.1x
bzip2	108	15.0	107	28.1	257	14.4	396	69.8	1,229	25.0
logzip (bzip2)	63	15.0	85	35.4	145	25.6	85	325.3	723	42.5
Improvement	41.7%	1.7x	20.6%	1.3x	43.6%	1.8x	78.5%	4.7x	41.2%	1.7x
lzma	96	16.9	122	24.7	167	22.2	118	234.3	1,126	27.3
logzip (lzma)	61	26.5	72	41.8	122	30.4	34	813.2	704	43.6
Improvement	36.5%	1.6x	41.0%	1.7x	26.9%	1.4x	71.2%	3.5x	37.5%	1.6x

certain log messages appear only a few times. This leads to decreased parsing accuracy.

B. Benchmarking for Log Compression

In this section, we demonstrate a case study of bechmarking existing log compression algorithms using loghub.

1) Existing Log Compression Algorithms: We evaluate 6 compression tools which can be categorized to *log-specific compression tools* including Cowic [40], LogArchive [9] and Logzip [45] and *general compression* tools including gzip, bzip2 and lzma. Note that Logzip should be utilized in conjunction with general compression tools (called compression kernels of Logzip), resulting in three variants: Logzip (gzip), Logzip (bzip2), and Logzip (lzma). Specifically, log-specific compression tools are specifically designed to compress log data by exploiting the inherent structures present in logs. This specialized approach enables these tools to achieve improved compression ratios for log files. On the other hand, general compression tools are designed to compress various types of files. These tools primarily identify redundant sequences of bytes within the compression objects and employ Huffman Coding techniques to minimize the required storage space.

2) Benchmarking on Loghub: To evaluate the effectiveness of the tools for log compression, we use the metric named compression ratio defined as follows:

$$CR = \frac{Original\ File\ Size}{Compressed\ File\ Size},$$

where the compressed file size is obtained after applying a compression tool. A smaller compressed file size can result in a higher CR, indicating a more effective compression tool.

We utilize five large-scale datasets within Loghub to benchmark these compression tools in terms of CR. The experimental results are shown in Table IV. We can make the following observations: (1) Among the general compression tools (lzma, bzip2, and gzip), lzma is the most effective on most datasets, followed by bzip2, while gzip performs the worst. (2) Two algorithms designed specifically for log data, LogArchive and Cowic, offer varying performance. LogArchive achieves a higher CR than gzip but is less effective than bzip2 and lzma, while Cowic performs worse than gzip because it prioritizes quick queries instead of a high CR. (3) Logzip, equipped with different compression kernels, outperforms these methods, with the compressed size determined by the kernel's effectiveness. It achieves higher CR on all five datasets, with an average CR of 4.56x and a maximum of 15.1x over gzip, resulting in significant storage savings. Similar results are observed with other compression kernels.

3) Remaining Questions and Challenges: According to the benchmarking results, we can summarize the following remaining questions and challenges. (1) The effectiveness of log-specific compression tools is impacted by the number of templates within the target log data. For example, the CR of different datasets varies a lot. Hence, it is crucial to conduct an in-depth study into how template distribution influences the performance of log-specific compression tools. (2) Current log-specific compression tools primarily concentrate on optimizing the CR, often at the expense of search efficiency. Practical applications frequently require on-site engineers to locate specific templates or keywords within compressed log files. Consequently, the issue of simultaneously achieving high CR and maintaining robust search efficiency remains an open question in the field. (3) The resource consumption of existing compression tools is still understudied. Log compression algorithms can be deployed on nodes with limited computational capacity or memory in real-world scenarios. Given their function as background processes, these algorithms must be sufficiently lightweight to prevent excessive overhead imposition on the host machine, an aspect yet to be thoroughly investigated.

C. Benchmarking for Anomaly Detection

In this section, we demonstrate the usage of loghub by a case study on benchmarking existing log-based anomaly detection approaches.

1) Existing Log-based Anomaly Detection Approaches: We have evaluated the performance of 9 log-based anomaly

TABLE V
SUMMARY OF LOG-BASED ANOMALY DETECTION.

