Noname manuscript No.
(will be inserted by the editor)

A Literature Review and Existing Challenges on Software Logging Practices

From the Creation to the Analysis of Software Logs

Mohamed Amine Batoun · Mohammed Sayagh · Roozbeh Aghili · Ali Ouni · Heng Li

Received: date / Accepted: date

Abstract Software logging is the practice of recording different events and activities that occur within a software system, which are useful for different activities such as failure prediction and anomaly detection. While previous research focused on improving different aspects of logging practices, the goal of this paper is to conduct a systematic literature review and the existing challenges of practitioners in software logging practices. In this paper, we focus on the logging practices that cover the steps from the instrumentation of a software system, the storage of logs, up to the preprocessing steps that prepare log data for further follow-up analysis. Our systematic literature review (SLR) covers 204 research papers and a quantitative and qualitative analysis of 20,766 and 149 questions on StackOverflow (SO). We observe that 53% of the studies focus on improving the techniques that preprocess logs for analysis (e.g., the practices of log parsing, log clustering and log mining), 37% focus on how to create new log statements, and 10% focus on how to improve log file storage. Our analysis of SO topics reveals that five out of seven identified high-level topics are not covered by the literature and are related to dependency configuration of logging libraries, infrastructure related configuration, scattered logging, context-dependant usage of logs and handling log files.

Keywords Software Logging, Systematic Literature Review, Topic Modeling, Developers' Discussion

Mohamed Amine Batoun, Mohammed Sayagh, and Ali Ouni
École de Technologie Supérieure
1100 Notre-Dame St W
Montreal - Canada
E-mail: mohamed-amine.batoun.1@ens.etsmtl.ca, mohammed.sayagh@etsmtl.ca, and ali.ouni@etsmtl.ca

Roozbeh Aghili and Heng Li
Polytechnique Montreal
2500 Chem. de Polytechnique
Montreal - Canada
E-mail: roozbeh.aghili@polymtl.ca and heng.li@polymtl.ca

1 Introduction

Software logging refers to the process of recording different events and activities that occur within a software system, which can be leveraged by practitioners to get feedback about the run-time behavior of a software system(112). In practice, developers insert logging statements in the code such as `LOG.info( ‘‘User logged in: {}’’, username)`. In this example, `LOG` is the instance object of a logging framework (e.g., Log4j[1]), `info` is the log verbosity level and `(‘‘User logged in: {}’’, username)` is the log message that contains a static text (i.e., `‘‘User logged in: {}’’`) and a dynamic variable (i.e., `username`) that will replace the placeholder "`{}`". Developers log their systems to help them trace the execution flow of the application, identify exceptions and potential issues and debug them, whereas operators use logs to detect (161), diagnose (168) and even predict (116) issues that can impact the availability and performance of their software system.

Practitioners face a large number of challenges when developing, maintaining and using their logs (58). A major challenge is to make informed decisions on where to log, what to log and which log verbosity level to use, in order to balance the level of details captured in logs with the performance overhead and the resources required to store and process log data. Additionally, software systems generate huge amounts of log files that grow larger by the day and quickly become overwhelming to manage, which may hinder the efficient storage and protection of log data that contains sensitive information (13). Furthermore, it is challenging to manually analyze the sheer volume of generated log files, which can have different formats and structures, and identify relevant information and extract useful insights for debugging or diagnosis (175).

A large body of research exists to address such challenges and assist different stakeholders in their logging activities. For example, researchers have studied logging challenges, benefits and practices in industrial and open-source systems (e.g., (37; 136; 58)) and suggested guidelines and best practices to help developers make informed logging decisions, and/or capture the necessary information while minimizing the performance and resources overhead (e.g., (73; 59; 30)). Recent studies also proposed novel approaches and frameworks to manage the large volumes of log data (e.g., (50; 125)) and to ensure the security and integrity of such data (e.g., (51; 14)). Researchers also focused on automating preprocessing log files for following-up log analysis tasks to reduce manual work done by developers and operators, as it is error prone and overwhelming (e.g., (95; 110)).

While there are a few literature reviews on software logging, they cover a sub-areas of software logging. No studies exist that systematically provide a larger picture of the state-of-the-art in the maintenance and the analysis of software logging and in-comparison with the state of practice. For example, previous works (20; 36; 45) studied the challenges and practices of log instru-

[1] Open-source logging library for Java applications: `https://github.com/apache/logging-log4j2`

mentation, while others (32; 66; 93; 153; 190) reviewed existing automated log analysis techniques. The closest study to our paper is the work of He et al. (164). While they only summarize 25 existing studies on log instrumentation, they did not deeply cover the methodologies of these studies. Besides, our study covers a deep investigation of 83 papers related to the instrumentation. In fact, they mainly focus on log analysis such as log parsing, anomaly detection, and failure prediction. In our paper, we cover log analysis but only the set of practices that preprocess log files for follow-up anomaly and failure detection and prediction. Anomaly detection and failure prediction techniques are outside the scope of our paper. Finally, our paper is different from these existing studies as we explore the problems practitioners face in practice.

In this paper, we aim to better understand the current state of software logging in both research and practice, by conducting a study that combines (1) a systematic literature review (SLR) of 204 papers on the logging practices from the creation to the analysis of logs (i.e., preprocessing steps that prepare log data for follow-up activities such as anomaly detection and prediction), with (2) a qualitative and quantitative analysis of 149 and 20,766 software logging questions discussed in StackOverflow (SO), the largest questions & answers (Q&A) forum for developers. Note that we define log analysis in the remainder of this paper as the set of logging practices that preprocess a log file to prepare it for further analysis, such as debugging an issue or anomaly detection purposes. An example of these preprocessing techniques is log parsing, template generation, and clustering log lines. While an SLR captures the state of research, questions on SO represent practical real-world challenges faced by practitioners in the field of software development and programming. By combining the two studies, we aim to provide a comprehensive view of the body of knowledge to guide practitioners on existing solutions to their challenges. We also aim to find a potential gap between research and practice by studying the common challenges that developers encounter that are not addressed by the literature. For instance, the lack of information from SO-like sources has a negative impact on our ability to build a comprehensive understanding of logging practices.

Our study reveals that the most studied topic (53%) focuses on how to improve the fundamental components of log analysis through different preprocessing techniques such as log parsing and clustering, 37% of the studies focus on how to create new logging statements while 10%, focus on how to improve the storage of log files. However, our LDA analysis of the logging questions in SO reveals seven high-level topics (and 22 fine-grained topics) out of which five are not covered by the literature. An example of such topics is how to guide developers in the configuration of logs, including the configuration of dependencies and the infrastructure. These are common topics that pose several challenges to developers, as we observed a total of 18,623 questions about the configuration of logs, out of which only 7,102 have an accepted answer. Furthermore, such answers are given after an average time of 19.25 days (i.e., 462 hours). Finally, our findings offer guidance to developers on existing studies that cover all aspects of software logging from development to analysis

and offer future studies the opportunity to cover directions that are currently unexplored by the literature. We summarize our main contributions as follows:

- A large systematic literature review on the whole logging pipeline from the creation to the analysis of logs.
- A large analysis of developers posts in StackOverflow leading to the identification of seven high-level topics and 22 fine-grained topics related to challenges that practitioners reported and discussed in StackOverflow.
- A qualitative analysis of a random set of 119 StackOverflow questions that cover each of the 22 fine-grained topics.
- A quantitative analysis to understand which of the high-level topics are more important in terms of their popularity and difficulty.

Overall, our study reveals existing solutions for practitioners and opportunities for researchers to address open challenges. Our data used in this paper are provided in our replication package [2].

The rest of the paper is structured as follows: Section 2 presents a background on software logging. Section 3 explains our research methodology. Section 4 covers the results and the taxonomy obtained. Section 5 discusses our implications. Section 6 presents the threats to validity of our findings. Finally, Section 7 concludes the paper.

2 Background

This section provides a general background (summarized in Figure 1) on software logging and the log management pipeline, from instrumenting the code, storing log files, to the analysis of log files.

2.1 Instrumentation

Developers start by inserting log statements (Section 2.1.1) in the source code (i.e., instrumenting the code), so a software system generates log files (a log file example is shown in Figure 4) that can be used to capture and manage important run-time information about the behavior and performance of the software system.

2.1.1 ***Log statements:***

typically contain a text describing an event and any additional contextual information that may be useful for management and operations, such as IP addresses, user IDs or error messages (160). Furthermore, log statements also specify the **log verbosity level** (59), which represents a way to categorize logging messages based on their purposes. Figure 2 shows an example of log

[2] `https://github.com/MABATOUN/SLRReplicationPackage.git`

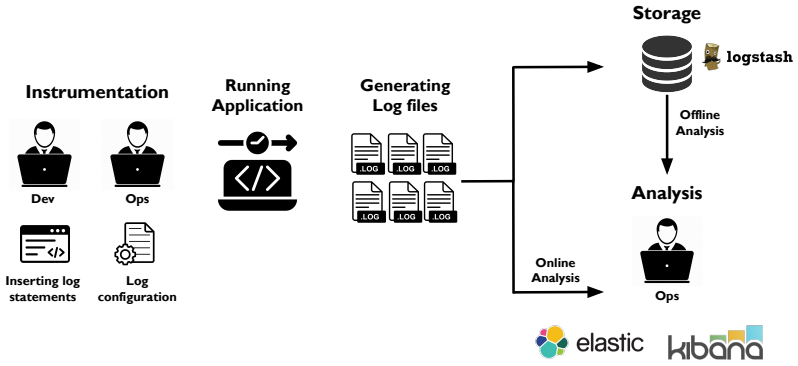


Fig. 1: Overview of the log management pipeline. The presented roles of developers and operators are high-level, while in other contexts such roles can overlap, or each role can be divided in sub-roles.

statements which log the progress information of a running task and an "error" when an exception is raised. The keywords "info" and "error" in the log statements are the log levels. Other log levels exist such as "fatal" to log fatal events and "debug" for logging debugging information. Note that the types and numbers of log levels can slightly change from one to another framework. For instance, Log4J has six log levels (i.e., "trace", "debug", "info", "warn", "error" and "fatal") while Python's built-in logging has five log levels (i.e., "debug", "info", "warning", "error" and "critical"). Log statements can be inserted in different locations in the code, e.g., within if-else statements, catch clauses, method declarations, try statements and loop statements (70).

```
while (inProgress){
    LOG.info("Progress of " + attemptId + " is: " + percentage);
    try {...} catch (Exception e){
        LOG.error("Container complete event for unknown container id " + containerId);
    }
    inProgress = attemptProgress(attemptId, percentage);
}
```

Fig. 2: Example of log statements

2.1.2 ***Log configuration:***

controls the logging behaviors (e.g., the output of log statements) from an external **configuration file**. Figure 3 shows an example of a configuration file that can be used with the Log4j framework. The first element of the configuration file is the *Appender*. In this example, three appenders are provided: the first appender outputs the log messages to the console (i.e., `''SYSTEM_OUT''`), the second one saves the logs in a file (i.e., `''app.log''` such as the one shown

in Figure 4), whereas the third one sends logs to "Elasticsearch" [3]. The pattern layout controls the format of how log statements are generated. In our example, a log line will start with a timestamp followed by the log level (e.g., *INFO*), the thread name, and finally the log message. Other configurations can be made through the same configuration file, such as controlling the verbosity of log files. For instance, one can output only log statements with certain log levels. In our example, only log messages with a log verbosity level higher or equal than *DEBUG* are logged, e.g., a log statement with the *WARN* level would be logged while a log statement with the *TRACE* level would not be logged using this configuration.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="INFO">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss} %-5level %t: %msg%n" />
        </Console>
        <File name="File" fileName="app.log">
            <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss} %-5level %t: %msg%n" />
        </File>
        <Elasticsearch name="Elasticsearch" type="http" index="logs" port="9200">
            <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss} %-5level %t: %msg%n" />
        </Elasticsearch>
    </Appenders>
    <Loggers>
        <Root level="debug">
            <AppenderRef ref="Console" />
            <AppenderRef ref="File" />
        </Root>
    </Loggers>
</Configuration>
```

Fig. 3: Example of a log configuration file