Anomaly Detection Approach	Technique	Mode	Industrial Use
SVM	Classification	Supervised	IBM
Decision Tree	Classification	Supervised	eBay
LR	Classification	Supervised	Microsoft
Clustering	Clustering	Unsupervised	Microsoft
Invariant Mining	Execution Flow Mining	Unsupervised	Microsoft
PCA	Dimension Reduction	Unsupervised	Google
One-Class SVM	Classification	Unsupervised	Microsoft
Isolation Forest	Proper Binary Tree	Unsupervised	H2O.ai
LOF	Clustering	Unsupervised	N.A.

detection approaches, which are illustrated in Table V. There are mainly two categories of anomaly detection approaches: *supervised* and *unsupervised*.

Supervised approaches (Decision Tree [6], SVM [38], and LR [3]) require training data that contain labels indicating whether an instance is an anomaly. A classifier is trained based on the labeled data for anomaly detection. Supervised approaches are used when there are decent amount of both normal and abnormal labeled data. *Unsupervised* approaches are based on different techniques, including classification (LR [60]), isolation via proper binary tree (Isolation Forest [42]), dimension reduction (PCA [72]), execution flow mining (Invariant Mining [47]), and clustering (LOF [4] and Clustering [41]). The core idea of unsupervised approach is to learn the common patterns in logs or log sequences, and report instances the deviating instances as anomalies. In practice, labeled data are often lacking, because (1) anomalies rare occur in real-world systems and (2) data labeling is label-intensive and time-consuming. Thus, unsupervised methods are more applicable in real-world production environment.

2) Benchmarking on Loghub: To evaluate the accuracy of anomaly detection, we use *precision*, *recall*, and *F-measure* (i.e., F1 score), which are the most commonly-used metrics [27].

All the anomaly detection approaches are evaluated on the labeled HDFS dataset. This dataset records the system operations on different HDFS blocks. After log parsing and some postprocessing, we can obtain the input: a *block-ID*-by-*event count* matrix. Each row of the matrix represents the system operations on a specific block, while each column represents the frequency of occurrence of a system event during runtime.

The experimental results are illustrated in Figure 3 and Figure 4. *Decision Tree* obtains the highest recall (0.99), F-measure (0.99), and the second-highest precision (0.99). The supervised approaches achieve better results. This is because the supervised approaches are trained on labeled data. Additionally, among the unsupervised approaches, This is because the supervised approaches are trained on labeled data.

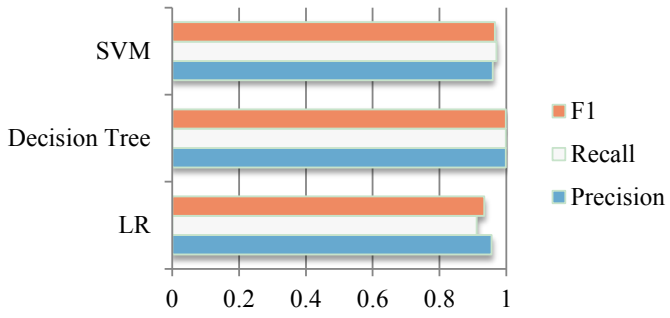

Fig. 3. Accuracy of Supervised Anomaly Detection Approaches

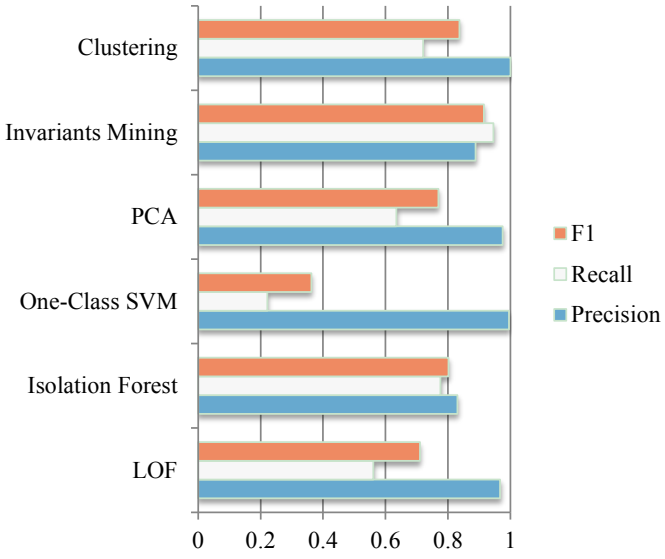

Fig. 4. Accuracy of Unsupervised Anomaly Detection Approaches

Additionally, among the unsupervised approaches, *Invariants Mining* approach has the best accuracy. *Clustering* obtains higher precision (1.00) than Invariants Mining. However, its recall (0.72) is much lower. Although *One-Class SVM* obtain high precision (0.99), its recall is too low (0.22), leading to low F-measure. This is because One-Class SVM is too conservative on reporting anomalies.

In additional to the labeled HDFS dataset used to evaluate anoamly detection method in this section, loghub also provides other 4 labeled datasets. Thus, in practice, developers could evaluate an anomaly detection method on different datasets and choose the most suitable one.

3) Remaining Questions and Challenges: After benchmarking existing anomaly detection approaches on loghub, we find some remaining questions and challenges, which will be discussed in this session. (1) The unsupervised approaches are not as accurate as the supervised approaches. Thus, an accurate unsupervised anomaly detection approach is highly in demand. (2) In practice, the number of anomalies is much less than that of normal instances. In industry, a system may only encounter 1 anomaly in a year, which makes supervised approaches ineffective. Thus, how to design an anomaly

detection approach that does not require historical abnormal instances remains an important and challenging problem. (3) Current anomaly detection approaches are all log sequences-based, which provide limited help to developers on further diagnosis, such as root cause analysis. (4) Most of the existing approaches will only report whether an instance is an anomaly without other information. Thus, there is a lack of visualization tools that help developers understand the reported anomalies. (5) Existing approaches mainly focus on improving accuracy for anomaly detection. However, a practical solution should be efficient enough to process a large number of log data in the runtime. (6) In real-world scenarios, log data evolves alongside software development. This phenomenon, known as concept drift, poses a challenge to existing methods in maintaining satisfactory performance.

V. Related Work

Logging Practice. In practice, there is a lack of rigorous guide and specifications on developer logging behaviors. To address this problem, a line of recent empirical research has focused on studying the logging practice of high-quality software [1], [5], [20], [35], [56], [76], [75]. Additionally, AI-based logging decisions have also been widely studied in recent years with a focus on "where to log" and "what to log". *Where-to-log*: The work [80] proposed a "learning to log" framework, which provides guidance on whether developers should put logging statement in a code snippet. *What-to-log*: Li et al. [36] developed an log verbosity level suggestion technique based on ordinal regression model. He et al. [24] characterizes the natural language descriptions in logging statements and explore the potential of automated description text generation for logging statements. AI-based strategic logging practices is an important component in AI-driven log analytics, where different practice will lead to different software logs at runtime.

Log Compression. Different from general natural language text, software logs have specific inherent structure (e.g., many logs are printed by the same logging statement). Thus, to achieve better compression rate, compression approaches specialized for log files have been well studied [22], [57], [12], [40], [16]. For example, Comprehensive Log Compression (CLC) [22] and Differentiated Semantic Log Compression (DSLC) [57] identify repetitive items by domain knowledge. Logzip [45] extracts redundant log templates using log parsing technologies, which improves over existing compression tools such as gzip. The work [70] further improves logzip with a more efficient implementation for compressing huge logs in cloud systems. Beyond compression, CLP [59] and Loggrep [69] achieve more search efficiency on compressed log data without decompression. Logreducer [74] detects log hot spot in the runtime to avoid saving massive and redundant logs. All the datasets in loghub can facilitate the evaluation of log compression approaches.

Log Parsing. Most automated and effective log analysis techniques require structured data as input. Therefore, log parsing is crucial in AI-driven log analytics, which transforms unstructured log messages into structured system events. In recent years, log parsers based on various techniques have been proposed. (1) *Frequent pattern mining*: SLCT [65], LFA [54], and LogCluster [66] regard log event templates as a set of constant tokens that occur frequently in log. (2) *Clustering*: a line of log parsing studies model it as a clustering problem and design specialized clustering algorithms accordingly. Typical parsers in this category include LKE [19], LogSig [63], LogMine [21], SHISO [52], and LenMa [62]. (3) *Heuristics*: AEL [33], IPLoM [48], Drain [23] and SPINE [68] parse log messages by heuristic rules inspired by unique characteristics of software logs. It is worth noting that Drain [23] has been a widely adopted solution in industry, such as IBM. (4) *Deep Learning*: UniParser [46]. (5) *Others*: Some other methods exist, such as Spell [14] that is based on longest common subsequence. All the datasets in loghub can be employed to evaluate log parsing methods.