```
2023-05-10 09:02:39 INFO    TaskAttemptListenerImpl: Progress of TaskAttempt attempt_1445 is : 0.1068813
2023-05-10 11:29:05 INFO    TaskAttemptListenerImpl: Progress of TaskAttempt attempt_1445 is : 0.2000002
2023-05-10 12:56:42 INFO    TaskAttemptListenerImpl: MapCompletionEvents request from attempt_1445
2023-05-10 16:56:42 INFO    TaskAttemptListenerImpl: Progress of TaskAttempt attempt_1445 is : 0.5323719
2023-05-10 18:09:31 ERROR RMContainerAllocator: Container complete event for unknown container id container_1445
2023-05-10 19:56:37 INFO    CommitterEventHandler: Processing the event EventType: TASK_ABORT
2023-05-11 15:56:06 WARN   FileOutputCommitter: Could not delete temporary_file_1001
```

Fig. 4: Example of log entries in a log file

2.2 Storing log data

After running an application, the generated log data is often stored in a file, a database or a log management system (e.g., ELK Stack which consists of

[3] A search engine developed in Java, allowing an easy search and analysis of large amounts of data

Elasticsearch, LogStash [4] and Kibana [5]). For instance, in Figure 3 the logs are stored in a file named ‘‘`app.log`’’. Different techniques are considered when storing log files. For example, due to the large amount of log data that can be Terrabytes of generated log files every day (126), one can compress log files to reduce their size without losing information. Furthermore, it is necessary to store log data in a way that ensures their confidentiality, integrity and availability. In fact, log data can contain sensitive information (e.g., passwords, credit card numbers, etc.) which should not be exposed while still allowing access to logs for analysis purposes. This can be achieved in a variety of ways, including immutable log storage, log pseudonymization, etc. **Immutable log storage** prevents log data from being modified or deleted (198), i.e., once data is written to an immutable log it cannot be altered, ensuring its authenticity and integrity. Whereas **log pseudonymization** is a process of anonymizing sensitive information in log files by replacing it with pseudonyms (13; 14).

2.3 Analyzing log data

Log data is leveraged by both developers and operators for different purposes. Developers analyze log data to trace the execution flow of their application and identify and debug any potential issues with their code. Whereas operators use logs to diagnose issues, identify their root causes and predict failures that can impact the performance and availability of their systems. However, analyzing logs for such purposes can be challenging due to the sheer volume of log files, with their different formats and structures. Therefore, various techniques exist to facilitate and automate log analysis tasks including but not limited to: **log template identification**, **log parsing**, **log clustering** and **log mining** (further discussed in the results). In practice, there are also several tools for analyzing and visualizing logs, such as “Elasticsearch”, “Logstash”, and “Kibana”. Figure3 shows an example of an appender that sends the log data to “Elasticsearch” for analysis. By leveraging such tools and approaches, stakeholders can gain insights into the behavior of their software system, predict potential failures and take preventive measures and informed decisions to optimize the system performance.

2.4 Existing Systematic Literature Reviews

The existing systematic literature reviews on software logging are summarized in Table 1. While each of the existing studies cover a sub-area of software logging, we cover software logging from a larger perspective; from the instrumentation of a software system to the different techniques used to preprocess logs for following-up log analysis. Furthermore, we cover more recent papers

[4] An open server-side data processing pipeline that ingests, transforms and sends data from various sources to a certain destination

[5] A data visualization dashboard software for Elasticsearch

Paper	Topic	Number of Covered Papers	Years Covered	Number of Papers We Covered on the Same Topic
(20)	Log instrumentation	69	1997-2019	75
(36)	Log levels	27	Until 2020	12
(45)	Log instrumentation	41	2000-2016	75
(32)	Log parsing and abstraction	89	2000-2018	44
(66)	Log analysis	34	2011-2020	108
(93)	Log analysis	118	2000-2021	108
(153)	Log parsing	19	2013-2021	44
(190)	Log analysis	16	2009-2019	108
(164)	logging instrumentation, storage, and analysis	158	1997-2020	204

Table 1: Overview of Systematic Literature Reviews on Software Logging. Our study covers papers published between 2013 and June 2023.

than these existing systematic literature reviews for each of their covered topics. For example, while Boyuan et al. (20) covered 69 papers from 1997 to 2019 on log instrumentation, our paper covers 75 papers from 2013 to June 2023. Although Korzeniowski et al. (93) covered more papers than our study on log analysis, their focus is on papers published between 2000 and 2021, while our study is on papers published between 2013 and June 2023. In addition, we cover log analysis from different perspectives. While they analyzed the goal of log analysis, the business area of the studied logs, and the different types of logs that are used in research (e.g., Access log, Event log, Network log, ...), we define log analysis as the preprocessing steps for following up analysis and we focus on the goal of the preprocessing technique and the used methodology. As discussed in the introduction, the closest work to our SLR is the study by He et al. (164). In addition to the differences discussed in the introduction, our study covered three more years of logging practices and the state-of-practice in software logging.

3 Methodology

In this paper, we perform an systematic literature review (SLR) to identify the current state-of-the-art software logging, followed by a qualitative analysis of logging questions in StackOverflow (SO). Our aim is to provide a comprehensive view of the body of knowledge to guide practitioners towards the solutions for their challenges that exist in literature, and identify the gap between research and practice by studying the challenges encountered by developers when logging their applications and discussing their problems in SO. We further discuss the approach of our SLR in Section 3.1, our qualitative

analysis of 149 SO questions in Section 3.2 and our quantitative analysis of 20,766 SO questions in Section 3.2.5.

3.1 Systematic Literature Review

The goal of our systematic literature review is to study the state-of-the-art on software logging. To do so, we follow the approach of Keele et al. (172) to perform our systematic literature review (SLR). Figure 5 summarizes our methodology which consists of four main steps, (1) apply a search query in six databases, (2) apply a set of inclusion and exclusion criteria to filter our irrelevant papers, (3) perform backward and forward snowballing, and (4) manually analyze the final set of 204 papers. In the following, we detail each of these steps.

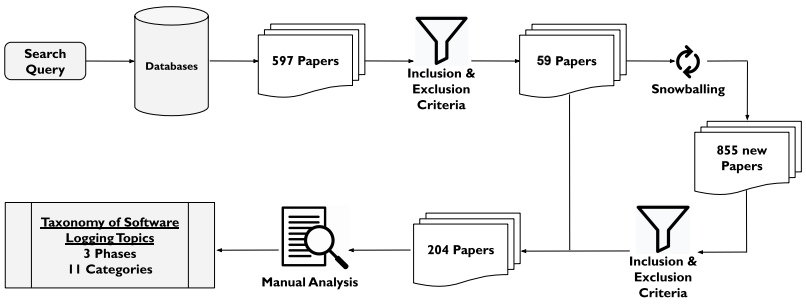


Fig. 5: Overview of our SLR methodology.

3.1.1 ***Search query:***

We use "Log*" as our main keyword to cover most papers that are related to software logging, then, as recommended by Staffs et al. (172) we searched for articles in six databases including Science Direct[6], Web of Science[7], IEEExplore[8], ACM Digital library[9], and INSPEC - El Compendex[10].

In particular, we apply the following search query, and then apply supplementary inclusion and exclusion criteria:

– **1)The title and abstract of the paper:** contains "Log*" AND NOT "Logi*". We used the keyword "Log*" to extract all the papers that are

[6] https://www.sciencedirect.com/
[7] https://www.webofscience.com/
[8] https://ieeexplore.ieee.org/
[9] https://dl.acm.org/
[10] https://www.engineeringvillage.com/

related to software logging. We excluded 'Logi*" from the query to avoid irrelevant papers such as these that are related to logistic, logic, etc.
- **2) The venue title:** contains "Software". We also specify that the venue title should contain the keyword "Software" to further reduce the possible noise of papers outside the scope of software engineering, especially as the log keyword can be a general term widely used in other fields. We are aware that we may miss relevant papers for which the publication title does not contain the keyword "Software" (e.g., papers from the Machine Learning and Knowledge Extraction journal). To mitigate such risk, we use forward and backward snowballing to identify relevant papers that we may have missed using our query.
- **3) Publication date:** between 2013-2023. Finally, we limit our scope to papers published in the last decade (i.e., between 2013 and 2023) to better manage our study. We believe that the literature we included in our study contains enough information to accurately represent the existing knowledge in the field, despite being limited to the span of ten years. We also decided to stick to the last ten years to cover the field of software logging at a large scope instead of covering just a specific sub-field (e.g., log compression, log parsing, etc.).

Our query is summarized as follows and translated into the queries shown in Table 2.

Final Query

The title and abstract of the paper contains "Log*" AND NOT "Logi*".
The venue title contains "Software".
The publication date is between 2013-2023.

Table 2: List of queries in each database. Note that IEEExplore and INSPEC - El Compendex require the years range to be filled manually. Additionally, the Science Direct and Web of Science databases do not generate a query so we manually fill the text field related to the Title and abstract with the keywords "Log*" NOT "Logi*" and the field related to the venue title with the keyword "Software"

Database	**Query**
ACM Digital Library	[Title: "log*"] AND NOT [Title: "logi*"] AND [Abstract: "log*"] AND NOT [Abstract: "logi*"] AND [Publication Title: software] AND [E-Publication Date: (01/01/2013 TO 12/31/2023)]
IEEExplore	("Abstract":"Log*" NOT "Abstract":"Logi*") AND ("Document Title":"Log*" NOT "Document Title":"Logi*") AND ("Publication Title":Software)
INSPEC - El Compendex	(((("Log*") WN KY) NOT (("Logi*") WN KY)) AND ((Software) WN ST))

Using such query, we were able to collect 597 unique papers. Next, we apply the inclusion and exclusion criteria on such papers to remove irrelevant ones.

3.1.2 ***Inclusion & exclusion criteria:***

Using the following inclusion and exclusion criteria we ended up with 59 papers. Specifically, the first and second authors separately applied the inclusion and exclusion criteria for 100 randomly selected papers, based on their title and abstract, then, they met to discuss their results and obtained an agreement score of 100% (Kripperndorff's $\boldsymbol{\alpha}$ (87)). As such process was straightforward, only the first author classified the remaining 497 papers.

Inclusion criteria:

- The paper is a journal or conference paper in English
- The paper empirically studies logging practices or introduces new approaches or frameworks to improve log management (i.e., instrumentation, storage and the fundamental components of log analysis)

Exclusion criteria:

- The paper is a preprint on arXiv,
- The paper is a secondary study (e.g., a review of other primary studies),
- The paper focuses on areas different from Software Engineering (e.g., medical patient monitoring systems),
- The paper leverages logging data only for other purposes such as debugging, detecting certain types of anomalies, analyzing the user/system behavior, analyzing the human resources' performance, etc. rather than improving the fundamental logging practices themselves

3.1.3 ***Snowballing:***

On the resulting 59 papers, we apply backward snowballing (i.e., identifying new papers from the reference list) and forward snowballing (i.e., identifying papers that are citing the paper being examined). As a result, we added 855 new unique papers, on which we applied the inclusion and exclusion criteria and we ended up with a final set of 204 papers.

3.1.4 ***Manual analysis:***

Next, we leverage a closed[11] and open[12] card sort analysis techniques to manually analyze the 204 selected papers and classify them into a high-level taxonomy. We leverage the closed card sort to classify each paper under one of the three phases of log management pipeline, i.e., instrumentation, storage and analysis. Then, we leverage the open card sort to classify them under different categories. In particular, we follow the next steps:

[11] Closed card sort is a categorization approach where participants are provided with predefined categories and asked to sort items into such categories.

[12] Open card sort is a categorization approach where the participants are free to add their own categories.

- **Step 1:** The first and the third authors manually and separately read the abstract, introduction, methodology, conclusion, and if necessary, the details of the research questions for each of the 204 selected papers, in order to get a better understanding of which phase of log management and which category does the paper belong to. Each author separately mapped each paper under one log management phase and category based on the goal of the paper, i.e., the problem it aims to solve. Since an open card sort technique is leveraged, raters are allowed to add new categories.
- **Step 2:** The first and the third authors met to discuss the labels they separately obtained at a high level, without discussing the details of the classification to avoid any classification bias. The authors then reviewed their labels before discussing the agreement and disagreements, as discussed in the following step.
- **Step 3:** In this step, we first cross validated the results to measure the agreement/disagreement score between the two raters. We obtained an agreement score of 89.22% (Kripperndorff's $\boldsymbol{\alpha}$ (87)). To solve the disagreement, they discussed the disagreement cases one by one to reach a final agreement score of 100%.

3.2 Qualitative and Quantitative analyses of logging questions in StackOverflow

While a large body of research focused on addressing a variety of logging challenges (e.g., log storage, automated log analysis, etc.), we want to understand logging issues and concerns from developers' perspectives. So, we create a mapping between the challenges and their existing solutions and identify the gap between the literature and practice. Therefore, we aim to better understand the common topics that are discussed by developers in the popular Q&A web site Stack Overflow. To do so, we collect questions related to logging and we use the Latent Dirichlet Allocation (LDA) model to extract a set of topics from the corpus of logging questions, to better guide our qualitative analysis (Section 3.2.4) and cover more topics instead of just investigating random samples of StackOverflow questions that might not cover all the topics. Figure 6 summarizes our approach.

3.2.1 ***Data collection:***

We follow a data collection strategy that is similar to the systematic literature review, as we collect all the questions in StackOverflow with the "Logging" tag starting from 1st January 2017, which sums up to a total of 20,766 questions. We only consider questions starting from the past six years because we observed that most questions prior to 2017 revolved around outdated techniques, technologies, and operating systems, such as inquiries about malfunctioning binary files or issues with obsolete operating systems like Windows

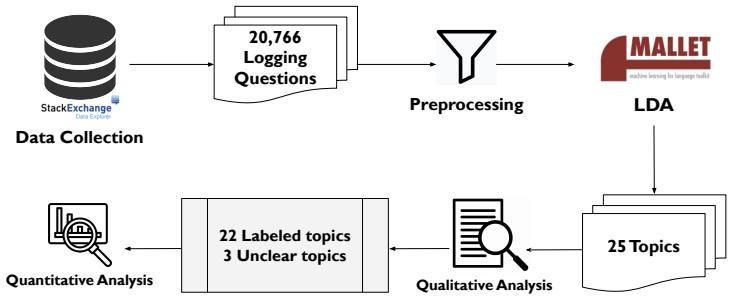


Fig. 6: Overview of our LDA methodology.

Vista. Therefore, we aim to focus our analysis on the more recent data to ensure that our findings are relevant and reflect the current industry challenges and advancements. In particular, we use the StackExchange Data Explorer[13] to query the "Question_ID", the "Question_Title" and the "Question_Body" of all the logging questions.

3.2.2 ***Preprocessing:***

We perform pre-processing techniques to remove noisy textual data from our collected questions. In particular, we lowercase the text, remove special characters, numbers and html tags (e.g., <p>, <b>, etc.), then, we remove stop words and apply stemming using the natural language toolkit (NLTK)[14] for Python. The resulting text is then introduced as input to MALLET[15], in order to extract a set of topics (K topics).

3.2.3 ***LDA:***

To identify our K topics, we need to choose a value of K that is small enough to avoid having multiple similar topics, but at the same time high enough to avoid losing details as a consequence of merging different topics together. To do so, similarly to prior work (57; 186), we experimented with a different number of topics K (50, 40, 30, 25, 20, 15, 10) and decided to choose K = 25 after manually analyzing the cohesiveness of our obtained topics. Then, LDA provides for each topic a list of related keywords and a list of questions that discuss that topic with a membership percentage. For example, the membership of topic #1 in the question with the id *55231609* [16] is 0.9048. Based on the generated keywords and some exemplary questions with high membership percentages (i.e., between five and ten questions) we manually assign a label to each topic.

[13] `https://data.stackexchange.com/stackoverflow/query/new`
[14] `https://www.nltk.org/`
[15] `https://mimno.github.io/Mallet/`
[16] `https://stackoverflow.com/questions/55231609`

3.2.4 ***Qualitative analysis:***

While LDA serves to identify the topics and classify different questions under these topics, our qualitative study aims to understand what developers ask and discuss around the obtained topics. To do so, we select five questions from each topic. The questions we select have the highest membership probability, i.e., the questions most likely discuss the said topic. As we were unable to understand seven topics from just five questions, we extended the number of questions up to ten. Our final set of questions studied is 149.

3.2.5 ***Quantitative analysis of the topics:***

The goal of our quantitative study is to compare different topics in terms of popularity (i.e., views, upvotes and downvotes) and difficulty (i.e., time taken to get an accepted answer). To do so, we first use the mapping between the 25 topics and their corresponding questions. In particular, we use the same threshold $\delta = 0.10$ defined by Barua et al. (9) to determine whether a question discusses a topic. Finally, we statistically compare the topics based on the aforementioned metrics by using the Scott-Knott clustering algorithm to regroup topics of the same level of difficulty and popularity, hence identify which set of topics are more difficult and/or popular.

4 Results

This section provides the results of our systematic review of the literature (SLR) and the topics obtained from the StackOverflow (SO) questions. In particular, our papers are classified under the three phases of the logging pipeline: Instrumentation (Section 4.1), Storage (Section 4.2) and Analysis (Section 4.3). The covered topics by the literature are summarized in Figure 7 and the SO topics in Table 8. Specifically, we observe that 37%, 10%, and 53% of the studied papers focus on log instrumentation, storage, and log analysis, respectively. Surprisingly, two out of the four studied SO high-level topics are not covered by the literature. The two high-level topics contain a total of 13 fine-grained topics that are mainly related to the configuration of logs and the logging of specific technologies and frameworks.

4.1 Instrumentation

A large body of research has empirically studied the practices, challenges, and benefits of logging. To help practitioners overcome the logging challenges, previous studies focused on guiding the logging decisions by recommending where to insert new log statements, which variables to log, which log verbosity level to use, when to review existing log statements, how to detect issues with the quality of log statements (e.g., anti-patterns in the logging code) and other issues related to securing logs (e.g., data leakage and information exposure).

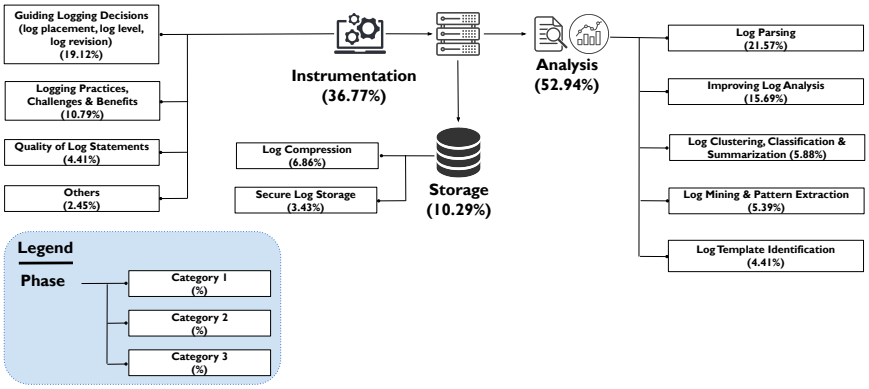


Fig. 7: Taxonomy of software logging topics in the existing literature.

State of the art:

4.1.1 ***Logging practices, benefits and challenges:***

Researchers investigated the logging practices, challenges and benefits discussed over the following five findings. In particular, the literature in this research direction can be summarised as a set of empirical studies that (**finding 1**) enumerate different benefits of logging, (**finding 2**) which motivate practitioners to use logs for different goals, yet not in a standard way. But in different ways that vary from one to another type of systems. (**finding 3**) However, using logs ends up with many challenges that are (**finding 4**) caused by the inconsistency of what developers aim for and what ends up being implemented from one side and (**finding 5**) from how practitioners maintain and evolve their logs from the other side.

Finding 1: Using logs come-up with several benefits such as system comprehension (e.g., understanding the major events at run-time), enabling tool monitoring (e.g., using external tools to integrate log-based alerts) and diagnosing certain run-time failures as reported by Li et al. (58). Furthermore, Rong et al. (45) observed that researchers from various fields show growing interest in logging practice.

Finding 2: The literature enumerated different goals and ways for using logs which are different among different types of systems. Developers add logs to report events, capture the state of the application and trace the run-time execution flow (37; 11). To do so, Fu et al. (133) identified five categories of logging practices in industrial systems: 1) logging assertion tests, 2) logging potential function return errors, 3) logging exceptions, 4) logging logic-branch points (e.g., *if* or *switch*), and 5) observing-point logging (i.e., all the other cases that do not belong to the four previous categories).

However, these practices can be different among different types of software systems and artifacts. For instance, Harty et al. (80) observed that mobile

analytics logs are less pervasive than the logging code of non-mobile applications, and more user-centered. That was also confirmed by Zeng et al. (210) as they found that the logging practices in mobile applications differ considerably from server and desktop applications. On the other hand, Zhang et al. (52) observed that test logging differs from production logging in terms of density, log levels and log messages and should be handled differently in future research. Li et al. (57) conducted a case study on six open-source systems to investigate whether certain topics of code snippets are more likely to be logged than others. They found that each of the studied systems shared 12% to 62% of its topics with other systems and that a small set of topics are often logged more than others. Interestingly, Zhi et al. (23) are the only ones who studied the practice of logging configurations in open-source and industrial software systems, and identified 10 findings about logging configurations related to the logging management, logging storage, logging formatting, and the quality of the log configurations. It is surprising that we did not identify other papers studying log configuration practices as it represents a real challenge for practitioners, and is one of the main reasons why developers tend to implement their own logging utilities (19). In fact, Chen et al. (19) found that 75,8% of the large-sized projects and 92.0% of the very-large-sized projects have developed their own logging utilities, mainly to avoid configuration issues related to log format, compatibility with other logging utilities, and the ease of configuration and dependency management. Finally, Lal et al. (144) performed a large-scale analysis of logging code constructs, to provide insights for future work and logging prediction tools. In particular, they studied logged and non-logged code constructs at both high-level (source code files) and low-level (catch-blocks) to identify relations between code constructs and logging characteristics.

Finding 3: While logging has several benefits, **the current logging practice is hindered by many challenges**, according to Rong et al. (45), and other studies (58; 18; 136; 155), among which deciding where and what to log (58), balancing between logging and system performance (58; 155),migrating from one logging library to another (177) and decreasing code readability (58). That is besides other online discussed topics, some of which (e.g., using specific logging tools and libraries such as Graylog [17] or logcat [18]) were not yet addressed by the research community according to Gujral et al. (53; 54; 55). Such challenges lead to inaccurate logs that make debugging errors more time-consuming when it is supposed to be the opposite (i.e., bug reports with logs are intuitively supposed to be easier to fix). Such a claim was confirmed by Chen B. et al. (18) who observed that bug reports containing log messages take a longer time to resolve than bug reports without log messages. Similarly, Chen A. et al. (136) observed in ten other open-source systems that logs frequently have missing system execution information. Besides, they also ob-

[17] Open-Source log capture tool and provides analysis solution for operational intelligence
[18] Command-line tool that dumps a log of system messages including messages written from an android application

served that users often do not attach accurate or sufficient logs, which extends the bug resolution time.

Finding 4: Among the empirically studied causes of such challenges is the inconsistency between the purpose of logging and what developers add to their code. Rong et al. (46; 47) observed, through a survey, that developers' logging intentions (e.g., to capture a performance anomaly or just plain errors) and concerns (e.g., balancing between the cost and benefit of logging) are poorly reflected in the code. They also found that the logging practice is highly inconsistent, in terms of the density and log levels, between different developers both across projects and even within one project. However, the log placement is consistent between different developers. Similarly, Zeng et al. (210) observed in mobile applications that 35% of the log levels are inconsistent with the developers' rationale (i.e., the reason for inserting the log statement, such as debugging, detecting anomalies, etc.), which can hinder the usefulness of logging or cause performance overhead.

Finding 5: Logging issues are also coming from the way developers maintain their logs. First, some studies looked at how practitioners maintain their logs. For instance, Patel et al. (85) observed that most changes in logging code in the Linux kernel are made to fix language issues, modify log levels, and upgrade logging code to use new logging libraries. Chen B. et al. (18) also observed that developers in Apache Software Foundation update logs mostly to enhance their quality (e.g., formatting, spelling/grammar fixes). However, such maintenance can come up with certain issues, among which is missing log revisions, as reported by Niu et al. (205). They found that more than half (54.14%) of log revisions belong to groups of similar log revisions, and 64.4% of such groups contain log revisions that are missed by developers. So they emphasized on the importance of understanding the patterns and similarities of log revisions to improve logging practices.

4.1.2 ***Guiding logging decisions (log placement, log level, log revision):***

Prior work aimed to address the logging challenges by guiding developers through the instrumentation process. In particular, by (**finding 1**) recommending which log verbosity level to use, (**finding 2**) where to log , (**finding 3**) what to log. Finally, after the creation of log statements, other studies developed approaches to suggest whether a log statement should be updated (**finding 4**). Table 3 summarizes the approaches used in each paper discussed bellow.

Finding 1: A large body of research focused on helping developers choose the appropriate log level for log statements. For instance, Kim et al. (179; 178) introduced an ML-based approach that can validate whether a log level is appropriate using semantic and syntactic features in the source code, and also recommend an alternative log level otherwise. Similarly, Anu et al. (49) proposed an ML-based approach to recommend a log level based on the logging intention, which is inferred from the following features in the source

code: called methods, log messages, exception types and code comments. Li et al. (59) used ordinal regression models to predict the most relevant level for a new log statement, based on five dimensions of metrics: logging statement metrics, containing block metrics, file metrics, change metrics and historical metrics. Using the same model as Li et al. (59), Ouatiti et al. (35) evaluated log-level prediction models in multi-component systems. They observed that using global models (trained at the whole project level) can be misleading, since local models (trained at the component level) statistically outperformed global models on a significant percentage of components, and because feature importance rankings were different among global and local models.

Other studies (225; 38) designed deep learning approaches that extract syntactic features of the logging statements and use them to recommend the appropriate log levels. Furthermore, Liu et al. (38) emphasized the importance of considering both intra-block (i.e., information about the source code block in which the log statements reside) and inter-block (i.e., information from surrounding blocks) syntactic features in suggesting appropriate log levels for software logs when leveraging a deep learning approach.

Instead of using syntactic features of logs, Tang et al. (211; 212) proposed an approach that can mine the history of code changes to determine developers' degree of interest [19] in the software source code surrounding the log statements. Then, they correlate such interests with log levels and suggest new log levels if mismatches between the interests and the log levels are discovered.

Mizouchi et al. (187) proposed an automated tool that dynamically adjusts the log level of a running system to record irregular events such as performance anomalies in detail (Trace level) while recording regular events concisely (User specified level). In fact, the tool is provided a log level (e.g., Error) for recording regular behavior in addition to a set of execution scenarios for learning regular behavior. Once an unknown execution scenario is detected, the tool switches to the trace level. A case study showed that the tool increased the log file size by just 34.1MB and the execution time by 30% compared to 1,431MB and 74% when the trace level is used constantly.

Finding 2: Besides recommending log levels, other papers focused on ensuring that logs are strategically placed in the source code by suggesting exact locations that need logging. For instance, Zhu et al. (73) introduced a ML-based approach which extracts structural, textual and syntactic features from existing log instances and feeds them to a classifier to determine the appropriate location for a log statement (e.g., an exception or a return value). Lal et al. (143; 145; 146; 147; 148) also presented three ML based approaches that can help developers optimize the number of log statements in two source code blocks: if-blocks and catch-blocks. Whereas Jia et al. (231) designed an approach to help developers place exception log statements, by mining placement rules from semantic features. The approach was 43% more accurate than an older state-of-the-art tool (34).

[19] Degree of interest model (DOI) was proposed by Kersten and Murphy (105) to measure the degree of developers' interests in program elements

Instead of recommending log statements in just one or two specific types of code blocks, Tschudin et al. (154) and Zhu et al. (75) proposed ML-based approaches that suggest fine-grained log locations (e.g., catch-blocks, if-blocks, switch-blocks, loops, methods) where to insert log statements based on the history of code changes, and based on textual and syntactic features, respectively. Similarly, Li et al. (228) proposed a deep learning approach for recommending log locations based on semantic and syntactic features. Zhao et al. (206; 207) developed an algorithm that balances the informativeness of the log placements and the performance overhead to recommend log locations. While the previous papers focused on proposing different approaches to recommend accurate log placements, Candido et al. (70) systematically studied the effectiveness of log placement models in an industrial setting. In particular, they investigated the effect of sampling techniques on the prediction results of log placement models, and found that sampling techniques improve recall but significantly deteriorate precision.

On the other hand, Rivera-Ortiz et al. (40; 39) proposed to help developers identify potential software misuse scenarios, expressed as annotated UML Sequence Diagrams, and use such diagrams to recommend different locations where log statements should be placed and the information that should be logged.

Finding 3: Beyond the log location, researchers explored the content of log statements. Specifically, they developed approaches to recommend which variables need to be logged and what should be the descriptive text of the log statement. Through an empirical study, He et al. (131) found three main categories of logging description: 1) description of the status of a program operation (e.g., ‘‘`Waiting for V1 to stop`’’), 2) description for an error message (e.g., ‘‘`Unable to get jobId`’’), and 3) semantic description of code variables, functions and branches (e.g., (‘‘`Missing `’’, `cId)`). Whereas, Liu et al. (201) suggested that generating descriptive texts in logging statements can be seen as a retrieval-based Q&A task which consists of searching the most suitable answer for a new query from existing Q&A knowledge. They conducted a systematic analysis of two retrieval algorithms (e.g., Information retrieval (IR)-based algorithms and neural networks (NNs)-based algorithms) and assessed their accuracy using metrics and human evaluation. They found that it is still challenging to apply the existing log text retrieval methods to cross-project scenarios because log texts differ greatly in different projects.

Ding et al. (232) and Gholamian et al. (171; 170) described the only existing ML-based approaches for generating logging descriptive texts by translating the textual and syntactic features of the source code into short textual descriptions. In addition to the descriptive text, Gholamian et al. (171; 170) also predicted where to log, and they found that leveraging code clone detection methods with natural language processing can increase log placement prediction accuracy.

Other studies (230; 160) introduced deep learning approaches that leverage semantic and syntactic features of the source code to recommend variables

to log. Mastropaolo et al. (10) also presented a deep learning approach that recommends where to inject a complete log statement with a human-readable logging message and the appropriate log level. While Jia et al. (184) argued that existing developer-written logs are designed for humans rather than machines to automatically detect system anomalies, they presented the first attempt to provide machine-written logs to improve automatic fault diagnosis. Specifically, the approach leverages a graph-based model to identify the minimum number of logging points that precisely reflect anomalies in the system. Finally, King et al. (69; 68) aimed to answer the question "to log or not to log?" by leveraging a set of heuristics (if-then rules based on expert knowledge) to help developers identify mandatory log events.

Finding 4: In parallel, prior work developed machine learning based tools to recommend log revisions. Kabinna et al. (176) found that 20 to 45% of the logging statements change during their lifetime. Li et al. (60) built classifiers that can make the "just-in-time" suggestions to help developers decide whether a log change (i.e., addition, deletion or modification of a log statement) is needed at commit time. Moreover, Li et al. (157; 158) introduced an automatic tool that analyzes the correlation between logging context (Logging context is generally made up of two main components: branch statements that indicate under what conditions the log message should be printed, and log statements that describe what variables to output) and log modifications to establish a set of log revision rules. These rules are then used to recommend candidate log revisions.

Table 3: Summary of the approaches used to guide logging decisions.

Objective	Family of techniques	Leveraged data	Description	Papers
Log level prediction	Machine learning	Semantic	Semantic features represent the words in a log message. Each log level has a set of most representative words	(179; 178; 59; 49; 35)
		Syntactic	Features characterizing the surrounding code of a log statement	(179; 178; 59; 49; 35)
		Change metrics	Information about the actual code changes associated with the newly-added logging statement	(59)
		History metrics	Code changes in the containing file in the development history	(59; 154)
	Deep learning	Syntactic	Features characterizing the surrounding code of a log statement	(225; 38)
	Correlation	History metrics	Code changes in the containing file in the development history	(211; 212)
Log placement prediction	Machine learning	Structural	Semantic meanings of error types and associated method names	(73; 143; 145; 146; 147; 148)
		Textual	Includes all the text in the code snippet, such as variables and types	(73; 143; 145; 146; 147; 148; 75)
		Syntactic	Features characterizing the surrounding code of a log statement	(73; 143; 145; 146; 147; 148; 75)