Log Analysis. Log analysis has been studied for decades to facilitate effective and efficient system maintenance [8]. Typical log analysis tasks include anomaly detection [72], [18], [27], [15], duplicate issue identification [13], [39], [58], incident diagnosis [77], usage statistics analysis [34], and program verification [2], [61], most of which design or adopt AI algorithms. For example, Xu et al. [72] employs principal component analysis (PCA), which is a dimension reduction algorithm, to detect potential anomalies in large-scale distributed systems. Du et al. [15] proposed an anomaly detection and diagnosis approach based on a deep neural network model utilizing Long Short-Term Memory (LSTM). Shang et al. [61] design a clustering algorithm to verify the deployment of big data applications. The 5 labeled datasets in loghub can be used to evaluate various log analysis methods.

VI. Conclusion and Future Work

This paper describes loghub, a large collection of log datasets for AI-driven log analytics. Loghub contains 19 log datasets where all the logs amount to over 77 GB. We present both common usage scenarios and benchmarking results for typical log analysis tasks including log parsing, log compression, and log-based anomaly detection. Based on these benchmarking results, we also discuss open challenges and future directions. We envision loghub acting as an open benchmarking system for AI-driven log analysis to benefit researchers and practitioners from academia and industry. As part of our future work, we plan to collect more large-scale and more abundant log datasets to fill the gap between research and practice. Additionally, we aim to build a benchmarking leaderboard based on loghub to assess log analysis models and tasks.

VII. Acknowledgement

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).

REFERENCES

[1] T. Barik, R. DeLine, S. Drucker, and D. Fisher, "The bones of the system: A case study of logging and telemetry at microsoft," in *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 2016, pp. 92–101.

[2] I. Beschastnikh, Y. Brun, S. Schneider, M. Sloan, and M. D. Ernst, "Leveraging existing instrumentation to automatically infer invariant-constrained models," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering (FSE)*. ACM, 2011, pp. 267–277.

[3] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen, "Fingerprinting the datacenter: automated classification of performance crises," in *Proceedings of the 5th European Conference on Computer Systems (EuroSys)*. ACM, 2010, pp. 111–124.

[4] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *ACM sigmod record*, vol. 29, no. 2. ACM, 2000, pp. 93–104.

[5] B. Chen and Z. M. J. Jiang, "Characterizing logging practices in java-based open source software projects–a replication study in apache software foundation," *Empirical Software Engineering*, vol. 22, no. 1, pp. 330–374, 2017.

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

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

[8] Q. Cheng, A. Saha, W. Yang, C. Liu, D. Sahoo, and S. Hoi, "Logai: A library for log analytics and intelligence," *arXiv preprint arXiv:2301.13415*, 2023.

[9] R. Christensen and F. Li, "Adaptive log compression for massive log data," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2013, pp. 1283–1284.

[10] A. Chuvakin. (2019) Public security log sharing site project. [Online]. Available: http://log-sharing.dreamhosters.com/

[11] CloudLab. (2019) Cloudlab. [Online]. Available: https://cloudlab.us/

[12] S. Deorowicz and S. Grabowski, "Sub-atomic field processing for improved web log compression," in *Modern Problems of Radio Engineering, Telecommunications and Computer Science, 2008 Proceedings of International Conference on*. IEEE, 2008, pp. 551–556.

[13] R. Ding, Q. Fu, J. G. Lou, Q. Lin, D. Zhang, and T. Xie, "Mining historical issue repositories to heal large-scale online service systems," in *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2014, pp. 311–322.

[14] M. Du and F. Li, "Spell: Streaming parsing of system event logs," in *ICDM'16 Proc. of the 16th International Conference on Data Mining*, 2016.

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

[16] B. Feng, C. Wu, and J. Li, "MLC: an efficient multi-level log compression method for cloud backup systems," in *2016 IEEE Trustcom/BigDataSE/ISPA, Tianjin, China, August 23-26, 2016*, 2016, pp. 1358–1365.

[17] Q. Fu, J. Zhu, W. Hu, J. Lou, R. Ding, Q. Lin, D. Zhang, and T. Xie, "Where do developers log? an empirical study on logging practices in industry," in *ICSE'14: Companion Proc. of the 36th International Conference on Software Engineering*, 2014, pp. 24–33.