		History metrics	Code changes in the containing file in the development history	(59; 154)

	Deep learning	Semantic	Represent the words in a code snippets, excluding the reserved words of the programming language	(228)
		Syntactic	Features characterizing the surrounding code of a log statement	(228)
	Data mining	Semantic	Represent the words in a code snippet	(231)
Log variables prediction	Machine learning	Textual	Includes the messages of the existing logging statements	(232; 171; 170)
		Syntactic	Features characterizing the surrounding code of a log statement	(232; 171; 170)
	Deep learning	Semantic	Represent the words in a code snippet	(230; 160; 10)
		Syntactic	Features characterizing the surrounding code of a log statement	(230; 160; 10)
Log revision prediction	Machine learning	History metrics	Code changes in the containing file in the development history	(60; 157)

4.1.3 ***Quality of log statements:***

Prior work focused on (**finding 1**) analyzing log statements to mitigate logging antipatterns (i.e., issues which can hinder the understanding and maintainability of logs), (**finding 2**) whereas other studies focused on logging related vulnerabilities and worked toward efficient log data protection.

Finding 1: Previous studies developed static code analysis tools to detect anti-patterns in the logging code, problematic duplicate logging code smells and other log-related issues. For instance, Chen et al. (17) identified and developed a static code analysis tool for five different logging anti-patterns: 1) Nullable log variable (i.e., objects that can be null may cause a NullPointerException) , 2) Explicit cast of a log variable (i.e., explicit casting of an object into a particular type may cause runtime type conversion errors and system crash), 3) Wrong verbosity level (i.e., using wrong log levels may cause logging overhead and large volumes of redundant logs), 4) Logging code smells, and 5) Malformed output (i.e., when an object does not have a human readable format such as a byte array). Additionally, Li et al. (226; 227) designed DLFinder, an automated static analysis tool to detect problematic duplicate logging code smells. Hassani et al. (103) developed an automated tool that detects four types of log-related issues: 1) Typos, 2) Missed exception message, 3) Incorrect log levels, and 4) Log level guards (i.e., when developers forget to add necessary if statements before logs, to reduce overhead). Finally, Chen et al. (16) studied logging related changes to enumerate logging related issues and interestingly found that a small fraction (3%) of the logging issues can be identified with code issue detection tools.

Finding 2: Prior work studied the security aspects of logs by investigating the causes of data leakage and information exposure caused by logging, and worked towards efficient log data protection. For instance Zhi et al. (22) investigated the root causes of information exposure through logging, and how exploitable are these vulnerabilities. They found that 1) 67.8% of the vulnerabilities can be exploited via the network,

and 89.3% of vulnerabilities can be exploited with low efforts; 2) malicious users can use 46.9% proof-of-concept exploits [20] to launch attacks without any expertise; 3) among the common root causes for such vulnerabilities is storing log files publicly accessible directories and improper implementation of sanitization (i.e., handling of sensitive data). Zhou et al. (139) proposed an approach called MobiLogLeak that identifies log statements in deployed Android applications that leak sensitive data, indicating the potential risks associated with poor logging practices in mobile applications. Interestingly, Sun et al. (77) proposed a real-time black-box attack framework called LogBug that can evade the anomaly detection by slightly modifying the logs. The authors evaluated LogBug on five emerging log parsers and observed that it can greatly reduce the accuracy of log parsers with minor perturbations in real time.

4.1.4 ***Others:***

We analyzed six papers on log instrumentation which did not belong to any of the previous categories.

For instance, prior studies (182; 221) proposed logging systems that aim at reducing the volume of logged data to improve the subsequent analysis tasks and optimize the logging overhead using different filtering criteria (i.e., using filtering criteria defined by administrators, the objective is to selectively reduce the logged data). Whereas Yang et al. (173) introduced a faster logging system with higher throughput (80 million log messages per second) and lower latency (18 nanoseconds) compared to existing logging systems using log compression and decompression.

In an industrial context, Bosch et al. (114) studied the various challenges of the approaches used to generate, store and manage the evolution of log data, and introduced an agile approach to overcome the identified challenges, which was validated based on expert interviews.

Finally, Zhi et al. (24) focused on studying the practice of logging guards [21] and introduced an accurate algorithm to detect missing logging guards in the source code.

Topics discussed in StackOverflow:

While only Zhi et al. (23) studied the practice of logging configurations in software systems, we observed that developers face numerous log configuration challenges according to topics #1, #6, #12, #10 #4, #16, #23, #13, #8 and #24 in Table 8. Such challenges may concern the dependency configuration of logging libraries. In particular, developers can misconfigure Maven to import logging dependencies, or make other configuration errors related to importing the dependency of Winston,

[20] PoC: a sample code that can be used to exploit a specific vulnerability, usually created by security researchers or ethical hackers to illustrate how a vulnerability can be exploited and to demonstrate the impact of such an exploit.

[21] Log statements can be guarded by conditional statements, known as logging guards, to ensure they are only executed when necessary (24)

resulting in runtime errors and the inability to log. For instance, an incorrect scope for the log4j dependency in the configuration file of Maven can lead to runtime issues when running the application. Additionally, the dependencies used by practitioners can leverage different logging frameworks which may cause conflicts and generate incorrect logs or logs that do not respect the specified configuration. This became a significant concern after the Log4J-related security vulnerability. Despite practitioners attempting to upgrade the relevant dependency, the issue persisted as the dependency was still present due to its origin from another dependent component. We also observe that practitioners tend to get various configuration errors when configuring their logs such as a log file not created or an empty log file, especially when they try to configure multiple logging files at the same time. Furthermore, the log configuration topic also includes issues related to infrastructure. For example, developers may find that the 3rd party libraries they use generate verbose logs and therefore ask questions on how to configure them in order to have a lower verbosity level. Users also face various challenges when configuring logs in Laravel. For instance, a user may be unable to write log files due to wrong permissions in the log location. Other users can find it difficult to log to different files in Laravel. Another issue we observe is when the logs of Laravel are too verbose they hide relevant information. Moreover, we observe two topics related to log configuration where developers try to configure multiple log outputs, or send/collect logs from different applications. For instance, Developers aim to separate different types of logs in different outputs but face challenges related to the configuration. For instance, they may face issues redirecting some logs to a specific appender and other logs to another different appender. Other users may want to save logs to different file for different frameworks. Developers also seek help to configure logs to be sent or collected from different applications such as Elasticsearch, Fluentd, CloudWatch and Kubernetes. Some of the issues they may face is when they try to send different logs to two or more different applications at the same time, but they see that all logs are being pushed to both applications. In other cases, the configuration does not send logs to the target application. Surprisingly, in some cases, developers ask questions about how to completely disable the logging of an application but the framework they use don't have such an option. For example, a developer stated that they would like to "turn off this logging under Javalin: [...] but Javalin does not seem to have a way to turn these logs off. Is there a way?" [22]. One notable finding is the wide range of technologies that developers encounter errors with when it comes to log configuration. From Maven to Winston applications and various other technologies (e.g., Laravel framework), log configuration can be a challenge regardless of the libraries and frameworks that are being used.

We also observed various logging related questions where practitioners seek help on the usage of logging libraries in the code. For instance, practitioners may not get the right log levels printed due to a mis-

[22] `https://stackoverflow.com/questions/57959261`

usage of the logging API (e.g., logging API in Python), and such a misusage can be coming from the documentation. In fact, a user copied a logging code from Python documentation which is supposed to print all the INFO logs, but the user was "only able to see the warn and above" [23]. On the other hand, developers ask questions about the appropriate way to use logging a library in their code, or ask for help as they face numerous challenges related to the usage of logs in .Net and Java projects. For example, an owner of a .Net core project followed .Net documentation to add logging to their application, but kept getting an exception related to the logger object [24]. Practitioners can also face challenges when trying to log a subprocess and therefore seek help on SO. For instance, some users may ask for how to handle subprocess logs, while others may want to know how to customize the way a subprocess outputs logs.

We identified four topics related to scattered logging where developers aim to write their logs into multiple locations, applications or databases, or across multiple processes. Two of these topics are closely related to log configurations, i.e., topics #8 and #24, and revolve around the configuration of logs to be separated into different outputs and the configuration of logs to be sent and collected from different applications, respectively. Moreover, Developers can face difficulties logging accross multiple concurrent processes or logging particular files in multiple process environments. In some cases, practitioners try a solution that does not work, and in other cases the solution does not work incorrectly. For instance, when running two subprocesses, a user can observe two logs created in each subprocess instead of one log in each subprocess [25]. On the other hand, we observed four cases where practitioners ask questions on the logging of databases. For example, practitioners may try to log two or more tables in a database but their logger object only logs one table [26]. Others simply don't know how to log data entries or updates in a database [27].

While a large body of literature addressed the questions "what to log? Where to log? And which log level to use?" No prior work tried to adapt such studies to specific contexts such as project migrations. This brings us to topic #17 where developers ask what variables need to be logged or why certain irrelevant information are being logged (e.g., IP address), while others ask question on how to get all the relevant information for a given context such as how to extensively log an application to ensure a secure migration of a project. For example, a developer who is trying to migrate legacy code posted a question on SO [28] asking for help on how to "track everything that is happening (or most of it) from an user's point of view" in order to have all the necessary details on what a user did when something mysterious happens, and to be speed up issues resolution.

[23] `https://stackoverflow.com/questions/43109355`
[24] `https://stackoverflow.com/questions/49973991`
[25] `https://stackoverflow.com/questions/53057510`
[26] `https://stackoverflow.com/questions/74447130`
[27] `https://stackoverflow.com/questions/67853661`
[28] `44921694`

Summary of Log Instrumentation

Regardless of the used logging library, log configuration is the most challenging task for practitioners. However, only Zhi et al. (23) studied the practices of logging configuration in software systems. While the literature focused on studying logging practices, challenges and benefits, in addition to recommending the appropriate log level and placement, practitioners are more concerned about the usage of logging libraries in the code, scattered logging and the context- dependent usage of logs.

4.2 Storage

Nowadays, software systems generate huge amounts of log data everyday, which consumes enormous amounts of storage space. Therefore, efficient and secure log storage techniques are necessary to reduce the storage costs and protect sensitive information such as user credentials from tampering.

State of the art:

4.2.1 ***Log Compression:***

A large body of research dedicated effort to (**finding 1**) designing customized compression techniques for logs to improve the compression ratios and facilitate the efficient retrieval of log data, and (**finding 2**) compared the existing compressors.

Finding 1: Researchers designed efficient log compression techniques based on log parsing, log template identification and other novel approaches. First, Liu et al. (78) created Logzip which uses fast iterative clustering [29] to extract log templates from raw logs. Then, it generates intermediate representations and provides them to a traditional compressor for a more effective compression. Then, Wei et al. (82; 81) designed LogReducer and LogGrep, two log compression techniques based on log parsing which achieve higher compression ratio compared to Logzip (78). Tian et al. (137) proposed LogDAC, a parser-based log compression method that converts logs to templates, headers (e.g., timestamp, log level, source) and parameters (i.e., log variables) to make log data compatible with general compressors that use the dictionary data structure. Likewise, Marjai et al. (125) used a dictionary-based method to store the log template and the parameter list. To extract the template of a log message, the authors use six log template identification algorithms, and compare the compression rate of their method using each algorithm. Later on, Marjai et al. (126) enhanced their algorithm using Huffman coding, which achieved a 94.98% compression rate, compared to the 67.4% compression rate of their previous algorithm.

[29] unsupervised data clustering algorithm that operates by iteratively refining the clustering results

Other studies (50; 15; 86; 89; 122; 48) designed customized compression techniques without the use of log parsing or log template identification. For example, Lin et al. (50) proposed a new approach named "column-wise compression" where each log message can be independently compressed or decompressed for analysis by separating the message into several columns and compressing each column with different models. Feng et al. (15) and Rodrigues et al. (86) presented two log compression techniques that leverage the redundancy in log data to improve the compression ratio. Whereas, Yao et al. (89) designed LogBlock, a novel approach to preprocess small log blocks before compressing them with a general compressor. The results of the evaluation on 16 log files showed that LogBlock can reach both a higher compression ratio and a faster compression speed than Logzip (78) and the "column-wise compression" (50). Finally, Fei et al. (122) proposed SEAL, a novel data compression approach that achieves "lossless" compression (i.e., all the information in logs can be retrieved) as opposed to "lossy" compression which removes logs matching pre-defined patterns, leading to unavoidable information loss (48).

Finding 2: Researchers also focused on evaluating the performance of existing techniques and tools for log compression. For instance, Yao et al. (88) evaluated the performance of general compressors (e.g., the LZ77 used in gzip) to compress log data. They found that log data can be compressed and decompressed faster than natural language data, with higher compression ratios. They also observed that configuring compression levels to be higher than the default level can help achieve the optimal compression performance. Alternatively, Spillner et al. (79) presented a systematic framework to choose between 30 compression tools and configurations that can exploit cost tradeoffs, such as investing in better compression levels while saving long-term storage costs. The authors showed that there is no one-size-fits-all solution for log compression, and the best compression tool and configuration depend on the specific use case and requirements.

4.2.2 ***Secure Log Storage:***

Finding1: Prior studies focused on securing the log storage using immutable log storage and log pseudonymization. For instance, Wang et al. (51), Pourmajidi et al. (198) and Aslan et al. (188) developed three blockchain-based systems [30] for log storage. While Pourmajidi et al. (198) focused on leveraging the immutability property of blockchain for log storage, to avoid the risk of tampering with log data (i.e., tampering includes adding, removing, and manipulating a log partially or entirely), Wang et al. (51) focused on improving the high latency and low throughput of blockchain-based systems, whereas Aslan et al. (188) developed a data structure that allows only personal data to be deleted without destroying log data integrity

[30] blockchain is tamper-proof and decentralized which allows enterprises to execute business processes with privacy and security

in accordance with the General Data Protection Regulation [31] On the other hand, Varanda et al. (13; 14) evaluated the effectiveness of log pseudonymization [32] strategies in different log processing phases: 1) log generation, 2) log collection, 3) log indexation (i.e., structuring log data) and 4) log presentation (i.e., displaying or visualizing log data in a user-friendly and understandable format). They found that log pseudonymization in the log collection phase significantly improves the querying performances (i.e., searching for specific information or data from the log records) and implemented an approach for log pseudonymization using ELK [33]. Portillo-Dominguez et al. (118) focused on addressing the challenge of protecting sensitive data contained within logs generated by IT infrastructures of companies. They proposed an automatic approach, namely SafeLog, to filter out information and anonymize log streams to safeguard the confidentiality of sensitive data and prevent its exposure and misuse from third parties. Finally, Zawoad et al. (156) introduced SecLaas (Secure-Logging-as-a-Service), a novel approach which stores virtual machines' logs and provides access to investigators (i.e., developers responsible for investigating logs to understand the behavior of the system) while ensuring the confidentiality of cloud users.

Topics discussed in StackOverflow:

While prior work focused on secure log storage and log compression, practitioners face challenges when handling log files. In fact, practitioners ask questions about how to perfrom log rotation, that is, how to start storing log data to a new log file once a certain criteria is met (e.g., once a limit file size is reached) or how to store the upcoming log data in a way to avoid files overwriting each other. For example, a user on SO has a process that generates log files daily to their home folder [34]. The issue is that "Everyday it overrides the existing file in the destination folder". Therefore, the user asked for help to preserve the log from previous dates. The user was then advised to use a wrapper around each job but no accepted answer has been provided even after 6 years.

Summary of Log Storage

While a large body of literature focused on improving and securing log storage by developing novel log compression techniques and using log pseudonymization and immutable log storage, we observe that practitioners face challenges when handling log files, such as how to perform log rotations to preserve log data from overwriting.

[31] GDPR: a data protection and privacy regulation implemented by the European Union (EU)

[32] Data protection technique that involves replacing or encrypting personal data with pseudonyms or pseudonym identifiers

[33] Elasticsearch, Logstash and Kibana

[34] `https://stackoverflow.com/questions/41755437`

4.3 Analysis

With the sheer volume of log data, techniques for automated log analysis become crucial for understanding the behavior and performance of complex and large-scale software systems. Such techniques include pre-processing tasks such as log parsing, log template identification, log mining and pattern extraction, log clustering and summarization, as well as the process of deriving critical information about the system's performance and knowledge about users' behaviour (see Figure 8). In this paper, we explore research that focused on improving these fundamental logging-practices of log analysis. The use of logs or techniques for debugging specific anomalies is outside the scope of this paper.

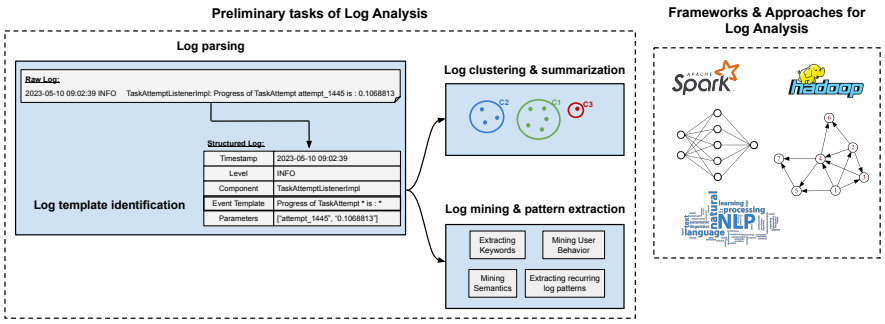


Fig. 8: An overview of log analysis techniques. These are the preprocessing techniques of log files that can lead to an easier use of logs for troubleshooting issues.

State of the art:

4.3.1 ***Log Template Identification:***

Log template identification involves grouping similar log messages into templates based on their structure and content. It is a challenging task and yet important for log analysis. For example, log messages belonging to the same template usually contain information on a common system behavior. By establishing a baseline behavior, anomalies can be detected by comparing new log templates against the established templates. Therefore, (**finding 1**) researchers have developed various approaches to automate and improve the accuracy of log template identification. Table 4 summarizes the approaches used in our analyzed papers which we describe in the following finding.

Finding 1: Prior work (150; 151; 142; 21; 140; 1) proposed several approaches for log template identification, including Natural Language Processing, search-based approaches, regular expressions and log clustering. Such techniques have shown promising results in improving the accuracy and efficiency of log template identification.

For instance, Kobayashi et al. (150; 151) proposed an NLP-based algorithm to generate accurate log templates from the textual data of log events, then designed a log analysis framework, namely Amulog, that further reduces 75% processing time for template generation compared to their initial approach (150), by using a search tree based algorithm for log template identification. One approach using incremental learning was proposed by Mizutani et al. (99) who aimed to mine log templates and refine them continuously in real-time based on incoming log messages. Serasinghe et al. (141) presented iLSE, a web-based solution for extracting templates from log files using regular expressions and unsupervised clustering of log data. Messaoudi et al. (142) and Tak et al. (21) presented two search-based approaches to identify log templates using the NSGA-II algorithm [35], and the Inductive Logic Programming (ILP) framework [36], respectively. Whereas Yang et al. (140) developed log a template identification method that vectorizes log data using the word2vec technique and further applied a clustering algorithm to generate log templates in real-time. Alternatively, Jayathilake et al. (132) introduced a method that uses Regular Expressions to convert the content unstructured log files into an abstract template, which can directly be converted to the format required by an existing automated log analysis framework (33).

Finally, Khan et al. (1) provided three guidelines for assessing the accuracy of log template identification techniques: 1) using appropriate accuracy metrics, 2) validating the manually determined templates that are used in the evaluation of the technique, and 3) performing the analysis of incorrect templates to understand why a technique may incorrectly identify some templates.

Table 4: Summary of the approaches used log template identification.

Objective	Family of techniques	Leveraged data	Description	Papers
Log template identification	Natural Language Processing	Textual	Refers to the text in the event logs, which contains information about the activities recorded by a software system. Each event is described by a single line in the log files	(150; 151)
	Machine learning			(99; 141)
	Search-based			(142; 21)
	Vectorization			(140)

4.3.2 ***Log Parsing:***

Log parsing involves transforming unstructured log data into structured format. The main difference between log parsing and log template identification is that the latter aims to group and categorize logs into templates based on their structure while the former consists of separating the static and dynamic parts of log messages, in addition to identifying different elements of the static part of a log event (e.g., timestamp, log level, log variables, etc.), as shown in Figure 8. In other words, log template identification is a sub-task of

[35] NSGA-II: Non-dominated Sorting Genetic Algorithm, a well-known and efficient technique to solve problems with multiple objectives

[36] ILP is a subfield of AI which uses logic programming to induce logical rules from sets of examples.

log parsing. Such techniques serve as a first step towards automated log analysis, as most of the existing log analysis methods require structured input data to perform tasks like diagnosing problems and detecting security threats in complex modern systems. A large body of research introduced both (**finding 1**) offline (i.e., parsing historic log files that have already been generated and stored) and (**finding 2**) online approaches (i.e., parsing log files in real time as they are generated) for log parsing and (**finding 3**) compared the existing ones. Table 5 summarizes the approaches of each paper which we further discuss in the following findings.

Finding 1: Prior work introduced several automated offline log parsers based on different techniques, mainly machine learning, deep learning and natural language processing. For instance, Meng et al. (194), Rucker et al. (111) and Chu et al. (44) developed three adaptive log parsers that can learn from existing log datasets and adapt to changes in log files to learn new log templates without additional manual work (e.g., human supervision). Pi et al. (4) and Abbasli et al. (115) proposed two NLP-based log parsers to extract key information from unstructured log messages, and annotate them using custom tags (e.g., "Time", "Level", "Content", etc.). Previous studies (117; 149; 5) also introduced ML-based log parsers which aim to extract a structured representation of logs by incorporating the textual features of logs (117; 149; 5).

Additionally, we observed various approaches for automated log parsing based on deep learning (67; 41), tree structures (12; 28) and other novel algorithms such as a log parser that uses features of log messages (e.g., length of a log message, tokens in log message) to identify log templates (209; 224) On the other hand, Fang et al. (94) and Tovarnak et al. (29) leveraged regular expressions (RegEx) to facilitate the log parsing (by replacing the log variables with wildcard ("*") and creating log templates). However, the use of RegEx requires manual efforts to define such regular expressions which is not efficient. Therefore, Raynal et al. (101) designed a novel clustering algorithm to infer the regular expressions required to parse log data.

Interestingly, rather than using log mining and pattern extraction as a subsequent step for log parsing, recent studies proposed log parsers (64; 217; 219) that leverage log mining and pattern extraction to convert raw log data into structured data. Similarly, Zhang et al. (91) and Guo et al. (169) proposed two approaches based on log clustering to parse structured log templates from raw or semi-structured log records.

Finding 2: A large body of research focused on providing online log parsers that can analyze log files in run-time. For example, Du et al. (107; 108) and Agrawal et al. (6) proposed Spell and Logan, respectively, two online log parsers that leverage a Longest Common Subsequence (LCS [37]) based approach to extract log patterns from log messages, and automatically discovers semantic meanings for parameter fields (e.g., in the template "Tem-

[37] In the context of natural language processing or text analysis, the LCS approach is often employed to measure the similarity or dissimilarity between two texts or strings

perature * exceeds warning threshold", the semantic meaning of "*" is *temperature*). Another significant contribution to this domain is Drain (130), an online log parsing method which can parse log messages and update its parsing rules during runtime. In fact, after preprocessing a new raw log message, Drain searches for a suitable log group (i.e., group of logs with similar templates) based on the encoded parsing rules. If a matching log group is found, the log message is associated with that group. Otherwise, a new log group is created and the parsing rules are updated accordingly. Marlaithong et al. (183) further enhanced Drain (130) by leveraging the Artificial Bee Colony (ABC) algorithm [38] to automatically tune Drain's parameters, and avoid the manual parameter setting by developers. Fu et al. (213) also worked on improving Drain (130) by introducing two components: 1) the first component generates separators (e.g., ", | ; [ ] ( ) -") automatically for log message splitting instead of using a uniform separator (i.e., blank) which can reduce the effectiveness of the log parser, while 2) the second component merges the candidate event templates by a template similarity method.

Other studies worked on improving the accuracy and efficiency of Spell (107; 108), Logan (6) and Drain (130) by proposing multiple online log parsers which leverage the textual data of logs using different techniques like partitioning (229) (i.e., to avoid unnecessary comparisons between logs and existing log templates, log messages are grouped into different partitions based on the message length, then, a log template is extracted from each partition.), vectorization (185) (i.e., log messages are converted to vectors, then, a log template match is identified based on the distance between the log message vector and the vectors of log templates that were identified from previous logs.), sequence labeling (167) (i.e., labeling each word in a log message as either a template or a variable to identify the structure of the logs), machine learning (90), multi-layer structures (218) (i.e., in each layer different preprocessing and comparison techniques are employed), tree-based (123; 42) and dictionary-based (159; 163) approaches to parse log messages into structured formats and improve the parsing efficiency.

Finding 3: While most state-of-the-art log parsers demonstrated high accuracy, choosing the right log parser can be challenging. In fact, using the right log parser can significantly improve the performance of subsequent log mining tasks (128).

To address this challenge, prior studies (128; 129; 74; 208; 135; 213) characterized the state-of-the-art log parsers according to their performances. Additionally, He et al. (128) and Zhu et al. (74) presented a toolkit and a benchmarking to allow the reuse of the studied log parsers. Finally, Fu et al. (214) performed a comprehensive empirical study to investigate the impact of six state-of-the-art log parsers (130; 107; 61; 3; 104) on six state-of-the-art log-based anomaly detection methods (191; 134; 124; 106; 109; 195). Interestingly, they found that high parsing accuracy does not necessarily imply high anomaly

[38] Algorithm inspired by the behavior of honey bees, that is used to solve optimization problems

detection performance. Whereas increasing the number of parsed event templates decreases the efficiency of anomaly detection. The authors recommended the heuristic-based parsers such as Drain (130) (i.e., approaches with encoded parsing rules which are defined by experts) as they have less impact on the efficiency of anomaly detection methods, and all the anomaly detection methods perform more efficiently with such parsers.

Table 5: Summary of the approaches used in log parsing.

Objective	Family of techniques	Leveraged data	Description	Papers
Offline log parsing	Machine learning	Textual	Refers to the text in the event logs, which contains information about the activities recorded by a software system. Each event is described by a single line in the log files	(194; 117; 149; 5)
	Deep learning	Textual	Refers to the text in the event logs, which contains information about the activities recorded by a software system. Each event is described by a single line in the log files	(111; 67; 41)
	Graph structure			(44)
	Tree structure			(12; 28)
	NLP			(115; 4)
	Regular expressions			(94; 29)
	Log mining and pattern extraction			(64; 217; 219)
	Log clustering			(91; 169)
Online log parsing	NLP	Semantic	Refers to the semantic meanings of the variables in the event logs. Each event is described by a single line in the log files	(107; 108; 6)
	Dictionary structure	Textual	Refers to the text in the event logs, which contains information about the activities recorded by a software system. Each event is described by a single line in the log files	(159; 163)
	Tree structure			(123; 42; 130; 213)
	Artificial Bee Colony			(183)
	Machine learning			(90)

4.3.3 ***Log Clustering, Classification & Summarization:***

Log clustering, classification and summarization are techniques used to efficiently group similar logs and distill the essential information into a concise and summarized form that can be easily analyzed by practionners. For instance, by grouping logs with similar structures together, log clustering (unsupervised) and log classification (supervised) can help practitioners detect anomalies that deviate from the established clusters. Once an anomaly is detected, operators still have to inspect the raw logs to gain a summarized view before taking actions (192). Thus, researchers focused on (**finding 1**) developing log clustering techniques (techniques to group similar log lines into a cluster) and surveying the existing ones, as well as improving the log classification process. Additionally, researchers aimed on (**finding 2**) providing log summarization approaches (see Figure 9) that leverage ML techniques and existing log clustering techniques. Table 6 describes the approaches used in each paper which we further explain in the following findings.

Finding 1: To help address the challenges of analyzing huge amounts of log data, prior work designed novel log clustering approaches to

efficiently group similar logs, and provided surveys to help practitioners choose the appropriate approach for their requirements. For instance, Vaarandi et al. (138) proposed LogCluster, an algorithm that discovers both frequent events and outlier events (which may represent anomalous events) from the textual data in event logs, and clusters log data based on the discovered patterns. Similarly, Raffety et al. (65) proposed a method of clustering log statements from separate log files based on the similarity of the variables of log statements, to help future log analysis in distributed systems. Landauer et al. (100) and Egersdoerfer et al. (25) introduced two ML-based approaches to cluster logs based on the semantic similarity (i.e., measured as the distance between the vector representations) of the textual data in log files. Furthermore, Landauer et al. (100) presented a cluster evolution technique that generates clusters over different time windows to help detect anomalies related to periodic changes in log lines. Interestingly, He et al. (165) and Rosenberg, et al. (98) introduced clustering-based approaches to help identify problematic clusters of logs in cloud and network systems, respectively.

On the other hand, Shehu et al. (222) and Prayurahong et al. (121) aimed to improve the process of log classification, which refers to assigning predefined labels from different categories to log messages, using NLP and LDA, respectively. Dusane et al. (120) also leveraged NLP besides machine learning to design a tool that classifies log messages with negative sentiment, which were considered as events of interest within the system.

Finding 2: Other researchers introduced log summarization approaches that leverage ML techniques and existing clustering approaches to convert logs into a concise form that can be easily analyzed by practionners. For example, Dai et al. (61) and Locke et al. (174) designed "Logram" and "LogAssist", respectively, which leverage log parsing and n-gram models [39] to convert raw logs into a concise representation, by eliminating redundancies which may mask real problems in the logs or introduce additional challenges in log analysis. Meng et al. (192; 193) developed LogSummary, an automated approach for log summarization which keeps only the most important semantic information in the log summmary based on the TextRank [40] algorithm. Whereas Bunker et al. (83) proposed a clustering strategy that runs on subsets of the log data to effectively summarize the data without impacting the performance of ML models that may use such data.

Table 6: Summary of the approaches used in log clustering & summarization.

Objective	Family of techniques	Leveraged data	Description	Papers