[18] Q. Fu, J. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *ICDM 2009, The Ninth IEEE International Conference on Data Mining, Miami, Florida, USA, 6-9 December 2009*, 2009, pp. 149–158.

[19] ——, "Execution anomaly detection in distributed systems through unstructured log analysis," in *ICDM 2009, The Ninth IEEE International Conference on Data Mining, Miami, Florida, USA, 6-9 December 2009*, 2009, pp. 149–158.

[20] Q. Fu, J. Zhu, W. Hu, J.-G. Lou, R. Ding, Q. Lin, D. Zhang, and T. Xie, "Where do developers log? an empirical study on logging practices in industry," in *Companion Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 24–33.

[21] H. Hamooni, B. Debnath, J. Xu, H. Zhang, G. Jiang, and A. Mueen, "Logmine: Fast pattern recognition for log analytics," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, 2016, pp. 1573–1582.

[22] K. Hätönen, J. Boulicaut, M. Klemettinen, M. Miettinen, and C. Masson, "Comprehensive log compression with frequent patterns," in *Data Warehousing and Knowledge Discovery, 5th International Conference, DaWaK 2003, Prague, Czech Republic, September 3-5,2003, Proceedings*, 2003, pp. 360–370.

[23] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *ICWS'17: Proc. of the 24th International Conference on Web Services*, 2017.

[24] P. He, Z. Chen, S. He, and M. R. Lyu, "Characterizing the natural language descriptions in software logging statements," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 2018, pp. 178–189.

[25] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "An evaluation study on log parsing and its use in log mining," in *46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2016, Toulouse, France, June 28 - July 1, 2016*, 2016, pp. 654–661.

[26] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE International Conference on Web Services, ICWS 2017, Honolulu, HI, USA, June 25-30, 2017*, 2017, pp. 33–40.

[27] S. He, J. Zhu, P. He, and M. Lyu, "Experience report: System log analysis for anomaly detection," in *ISSRE'16: Proc. of the 27th International Symposium on Software Reliability Engineering*, 2016.

[28] S. He, P. He, Z. Chen, T. Yang, Y. Su, and M. R. Lyu, "A survey on automated log analysis for reliability engineering," vol. 54, no. 6. New York, NY, USA: Association for Computing Machinery, jul 2021.

[29] S. He, Q. Lin, J.-G. Lou, H. Zhang, M. R. Lyu, and D. Zhang, "Identifying impactful service system problems via log analysis," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 60–70.

[30] S. He, X. Zhang, P. He, Y. Xu, L. Li, Y. Kang, M. Ma, Y. Wei, Y. Dang, S. Rajmohan, and Q. Lin, "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*, ser. ESEC/FSE 2022, 2022, p. 1465–1476.

[31] Y. Huo, C. Lee, Y. Su, S. Shan, J. Liu, and M. Lyu, "Evlog: Evolving log analyzer for anomalous logs identification," *arXiv preprint arXiv:2306.01509*, 2023.

[32] Y. Huo, Y. Su, C. Lee, and M. R. Lyu, "Semparser: A semantic parser for log analytics," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 881–893.

[33] Z. M. Jiang, A. E. Hassan, P. Flora, and G. Hamann, "Abstracting execution logs to execution events for enterprise applications (short paper)," in *2008 The Eighth International Conference on Quality Software*. IEEE, 2008, pp. 181–186.

[34] G. Lee, J. J. Lin, C. Liu, A. Lorek, and D. V. Ryaboy, "The unified logging infrastructure for data analytics at twitter," *PVLDB*, vol. 5, no. 12, pp. 1771–1780, 2012. [Online]. Available: http://vldb.org/pvldb/vol5/p1771_georgelee_vldb2012.pdf

[35] H. Li, T. Chen, W. Shang, and A. E. Hassan, "Studying software logging using topic models," *Empirical Software Engineering*, 2017.

[36] H. Li, W. Shang, and A. E. Hassan, "Which log level should developers choose for a new logging statement?" *Empirical Software Engineering*, vol. 22, pp. 1684–1716, 2017.

[37] Y. Liang, Y. Zhang, A. Sivasubramaniam, R. K. Sahoo, J. Moreira, and M. Gupta, "Filtering failure logs for a bluegene/l prototype," in *2005 International Conference on Dependable Systems and Networks (DSN'05)*. IEEE, 2005, pp. 476–485.