[39] N-gram models are statistical language models used to analyze the patterns and relationships within textual data

[40] Graph-based ranking algorithm used for automated text summarization and keyword extraction

Log clustering	Pattern mining	Textual	Refers to the text in the event logs, which contains information about the activities recorded by a software system. Each event is described by a single line in the log files.	(138; 65)
	Machine learning			(100; 25)
Log summarization	Machine learning	Textual		(61; 174)
		Semantic	Refers to the words in the event logs and their meanings. Each event is described by a single line in the log files.	(192; 193)
	Log clustering	Textual	Refers to the text in the event logs, which contains information about the activities recorded by a software system. Each event is described by a single line in the log files.	(83)

4.3.4 ***Log Mining & Pattern Extraction:***

Log mining and pattern extraction involve the analysis of large and complex log data to extract meaningful information or recurring patterns on a system's behavior. A pattern is when a sequence of log events happen following a specific order. For instance, we can consider the following events in Hadoop Distributed File System (HDFS): (E1) adding a datanode, (E2) adding a new storage ID, and (E3) registering a datanode. The sequence E1-E2-E3 is a pattern which represents the behavior of adding a new datanode in HDFS. Log mining and pattern extraction are two techniques that can help with several log-based maintenance tasks such as anomaly detection, behavior analysis, failure prediction and diagnosis, root cause analysis, etc. So, (**finding 1**) many studies focused on providing novel approaches for log mining and pattern extraction. Table 7 describes the approaches used in each paper which we discuss in the following finding.

Finding 1: Prior work proposed to improve log mining and pattern extraction by leveraging machine learning, deep learning, log parsing and other techniques. For example, Huo et al. (216) designed LogVm, a ML-based approach for mining semantics from log messages, e.g., the variable semantics of the log message ‘‘`Listing instances in cell 949e1227`’’ are `(cell, 949e1227)`. Whereas Li et al. (199) and Duan et al. (202) proposed two deep learning neural networks to extract information such as the timestamp, the log level, as well as keywords from the log message that can be used for other log analysis tasks (e.g., anomaly detection, fault prediction, etc.). Text mining was leveraged by Kubacki et al. (97) and Obrębski et al. (27) to extract features from log events which can be used to characterize the operation of the application.

To improve the speed of the log mining process, Li et al. (220) used Spark for parallel processing and Elasticsearch to make such logs searchable, while Hamooni et al. (62) used a MapReduce [41] framework for distrubuted platforms to process millions of log messages and generate log patterns with 500x speedup

[41] Programming model and framework designed to process and analyze large volumes of data in a parallel and distributed manner

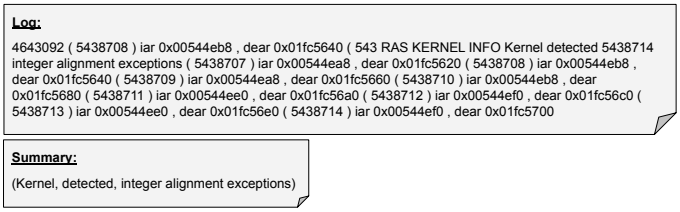


Fig. 9: An example of log summarization.

compared to a naive approach, namely HLAer (200). Interestingly, Aussel et al. (116) found that using simple NLP-based log parsing techniques can raise the performances of log mining from an F-score of 96% (when using a log parser in combination with an LSTM deep neural network) to 99.2%. On the other hand, Zhao et al. (215) and Chen et al. (71) proposed improvements to the execution time of log pattern extraction algorithms. Specifically, Zhao et al. (215) proposed an approach based on text similarity that can determine which pattern a log statement belongs to with a much smaller number of comparisons, while Chen et al. (71) leveraged a bitmap structure [42] and log parsing, respectively.

Table 7: Summary of the approaches used in log mining & pattern extraction.

Objective	Family of techniques	Leveraged data	Description	Papers
Log mining	Machine Learning	Semantic	Refers to the semantic meanings of the variables in the event logs. Each event is described by a single line in the log files	(216)
	Deep learning	Textual	Refers to the text in the event logs, which contains information about the activities recorded by a software system. Each event is described by a single line in the log files	(199; 202)
	Spark and ELK			(220)
	MapReduce			(62)
	Log parsing			(116)
Log pattern extraction	Text similarity	Semantic	Refers to the semantic meanings of the variables in the event logs. Each event is described by a single line in the log files	(215)
	Log parsing	Textual	Refers to the text in the event logs, which contains information about the activities recorded by a software system. Each event is described by a single line in the log files	(71)

4.3.5 ***Improving log analysis:***

Log analysis is the process of deriving critical information about the system's performance and security issues as well as knowledge about users' behaviour, from raw log files or preprocessed ones using the previously discussed techniques such as log template identification, log parsing, log clustering and summarization and log mining and pattern extraction. With the increasing amount of log data generated by various systems and applications, efficient

[42] Collection of binary values where each bit represents whether a log chunk contains logs of a certain format

log analysis has become a challenge. Previous studies have worked on addressing such challenges by improving log analysis in different ways, e.g., (finding 1) characterizing the state-of-the-art in log analysis, (finding 2 + finding 3) proposing novel analysis approaches, and developing scalable and efficient log analysis frameworks.

Finding 1: A large body of research conducted surveys and systematic mapping studies to characterize the state of the art in log analysis. A large body of research (8; 110; 175) conducted comprehensive reviews and systematic mapping studies of the existing literature on automated log analysis approaches. Some studies (8) identified the main challenges faced by practitioners (e.g., clustering approaches do not support the real-time analysis of huge data sets, log parsers based on regular expressions suffer from slow runtime performance) and discussed some practical solutions. Among these solutions is an incremental clustering approach proposed by Skopik et al.(43) that implements character-based clustering enables real-time analysis of logs, tree-based parsers to reduce the complexity of parsing and improve their performances. Whereas other studies highlighted the advantages (e.g., monitoring of user or system performance, detection of security threats) and limitations (e.g., scalability, the use of artificially produced log datasets) of different automated log analysis techniques (110; 175). Finally, He et al.(166) carried out a comprehensive study on log analysis at Microsoft by surveying 105 employees using a questionnaire of 13 questions and interviewing 12 employees. They also compare industrial practices with academic research to gain insight into the gap between research and practice. For instance, while research is mostly focused on analyzing unstructured log data, in reality, the majority of logs are either structured or semistructured.

Finding 2: Several studies proposed approaches to build scalable and efficient log analysis frameworks. While many researchers (204; 72; 152; 189) leveraged the distributed computing and storage features offered by Hadoop to efficiently analyze large amounts of logs data, Mavridis et al. (63) evaluated the performance (i.e., scalability, resource utilization, and power consumption) of Hadoop and Spark in terms of log analysis, and found that Spark performs better than Hadoop. So, Li et al. (203) proposed to leverage Spark for analyzing structured log data. Additionally, Kiran et al. (31) designed a framework that integrates Spark, Apache Flume, Apache Kafka, and ELK Stack to deliver a platform that combines online and offline big data processing methods while enabling real-time analysis. Interestingly, Instead of using three observability solutions for logging, tracing and collecting metrics separately, Kratzke et al. (113) presented a unified architecture for processing several thousand events per minute.

Wagner et al.(180) aimed to address the issue of slow performance in log management systems due to the large size of log data and the complexity of queries. To do so, they proposed to maintain a sample of the logs in a smaller database known as a sublog, in order to improve query performance, but at the cost of reduced accuracy. Setayeshfar et al. (119) proposed a graphical log analysis system, namely GrAALF, to help identify ongoing abnormal behaviors

and protect systems more effectively. Di et al. (162) developed LogAider, a toolkit that can reveal three types of potential correlations among fatal log event, which is essential for system administrators to understand their causality and improve the efficiency of the system.

Finding 3: Researchers also leveraged graph-based analysis techniques to assist practitioners with different log analysis tasks. For instance, Ekelhart et al. (7) and Kurniawan et al. (84) developed two frameworks to automatically construct knowledge graphs (i.e., network of log events and their properties such as type, timestamp, etc. which can be used in log analysis to create structured representations) from heterogeneous raw log messages. Such knowledge graphs can be used for several analysis tasks. For instance, Xie et al. (223) developed a framework called LogM that leverages knowledge graphs technology in addition to deep learning models to improve the failure prediction and log analysis in Hadoop platform. Another type of graphs that can be used for log data analysis are correlation graphs (Similar to knowledge graphs but specifically describe the statistical relationships and dependencies between different events). For example, Platini et al. (95) presented a tool designed to assist human operators in analyzing logs by automatically constructing graphs of correlations between log entries. Furthermore, temporal graphs (i.e., a type of graph that represents relationships between entities that evolve over time) were also used with log analysis. In fact, Zuo et al. (233) proposed to use data mining and analysis methods to reveal the inner structures of the system and visualize the services' temporal relations using temporal graphs to gain deep understanding of a system behavior over time.

Finding 4: The remaining studies aimed to improve log analysis by developing novel approaches and frameworks to address a variety of challenges. Prior studies (127; 181; 96) aimed to provide all-in-one frameworks and libraries that cover multiple tasks of log management at the same time, such as log placement, log compression, log clustering and summarization, log parsing and anomaly detection. Other studies (2; 196; 76; 222; 121; 56; 92) aimed to improve the practice of log representation and log differencing. Log representation involves converting unstructured log data to structured vectors or matrices, and several approaches were introduced for this purpose(2; 196; 76). Log differencing on the other hand involves comparing logs to identify differences between various execution scenarios, was also addressed by prior work (56; 92). To analyze log data and trace it back to its original log statement was another challenge that researchers focused on. Hickman et al. (102) and Schipper et al. (26) addressed such challenge using Elasticsearch and state-of-the-art log parsers, respectively. Finally, practitioners often have difficulties understanding the meaning of specific log lines and their impact on the system, therefore, Shang et al. (197) introduced an approach that can associate the development knowledge (e.g., code commits, issue reports) with specific log lines to help practitioners unravel the meaning and impact of specific log lines.

Topics discussed in StackOverflow:

We observed only one topic (Topic #9 in Table 8) on log analysis, specifically about the parsing of log data. In fact, developers can seek help to extract structured information from unstructured log files, or to parse log files into JSON format as well as parsing complex log messages that are in the JSON format. For example, a developer posted a question [43] asking for help to process a log file that is a structured as a collection of json based strings, and extract the information they are looking for.

Summary of Log Analysis

We observed extensive work on automated log analysis techniques including log parsing, log clustering, log mining, etc. whereas in practice, developers mainly focus on parsing large amounts of log data into JSON format as well as parsing complex log messages that are in the JSON format.

4.4 Quantitative Results

The goal of our analysis of StackOverlfow (SO) questions is to highlight the SO topics that have not been addressed by the literature and compare them with the addressed topics in terms of popularity and difficulty.

Table 8: Software logging topics in StackOverflow (SO). The 1st column represents the high topic label, the 2nd column represents whether the topic was covered by the literature, the 3rd and 4th columns are the fine-grained topic label and id, respectively. The 5th and 6th columns represent the topic's description and an example question from SO, respectively. The 7th column represents the IDs of the studied questions in StackOverFlow. The 8th column describe which subsection discusses the sub-topic. Note that sub-topics #3, #15 and #20 do not appear in the table because they were "Unclear" as we could not label such topics based on the keywords and the studied examples in SO. We also observed that such topics include various questions that are not necessarily related to logging.

High-Level Topic	Covered	Fine-Grained Topic	Id	Topic description	Example from StackOverflow	Studied questions	Subsection

[43] https://stackoverflow.com/questions/65725709

Handling log files	No	Handling log files	0	Practitioners ask questions about how to handle their log files such as how to perfrom log rotation (e.g., start writing to a new log file once a certain limit file size is reached) or how to store upcoming log files in a way to avoid files overwriting each other	We have a JBoss Server which has 6 instances named as [...] In the java code we have Log4j.properties and log file path is [...] Right now,for all 6 instances the logfile is being created at home/crm/logs. But it is required that for all 6 instances,the log file is created at [6 different paths].The logs should also go in their respective log file paths correctly [...] How should I get these paths dynamically in log4g properties?	62698936 51926558 41755437 63618430 41828173	4.2
Dependency configuration of logging libraries	No	Interference between logging libraries	1	The dependencies that practitioners use leverage different logging frameworks causing conflicts which end up generating incorrect logs or logs not respecting the configuration specified. This issue particularly became a significant concern after the security vulnerability related to Log4J. While practitioners attempted to upgrade the dependency, the issue persisted as the dependency was still present since it was coming from another dependent component	I have a Spring application which I inherited. The application contained some log4j.properties files, but these were obviously not in use, as changing these had no effect on log output [...] The java classes all seem to use the slf4j "Logger" interface, but I know slf4j is just a facade that is calling a logging framework. I'd like to configure the logging framework, but I don't know which one is in use, so experimenting by creating different config files in different locations is not getting me very far. How do I determine which logging framework is being used by this application? How do I determine which logging framework is actually in use? [...] Right now Hibernate appears to respond to the contents of a logback configuration file being present, but oddly enough the rest of logging from Spring Boot does not. Also the presence/absence of a log4j.xml file has no effect. List of Dependencies is here [...]	77039617 70376682 71142413 55231609 66316336	4.1
		Misconfiguration of Maven to import logging dependencies	6	Developers can make errors in their configuration of Maven to import logging dependencies, resulting in runtime errors and the inability to log. For instance, an incorrect scope for the log4j dependency in the configuration file can lead to runtime issues when running the application	I am trying to write a maven integrated Java API. I have included log4j for logging purpose. Which works well when running through eclipse, but when maven package is done and the jar is run it is unable to run from cmd line [...] throwing an error [...] Have tried searching for answers but none worked for me. Any help available	46324800 76170319 62244070 71215247 45650710	4.1

	Configuration and dependency issues related to Winston	12	Practitioners face configuration issues related to importing the dependency of Winston as well as configuraiton issues related to the use of Winston logs such as coloring the appropriate logs or formating the log entries	After I updated winston to version 3.2.1, I get an error while trying to hot recompile the project (when my project is started and I make changes). I tried to update all my dependencies to the latest versions, but this did't help. It seems webpack-hot-middleware doesn't work correctly with the latest version of winston. I would be grateful for advice on how to fix this. Logger config: [...] Dependency versions: [...]	60149797 55069013 58854575 58581128 75283923	4.1
	Log configuration errors	10	Practitioners face configuration errors when configuring their logs such as a log file not created, an empty log file, etc., especially when they try to configure multiple logging files at the same time	I'm using Python's logging module to log an installation script. The logs are being printed to the console, but the log file is being created empty. I'm loading the conf from an external file [...] The config file logging.conf has the following config [...] The install.log is always created empty. No matter what handler I select in the logging.conf it only writes to the console	46611637 74717128 71517149 47405336 64917888	4.1
Infrastructure related No configuration	Usage of mobile infrastructure logs	2	Users often report their android logs and ask for help to better understand them, especially due to missing information	Im experiencing an error in my app affecting almost 80% of my active users. It seems it has something to do with the thread [...] but I haven't been able to track it down, the worst thing of all is that I don't even know when this crash occurs, my guess is that it happens when the app is launching [...] Hope you can help me out, I am new to this crash tracking thing. Here is the crash log [...]	63079220 54126073 46819560 51082628 53811892	4.3
	Usage of server infrastructure logs	4	Users may try to understand an error from their log files, but they either do not find them due to a configuration issue in the system or find it difficult to interpret such logs	[...] So, the situation here is that i have a [system] which should accept connections on port 514 in TCP, and after that it saves the logs in a file called "logs.txt". The problem here is that [...] the client is trying to send logs in TCP, but it seems to me that the server is dumping them for some strange reason? [...] maybe there are too much programs sending their logs and my server can't keep up? If that's the case, what should I do about it?	53388875 70060597 72080671 59078561 58553023	4.1
	Understanding web-server logs	7	We observe that developers try to parse some relevant information out of verbose logs (e.g., whether a query is http or https) or simply understand the logs of their web servers or middelware	I'm going through my IIS logs. I want to see if a request was sent with http or https protocol. How do I see that in the logs?	53794889 43927446 45549075 60358763 70527238 57686042 47510292 58535580	4.3

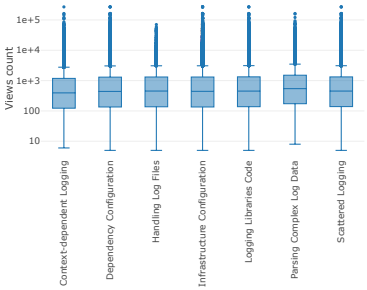
		Log management for subprocesses and bash scripts	21	Practitioners can face challenges when trying to log a subprocess and therefore seek help on SO. For instance, some users may ask for how to handle subprocess logs, while others may want to know how to customize the way a subprocess outputs logs	Below is my script which calls another script using below example and produces entire output after execution is completed while expectation is that it should produce live output as well as write in file at the same time. What happening is that its not producing output line by line and printing entire output in the end [...] I also tried with below command but it prints only on Terminal and not saving output in file.import subprocess [...]	47975233 46125449 47969034 55863060 63323050	4.1
		Log configuration for external libraries	16	Developers may find that the 3rd party libraries they use generate verbose logs and therefore ask questions on how to configure them in order to have a lower verbosity level	I am using maven to import external dependency BrowserMob Proxy into my Java project: [...] However, when I am using this library it is VERY verbose in the output at the DEBUG level, to the point where I can barely see what my program is doing. How can I reduce the amount of output this library produces? I am basically seeing a lot of this: [...]	47212912 56629552 58191785 49741241 60393447 64180289 71093903 59349852	4.1
		Disabling application logging	23	Surprisingly, in some cases, developers ask questions about how to completely disable the logging of an application but the framework they use don't have such an option (e.g., javalin)	How I can disable the hibernate jpa logs in console? Is it possible to disable them in persistence.xml file? The logs are same this: [...]	57959261 56056016 46482267 66430666 47514027	4.1
		Log configuration in Laravel	13	Users face various challenges when configuring logs in Laravel. For instance, a user may be unable to write log files due to wrong permissions in the log location. Other users can find it difficult to log to different files in Laravel. Another issue we observe is when the logs of Laravel are too verbose they hide relevant information	I've a laravel app in which I log messages (i'm using stderr as output) so the log messages appear. the problem is that laravel internally logs so many messages that my log almost lost in those internal messages. for instance, this is the output in which the first message is the one that I have using Log::warning() [...] Is there a way to turn off laravel internal logging so that only my log messages appear in the output?	47590983 67603621 58351511 61100619 77468171	4.1
Usage of logging libraries in the code	Yes	Misusage of logging API	5	Practitioners may not get the right log levels printed due to a misusage of the logging API (e.g., logging API in Python), and such a misusage can be coming from the documentation	The below code is copied from the documentation. I am supposed to be able to see all the info logs. But I don't. I am only able to see the warn and above even though I've set setLevel to INFO. Why is this happening? foo.py: [...] Output: [...] Where did the info and debug messages go??	46700371 63973214 57905884 56799138 41796934	4.1

		Usage of logs in the code	11	Developers ask questions about the appropriate way to use logging library in their code, e.g., in multi-threading applications, how to use logfactory, etc.,	My class to be tested is of HostApi with the static logger [...] My JUnit test class is HostApiTest [...] Without the logger in the code (i.e. commenting out in HostApi) it works, however after adding logger it throws assertion error. I added static mocking of logfactory however it does not seem to work. What is it that I am doing wrong in mocking? I can only use powermock	43470719 47152070 42712923 56901812 46447627	4.1
		Logging challenges in .NET projects	18	Users can face different challenges related to the usage of logs in .Net applications, and therefore ask for help on how to correctly log them	I am trying to build a custom LoggerProvider that writes all logs to the database via a repository/dbcontext. I added logging in the ConfigureServices and Added the new provider in the Configure as follows [...] I created a standard LogRepository which uses the datacontext that was registered in the ConfigureServices [...] This is the implementation of the ILogger [...] Im calling the logger from my APIController which is as follows [...] I put a breakpoint on my LogRepository and when I hit that breakpoint I see that there is a DataContext. However, when the LogInformation is called and I look at the Repository, the DataContext is NULL. What am I missing?	62148204 42767097 49973991 64759300 66579896	4.1
		Logging challenges in Java projects	19	Similarly, practitioners can face various challenges related to the usage of logs in Java applications and ask for help on how to log such applications	I was following this answer in order to add a appender on runtime. Even though that works for the original poster, I get this exception in line Logger logger = (Logger) LoggerFactory.getLogger("abc.xyz") [...] why does it work for him and not for me?	63224928 64916258 56801799 49382554 47114560	4.1
Scattered logging	No	Configuring multiple log locations	8	Developers aim to separate different types of logs in different outputs but face challenges related to the configuration. For instance, they may face issues redirecting some logs to a specific appender and other logs to another different appender. Other users may want to save logs to different file for different frameworks	I have a logback-test xml as follows. I wanted to filter out any logs from package org.jdbcds from console and from GENERIC_LOG_FILE and have them in a seperate file under appender SQLFILE. But neither of them works. All the logs from org.jdbcdslog logged in console and in GENERIC_LOG_FILE, but not under the file given in appender SQLFILE. Please help.	44430465 47678582 61056186 46078774 47411556	4.1

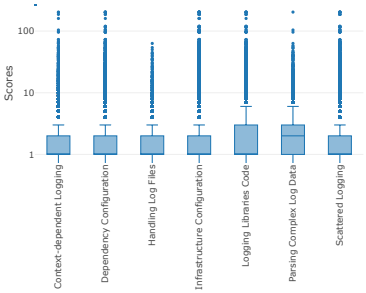
Distributed logging	14	Developers can face difficulties logging accross multiple concurrent processes or logging particular files in multiple process environments. In some cases, practitioners try a solution that does not work, and in other cases the solution does not work incorrectly. For instance, when running two subprocesses, they can observe two logs created in each subprocess instead of one log in each subprocess	I want to log to a single file in a multiple process environment. I can get the sample code work from python logging cookbook. But when I replace with the ProcessPoolExecutor it's not working [...]	44625844 75888699 57136573 67297494 53057510	4.1
Logging databases	22	Practitioners can ask various questions on the logging of databases. For instance, practitioners may try to log two or more tables in a database but their logger object only logs one table. Others simply don't know how to log data entries or updates in a database	I need to create a CRUD logger, and its working as well with folowing script, but the problem is that the script does in only one table... Objective: Install trigger above in ALL tables, not in ONE TABLE. I tried to put script 1 into script 2 but no sucess, with so much concatening errors. Script 1 (its working but in only a table): [...] Script 2: [...] My try: [...]	74447130 67871555 53674649 61424410 67853661	4.1
Transfering log data	24	Developers seek help to configure logs to be sent or collected from different applications such as Elasticsearch, Fluentd, CloudWatch and Kubernetes. Some of the issues they may face is when they try to send different logs to two or more different applications at the same time, but they see that all logs are being pushed to both applications. In other cases, the configuration does not send logs to the target application	I have my fluentd-daemonset configured in kubernetes cluster to send logs to cloudwatch. I followed this tutorial and set fluentd. However in cloudwatch I can see that I see the logs by fluentd as well. How can I stop fluentd logs to be pushed to cloudwatch? This is the config file I use [...]	56982947 77293964 60710515 62550249 74887412	4.1

Parsing complex logging data	Yes	Parsing complex log data	9	Practitioners ask questions related to parsing log files into JSON format as well as parsing complex log messages that are in the JSON format.	I'm trying to parse JSON logs into their own tags from a file which have the format [...] The sections contain an arbitrary amount of data. What I want is the nested top level keys in "transaction" to be tags so that when I view them in Elastic-Search they are separate drop downs. Right now I can parse the value of "transaction" and send it with a single tag, which shows up as a single long string containing all of the data. This result looks like this: [...] How do I create separate tags for each of those keys so it looks like: [...]	67415884 62598075 69805480 65725709 75133191 75721839 76699793 72585200	4.3
Context-dependent usage of logs	No	Context-dependent usage of logs	17	Practitioners ask why certain irrelevant information are logged (e.g., IP address), while others ask question on how to get relevant information for a given context (e.g., how to extensively log an application for a secure migration)	I've been asked to put something in place into our programs in order to log somewhere what's happening on forms... Logging something such ...clicked either this or that button, context menu, events fired on components and so on.. This mostly because we are dealing with legacy code [...] we would like to track everything is happening (or most of it) from an user's point of view [...] Do you know if there is any library that might do something similar? If not, how would you try to achieve this?	50170788 76384954 44580011 51590217 44921694	4.1

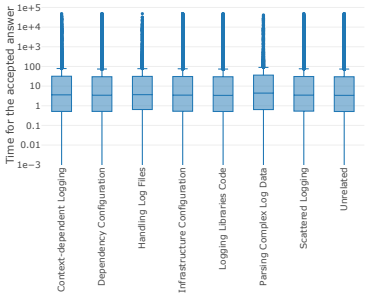
Finding 1. Our LDA analysis of the logging questions in Stack-Overflow reveals seven different high-level topics (and 22 fine-grained topics), out of which five were not covered by the literature. These topics revolve around dependency configuration of logging libraries, infrastructure related configuration, scattered logging, context-dependant usage of logs and handling log files. For instance, no studies exist on how to guide developers in the configuration of logs, whether it is to configure dependencies or the infrastructure, even though it is a common topic that poses several challenges to developers. In fact, we observed a total of 18,623 questions about the configuration of logs, out of which only 7,102 have an accepted answer. Furthermore, such answers are given after an average time of 19.25 days (i.e., 462 hours). This suggests that topics related to log configuration are challenging for many developers. On the other hand, little is known about scattered logging which refers to the practice of separating various types of logs into different outputs, presenting developers with intricate configuration challenges. Such challenges include redirecting specific logs to different appenders, logging multiple databases or multiple tables in the same database, and logging across multiple concurrent processes. Similarly, handling log files, including topics like log rotation and strategies to prevent file overwriting, has not been thoroughly investigated as research focused more on log compression and secure log storage. Finally, there are no studies that aim to guide developers in



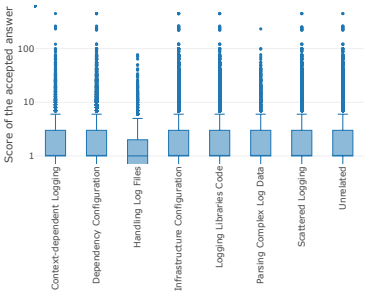
(a) The views' count for our analyzed questions.



(b) The scores of our analyzed questions.



(c) The necessary times in hours to get the accepted answers.



(d) Scores of the accepted answers.

Fig. 10: Distribution of different metrics related to the seven high-level topics identified through our LDA analysis

their logging decision in specific contexts such as the type of a software system. For instance, a user asked for help on extensively logging a legacy system for a secure migration, highlighting the relevance of the application type and development objectives in this context. In another scenario, a user expressed surprise regarding the logging of their IP address, arguing that such information is unnecessary for a simple web application and could pose a security threat. In this case, the context revolves around the goals and requirements of the user's web application. Therefore, we suggest future studies to further identify relevant contexts for logging decisions and assess how to incorporate such contexts into the existing solutions in the literature. Table 8 presents all the analyzed topics and also shows which topics are covered by the literature, and which topics require further attention from researchers.

Finding 2: Challenges not covered by the literature are not less important than those that were addressed in terms of popularity (i.e., number of views) and difficulty (i.e., time to get a response). Most (a median of 75.26% over the seven high-level topics) of the analyzed questions receive an accepted answer before 24 hours, and the accepted answers tend to have more upvotes than downvotes. Our analyzed questions receive an accepted answer after a median time between 3.02 and 4.41 hours (median of

3.49 hours), as shown in figure 10c. Using the Scott-Knott test, we observe that all the high-level topics are ranked in one cluster based on the times for the accepted answers, as well as the count of views, suggesting that the topics that are not covered by prior work are not less popular nor less difficult than those that are covered. On the other hand, the accepted answers tend to have more upvotes than downvotes as they get a median score of one (i.e., the difference between the upvotes and downvotes) , as shown in Figure 10d. On the other side, all our analyzed questions are popular, tend to have positive scores and receive at least one answer. Such questions have median views between 393 and 543 views over the seven high-level topics, and can get up to 269K views, as shown in Figure 10a.

5 Discussion

We observe through our systematic literature review and our study of logging questions in StackOverflow that many practical challenges of logging have been covered by literature, yet there are still some gaps between research and practice. Below, we list a set of implications that concern both researchers and practitioners:

For researchers:

We recommend that future studies propose approaches for handling heterogeneous logs, which are logs generated by various frameworks and third-party dependencies. In practice, not only does the main application developed by developers generate logs, but also other dependencies imported by developers can produce their own logs. While previous studies suggested approaches to control log verbosity, they primarily focus on how to instrument an application to avoid generating an excessive amount of verbose logs. However, the dependencies used by developers can generate logs that increase the verbosity of log files, making them more challenging to parse and analyze, and potentially obscuring relevant application information. This becomes particularly problematic when different versions of log libraries come from various dependencies, leading to conflicts in the configuration of logging libraries and errors or inconsistencies in the generated logs. For example, we have observed cases where developers attempted to upgrade the Log4j dependency due to the recent Log4j vulnerability, while other dependencies still included the vulnerable version. Therefore, we recommend that researchers concentrate on reducing the verbosity of the entire application stack, not just the application itself. Additionally, we suggest future studies investigate the overhead introduced by logs from dependencies in log analysis. In general, we propose exploring the impact of logs from third-party dependencies on logging practices, including log parsing and analysis. Furthermore, we recommend future studies focus on developing approaches to minimize inconsistencies between different versions of logging.

We suggest that future studies explore methods for merging different logging files to enhance log storage and analysis accuracy.

We observe that practitioners often generate distinct logging files and collect logs from various heterogeneous sources, introducing additional complexity in the log parsing process and potentially affecting log storage. For example, practitioners may choose to store different log levels in separate files, obtain logs from different concurrent processes, and even from multiple databases. Parsing a single log file may provide an incomplete picture of the system's execution. Similarly, as existing studies on log compression typically assess their approaches within a single file, we suggest that future studies assess their methods for compressing various files. Different files may have distinct formats, potentially influencing the performance of the compressors proposed and evaluated in the literature. Thus, we encourage future studies to explore how to combine logs from diverse sources, compress them, and integrate them into the log analysis process, enabling practitioners to analyze their issues more effectively.

We recommend future studies to consider the comprehensibility of logs, especially those generated by the infrastructure on which an application is deployed. We observe several cases where practitioners face difficulties in understanding logs related to the infrastructure on which their application runs, claiming that they cannot locate the desired information or interpret these infrastructure logs. The observed infrastructures include mobile infrastructure (e.g., system, device, and iOS), web server infrastructures (e.g., web servers, API servers, middleware), and web frameworks (e.g., Laravel). Therefore, we suggest that future studies focus on enhancing the overall comprehension of logs, with a specific emphasis on infrastructure logs. Additionally, we observe cases where practitioners seek guidance on controlling the verbosity of such logs. Hence, we encourage future studies to assist practitioners in configuring third-party dependencies to retain relevant information from dependencies and infrastructure without generating verbose logs that obscure relevant details from the application or negatively impact application performance.