[38] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, "Failure prediction in ibm bluegene/l event logs," in *the 7th IEEE International Conference on Data Mining (ICDM)*. IEEE, 2007, pp. 583–588.

[39] M.-H. Lim, J.-G. Lou, H. Zhang, Q. Fu, A. B. J. Teoh, Q. Lin, R. Ding, and D. Zhang, "Identifying recurrent and unknown performance issues," in *2014 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2014, pp. 320–329.

[40] H. Lin, J. Zhou, B. Yao, M. Guo, and J. Li, "Cowic: A column-wise independent compression for log stream analysis," in *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2015, pp. 21–30.

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

[42] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *the 8th IEEE International Conference on Data Mining (ICDM)*. IEEE, 2008, pp. 413–422.

[43] J. Liu, S. He, Z. Chen, L. Li, Y. Kang, X. Zhang, P. He, H. Zhang, Q. Lin, Z. Xu *et al.*, "Incident-aware duplicate ticket aggregation for cloud systems," *arXiv preprint arXiv:2302.09520*, 2023.

[44] J. Liu, J. Huang, Y. Huo, Z. Jiang, J. Gu, Z. Chen, C. Feng, M. Yan, and M. R. Lyu, "Scalable and adaptive log-based anomaly detection with expert in the loop," *arXiv preprint arXiv:2306.05032*, 2023.

[45] J. Liu, J. Zhu, S. He, P. He, Z. Zheng, and M. R. Lyu, "Logzip: Extracting hidden structures via iterative clustering for log compression," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 863–873.

[46] Y. Liu, X. Zhang, S. He, H. Zhang, L. Li, Y. Kang, Y. Xu, M. Ma, Q. Lin, Y. Dang, S. Rajmohan, and D. Zhang, "Uniparser: A unified log parser for heterogeneous log data," in *Proceedings of the ACM Web Conference 2022*, ser. WWW '22. Association for Computing Machinery, 2022, p. 1893–1901.

[47] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection." in *USENIX Annual Technical Conference*, 2010, pp. 1–14.

[48] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "Clustering event logs using iterative partitioning," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*, 2009, pp. 1255–1264.

[49] C. Manning, P. Raghavan, and H. Schutze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[50] S. Messaoudi, A. Panichella, D. Bianculli, L. Briand, and R. Sasnauskas, "A search-based approach for accurate identification of log message formats," in *Proceedings of the 26th Conference on Program Comprehension*. ACM, 2018, pp. 167–177.

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

[52] M. Mizutani, "Incremental mining of system log format," in *2013 IEEE International Conference on Services Computing, Santa Clara, CA, USA, June 28 - July 3, 2013*, 2013, pp. 595–602.

[53] P. Mosendz. (2014) When it goes down, facebook loses $24,420 per minute. [Online]. Available: https://www.theatlantic.com/technology/archive/2014/10/facebook-is-losing-24420-per-minute/382054/

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

[55] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *DSN*, 2007.

[56] A. Pecchia, M. Cinque, G. Carrozza, and D. Cotroneo, "Industry practices and event logging: Assessment of a critical software development process," in *Proceedings of the 37th International Conference on Software Engineering-Volume 2*. IEEE Press, 2015, pp. 169–178.

[57] B. Rácz and A. Lukács, "High density compression of log files," in *2004 Data Compression Conference (DCC 2004), 23-25 March 2004, Snowbird, UT, USA*, 2004, p. 557.

[58] M. S. Rakha, C.-P. Bezemer, and A. E. Hassan, "Revisiting the performance evaluation of automated approaches for the retrieval of duplicate issue reports," *IEEE Transactions on Software Engineering*, vol. 44, no. 12, pp. 1245–1268, 2018.

[59] K. Rodrigues, Y. Luo, and D. Yuan, "{CLP}: Efficient and scalable search on compressed text logs," in *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*, 2021, pp. 183–198.

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

[61] W. Shang, Z. M. Jiang, H. Hemmati, B. Adams, A. E. Hassan, and P. Martin, "Assisting developers of big data analytics applications when deploying on hadoop clouds," in *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, 2013, pp. 402–411.

[62] K. Shima, "Length matters: Clustering system log messages using length of words," *arXiv preprint arXiv:1611.03213*, 2016.

[63] L. Tang, T. Li, and C. Perng, "Logsig: generating system events from raw textual logs," in *Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011, Glasgow, United Kingdom, October 24-28, 2011*, 2011, pp. 785–794.

[64] UpGuard. (2016) The cost of downtime at the world's biggest online retailer. [Online]. Available: https://www.upguard.com/blog/the-cost-of-downtime-at-the-worlds-biggest-online-retailer

[65] R. Vaarandi, "A data clustering algorithm for mining patterns from event logs," in *IP Operations & Management, 2003.(IPOM 2003). 3rd IEEE Workshop on*. IEEE, 2003, pp. 119–126.

[66] R. Vaarandi and M. Pihelgas, "Logcluster-a data clustering and pattern mining algorithm for event logs," in *2015 11th International Conference on Network and Service Management (CNSM)*. IEEE, 2015, pp. 1–7.

[67] X. Wang, X. Zhang, L. Li, S. He, H. Zhang, Y. Liu, L. Zheng, Y. Kang, Q. Lin, Y. Dang *et al.*, "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*, 2022, pp. 1198–1208.

[68] X. Wang, X. Zhang, L. Li, S. He, H. Zhang, Y. Liu, L. Zheng, Y. Kang, Q. Lin, Y. Dang, S. Rajmohan, and D. Zhang, "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*, ser. ESEC/FSE 2022. Association for Computing Machinery, 2022, p. 1198–1208.

[69] J. Wei, G. Zhang, J. Chen, Y. Wang, W. Zheng, T. Sun, J. Wu, and J. Jiang, "Loggrep: Fast and cheap cloud log storage by exploiting both static and runtime patterns," in *Proceedings of the Eighteenth European Conference on Computer Systems*, 2023, pp. 452–468.

[70] J. Wei, G. Zhang, Y. Wang, Z. Liu, Z. Zhu, J. Chen, T. Sun, and Q. Zhou, "On the feasibility of parser-based log compression in {Large-Scale} cloud systems," in *19th USENIX Conference on File and Storage Technologies (FAST 21)*, 2021, pp. 249–262.

[71] J. Xu, Q. Fu, Z. Zhu, Y. Cheng, Z. Li, Y. Ma, and P. He, "Hue: A user-adaptive parser for hybrid logs," *arXiv preprint arXiv:2308.07085*, 2023.

[72] W. Xu, L. Huang, A. Fox, D. A. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *SOSP*, 2009, pp. 117–132.

[73] ——, "Detecting large-scale system problems by mining console logs," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, 2010, pp. 37–46.

[74] G. Yu, P. Chen, P. Li, T. Weng, H. Zheng, Y. Deng, and Z. Zheng, "Logreducer: Identify and reduce log hotspots in kernel on the fly," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 1763–1775.

[75] D. Yuan, S. Park, P. Huang, Y. Liu, M. M. Lee, X. Tang, Y. Zhou, and S. Savage, "Be conservative: enhancing failure diagnosis with proactive logging," in *Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12)*, 2012, pp. 293–306.

[76] D. Yuan, S. Park, and Y. Zhou, "Characterizing logging practices in open-source software," in *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press, 2012, pp. 102–112.

[77] X. Zhang, Y. Xu, S. Qin, S. He, B. Qiao, Z. Li, H. Zhang, X. Li, Y. Dang, Q. Lin, M. Chintalapati, S. Rajmohan, and D. Zhang, "Onion: Identifying incident-indicating logs for cloud systems," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2021. Association for Computing Machinery, 2021, p. 1253–1263.

[78] J. Zhou, Z. Chen, H. Mi, and J. Wang, "Mtracer: A trace-oriented monitoring framework for medium-scale distributed systems," in *8th IEEE International Symposium on Service Oriented System Engineering (SOSE)*, 2014, pp. 266–271.

[79] J. Zhou, Z. Chen, J. Wang, Z. Zheng, and M. R. Lyu, "Trace bench: An open data set for trace-oriented monitoring," in *IEEE 6th International Conference on Cloud Computing Technology and Science (CloudCom)*, 2014, pp. 519–526.

[80] J. Zhu, P. He, Q. Fu, H. Zhang, M. R. Lyu, and D. Zhang, "Learning to log: Helping developers make informed logging decisions," in *ICSE*, 2015.

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