We suggest future studies to establish best practices related to the usage of logs within the code and explore the generalizability of such practices across different technologies. We observe various questions from developers in SO seeking guidance on the usage of a logging library (e.g., how to instantiate the logger object). Additionally, we observe cases where practitioners misuse logging libraries have. While our literature review has identified several studies offering guidance on instrumenting systems by guiding where and what to log, there is a notable absence of studies addressing the proper usage of the logging libraries itself. This gap includes understanding potential misusages that could impact the performance and correctness of logging activities, as well as identifying specific issues related to logging libraries in a given technology (e.g., .Net or Java).

While a large body of research exists on parsing unstructured log data, we recommend that future studies expand these existing approaches to handle complex log data, such as data presented in JSON format. We observe a whole topic regarding how to parse log data

written in JSON format. Thus, there is a need for future studies to empirically identify various formats of log messages and assess the extent to which existing studies on log parsing can be applied to these advanced logging data formats. Additionally, we observe questions about parsing log files, extracting relevant information, and presenting the results in JSON format. Despite the existence of several studies on log parsing, practitioners in these scenarios often resort to using simple regular expressions.

While previous studies, particularly those focusing on the instrumentation of software systems, introduced various approaches to help developers in their logging practices, such studies neglect the importance of considering the context in which their solutions are applied. Only one study (35) emphasized that utilizing a log-level prediction model should account for the context of multi-component systems. They discovered that log-level prediction models perform poorly when evaluated separately for each component of a multi-component system and recommended leveraging context-based machine learning models. We encourage future studies to extend this research by exploring diverse contextual factors. One observed context in our discussions pertains to the type of software system. For instance, a user asked for help on extensively logging a legacy system for a secure migration, highlighting the relevance of the application type and development objectives in this context. In another scenario, a user expressed surprise regarding the logging of their IP address, arguing that such information is unnecessary for a simple web application and could pose a security threat. In this case, the context revolves around the goals and requirements of the user's web application. Therefore, we suggest future studies delve deeper into identifying relevant contexts for logging decisions and assess how to incorporate such contexts into proposed solutions in the literature.

For practitioners:

Prior studies can help developers instrument their code by suggesting the location where they need to insert log statements, the log verbosity levels, and which variables to log. By studying the literature, practitioners can gain knowledge and make informed decisions when instrumenting their code. For instance, they can find solutions and best practices to determine the appropriate locations for inserting new log statements with the appropriate log verbosity level and the relevant variables to log. This will help them capture the necessary information for troubleshooting and debugging while also avoiding performance and storage overhead caused by excessive logging. Furthermore, the literature emphasizes the importance of periodically reviewing existing log statements and provides guidelines and ML-based solutions to recommend log revisions. By reviewing their logs regularly, practitioners can better identify anomalies and address potential problems before they escalate.

We also observe that the large body of literature offers a variety of log analysis techniques that are not being leveraged by practitioners. We observed extensive work on automated log analysis techniques using machine learning, deep learning and several customized algorithms, which in-

cludes log parsing, log clustering, log mining, etc. whereas in practice, developers mainly focus on parsing large amounts of log data using basic techniques such as regular expressions which is not efficient and requires manual efforts. We suggest that practitioners need to maintain up-to-date knowledge with the latest research on automated log analysis solutions to help them efficiently analyze log data, and make informed decisions.

Similarly, we suggest that practitioners leverage the novel solutions provided by recent studies to optimize and secure log storage. Through our analysis of logging questions in StackOverflow, we observed that developers may ask for expert help to mitigate security threats after detecting malicious activities in their server logs. In such cases, using the pseudonymization or immutable log storage techniques offered by literature can help practitioners protect the private and sensitive data in log files (e.g, usernames, passwords, etc.). So practitioners need to get more familiar with the latest research on secure log storage to better protect their log data from external threats.

6 Threats to validity

External validity: Our external threat to validity is the generalizability of our results to all the topics and problems that practitioners face. There may be challenges that we miss in our study. However, StackOverflow is a popular website for practitioners to discuss their challenges, and we end up studying a large number of questions (20,766 questions). Among these questions, we obtained seven high-level topics and 22 fine-grained topics, for each of which we qualitatively studied five to ten questions. From our study, we found topics that have not yet been addressed in the literature, and we suggest future studies to replicate our work to identify more topics to be covered in the literature.

Internal validity: The first internal threat to validity is related to the selection of papers based on the queries and inclusion/exclusion criteria. Although our query, inclusion, and exclusion criteria are carefully defined, we can miss relevant papers. For example, our search query can miss relevant papers that do not have the "Software" keyword in the venue title. To mitigate such a risk, we conducted extensive backward and forward snowballing to identify as many relevant papers as possible to end up with a large number of studied papers (204 papers).

A second threat to validity is related to the labelling of different papers in the specific categories of logging. Our labelling might be impacted by the subjective analysis of a rater and the misclassification of a paper. To mitigate such a risk, we conduct a rigorous systematic qualitative analysis in which each paper is rated by two authors and an agreement/disagreement score is measured. Then, we follow up such an analysis with another iteration to address the disagreement up to reaching a high agreement score as reported in the methodology.

A third threat to validity is related to LDA analysis. Our analysis can be impacted by the number of topics (K = 25) that we identified, as choosing another number of topics could lead to different results. To mitigate such a risk, we manually inspected the cohesiveness of the topics obtained for different values of K, similar to previous studies (57; 186). The labeling of topics can also be affected by subjective analysis in our manual definition of these labels. To mitigate such a threat and define an accurate label for each topic, we qualitatively studied the top questions for each topic, ending up with 149 questions manually studied.

7 Conclusion

Practitioners encounter a wide range of difficulties in the process of creating, maintaining, and using logs. One of the significant challenges involves making informed decisions regarding where to log, what to log, and which log verbosity level to use to strike a balance between collecting enough details in logs and minimizing the performance and storage overhead. Furthermore, modern software systems generate huge amounts of log data every day, which may impact the efficient storage of log data and the protection of sensitive information. The analysis of large volumes of unstructured log data with different formats is also a major challenge.

In this study, we combine a systematic literature review of 204 papers on the entire logging pipeline, from the creation to the analysis of logs, with a qualitative and quantitative analyses of 119 and 20,766 software logging questions discussed in StackOverflow (SO), respectively, to better understand the current state of software logging in both research and practice. Our objective is to present a comprehensive overview of the knowledge in this field, to help practitioners choose the best solutions for their challenges. Additionally, we aim to identify the gap between research and practice by investigating the common challenges faced by developers when logging their applications and to identify the challenges that have not been addressed by the literature.

Our results show that the literature covers the complete pipeline of log management, from log instrumentation to log storage and log analysis. In particular, we observe that 53% of our studied papers focused on improving the fundamental components of log analysis (e.g., log parsing, log clustering, log mining), 37% focused on the creation of new log statements, and 10% focused on improving log storage. Through our LDA analysis of SO questions, we identified seven high-level topics (and 22 fine-grained topics) out of which five are not covered by the literature. Such topics revolve around dependency configuration of logging libraries, infrastructure related configuration, scattered logging, context-dependant usage of logs and handling log files. Our findings serve as a guide for developers in existing studies, covering all aspects of software logging from development to logging analysis. Our findings also highlight areas that have not yet been explored in the literature.

Declaration of Conflict of Interest

The authors declare that they have no conflict of interest.

Data Availability Statements

Our replication package is available on: `https://github.com/MABATOUN/SLRReplicationPackage.git`.

References

1. A, K.Z., Donghwan, S., Domenico, B., Lionel, B.: Guidelines for assessing the accuracy of log message template identification techniques. In: Proceedings of the 2022 International Conference on Software Engineering, pp. 1095–1106 (2022)
2. A, S.M., Shameem, P., Ji, L., Youssef, M., Maguette, T., Fawaz, A.Q., Ting, Y., Sanjay, C.: Log representation as an interface for log processing applications. Journal of Information Security and Applications p. 103021 (2021)
3. Adetokunbo, M., Nur, Z.H.A., E, M.E.: A lightweight algorithm for message type extraction in system application logs. Journal of Systems and Software pp. 1921–1936 (2011)
4. Aidi, P., Wei, C., Will, Z., Xiaobo, Z.: It can understand the logs, literally. In: Proceedings of the 2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 446–451 (2019)
5. Amey, A., Abhishek, D., A, S.N., Darshil, K., Vikram, A., Rajat, G., Rohit, K.: Delog: A high-performance privacy preserving log filtering framework. In: Proceedings of the 2019 IEEE International Conference on Big Data (Big Data), pp. 1739–1748 (2019)
6. Amey, A., Rohit, K., Rajat, G.: Logan: A distributed online log parser. In: Proceedings of the 2019 IEEE International Conference on Data Engineering (ICDE), pp. 1946–1951 (2019)
7. Andreas, E., J, E.F., Elmar, K.: The slogert framework for automated log knowledge graph construction. In: Proceedings of the 2021 International Conference on The Semantic Web, pp. 631–646 (2021)
8. Andriy, M., Abdelwahab, H.L., Enzo, C., Alf, L.: Operational-log analysis for big data systems: Challenges and solutions. IEEE Software pp. 52–59 (2016)
9. Anton, B., W, T.S., E, H.A.: What are developers talking about? an analysis of topics and trends in stack overflow. Empirical Software Engineering pp. 619–654 (2014)
10. Antonio, M., Luca, P., Gabriele, B.: Using deep learning to generate complete log statements. In: Proceedings of the 2022 International Conference on Software Engineering, pp. 2279–2290 (2022)

11. Antonio, P., Marcello, C., Gabriella, C., Domenico, C.: Industry practices and event logging: Assessment of a critical software development process. In: Proceedings of the 2012 IEEE Annual Computer Software and Applications Conference, pp. 169–178 (2015)
12. Arthur, V., Raja, C., Mar, C.Z.: Ustep: Unfixed search tree for efficient log parsing. In: Proceedings of the 2021 IEEE International Conference on Data Mining (ICDM), pp. 659–668 (2021)
13. Artur, V., Leonel, S., L.D.C, C.R., Adail, O., Carlos, R.: The general data protection regulation and log pseudonymization. In: Proceedings of the 2021 International Conference on Advanced Information Networking and Applications (AINA-2021), pp. 479–490 (2021)
14. Artur, V., Leonel, S., L.D.C, C.R., Adail, O., Carlos, R.: Log pseudonymization: Privacy maintenance in practice. Journal of Information Security and Applications p. 103021 (2021)
15. Bo, F., Chentao, W., Jie, L.: Mlc: An efficient multi-level log compression method for cloud backup systems. In: Proceedings of the 2016 IEEE Trustcom/BigDataSE/ISPA, pp. 1358–1365 (2016)
16. Boyuan, C., M, J.Z.: Extracting and studying the logging-code-issue-introducing changes in java-based large-scale open source software systems. Empirical Software Engineering pp. 2285–2322 (2019)
17. Boyuan, C., Ming, J.Z.: Characterizing and detecting anti-patterns in the logging code. In: Proceedings of the 2017 IEEE/ACM International Conference on Software Engineering (ICSE)), pp. 71–81 (2017)
18. Boyuan, C., Ming, J.Z.: Characterizing logging practices in java-based open source software projects –a replication study in apache software foundation. Empirical Software Engineering pp. 330–374 (2017)
19. Boyuan, C., Ming, J.Z.: Studying the use of java logging utilities in the wild. In: Proceedings of the 2020 IEEE/ACM International Conference on Software Engineering (ICSE), pp. 397–408 (2020)
20. Boyuan, C., Ming, J.Z.: A survey of software log instrumentation. ACM Computing Surveys pp. 1–34 (2021)
21. Byungchul, T., Wook-Shin, H.: Lognroll: Discovering accurate log templates by iterative filtering. In: Proceedings of the 2021 International Middleware Conference, pp. 273–285 (2021)
22. Chen, Z., Jianwei, Y., Junxiao, H., Shuiguang, D.: A preliminary study on sensitive information exposure through logging. In: Proceedings of the 2020 Asia-Pacific Software Engineering Conference (APSEC), pp. 470–474 (2020)
23. Chen, Z., Jianwei, Y., Shuiguang, D., Maoxin, Y., Min, F., Tao, X.: An exploratory study of logging configuration practice in java. In: Proceedings of the 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 459–469 (2019)
24. Chen, Z., Shuiguang, D., Junxiao, H., Jianwei, Y.: Towards automatic detection and prioritization of pre-logging overhead: A case study of hadoop ecosystem. Automated Software Engineering p. 11 (2022)

25. Chris, E., Di, Z., Dong, D.: Clusterlog: Clustering logs for effective log-based anomaly detection. In: Proceedings of the 2022 IEEE/ACM Workshop on Fault Tolerance for HPC at eXtreme Scale (FTXS), pp. 1–10 (2022)
26. Daan, S., Maurício, A., van Deursen Arie: Tracing back log data to its log statement: From research to practice. In: Proceedings of the 2019 IEEE/ACM International Conference on Mining Software Repositories (MSR), pp. 545–549 (2019)
27. Daniel, O., Janusz, S.: Log based analysis of software application operation. In: Proceedings of the 2020 International Conference on Dependability of Computer Systems, pp. 371–382 (2020)
28. Daniel, P., Mengjun, X.: Dip: A log parser based on disagreement index token conditions. In: Proceedings of the 2022 ACM Southeast Conference, pp. 113–122 (2022)
29. Daniel, T.: An algorithm for message type discovery in unstructured log data. In: Proceedings of the 2019 ICSOFT, pp. 665–676 (2019)
30. Daniel, T., Andrea, V., Svatopluk, N., Toma, P.: Structured and interoperable logging for the cloud computing era: The pitfalls and benefits. In: Proceedings of the 2013 IEEE/ACM International Conference on Utility and Cloud Computing, pp. 91–98 (2013)
31. Deshpande, K., Madhuri, R.: Modelling auto-scalable big data enabled log analytic framework. In: Computer Networks and Inventive Communication Technologies: Proceedings of Fifth ICCNCT 2022, pp. 857–870 (2022)
32. Diana, E.M., Fabio, P., Yann-Gaël, G., Abdelwahab, H.L., Anas, B.: A systematic literature review on automated log abstraction techniques. Information and Software Technology p. 106276 (2020)
33. Dileepa, J.: Towards structured log analysis. In: Proceedings of the 2012 International Conference on Computer Science and Software Engineering, pp. 259–264 (2012)
34. Ding, Y., Soyeon, P., Peng, H., Yang, L., M, L.M., Xiaoming, T., Yuanyuan, Z., Stefan, S.: Be conservative: Enhancing failure diagnosis with proactive logging. In: Proceedings of the 2012 {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12), pp. 293–306 (2012)
35. E, O.Y., Mohammed, S., Noureddine, K., E, H.A.: An empirical study on log level prediction for multi-component systems. IEEE Transactions on Software Engineering pp. 1–1 (2022)
36. Eduardo, M., Fabio, P.: Log severity levels matter: A multivocal mapping. In: Proceedings of the 2021 IEEE International Conference on Software Quality, Reliability and Security (QRS), pp. 1002–1013 (2021)
37. Fabio, B., Gabriella, C., Marcello, C., Domenico, C., Antonio, P., Agostino, S.: Event logging in an industrial development process: Practices and reengineering challenges. In: Proceedings of the 2014 International Symposium on Software Reliability Engineering Workshops, pp. 10–13 (2014)

38. Fabio, B., Gabriella, C., Marcello, C., Domenico, C., Antonio, P., Agostino, S.: Tell: Log level suggestions via modeling multi-level code block information. In: Proceedings of the 2014 International Symposium on Software Reliability Engineering Workshops, pp. 10–13 (2014)
39. Fanny, R.O.: Engineering forensic-ready software systems using automated logging. In: Proceedings of the 2022 REFSQ Workshops (2022)
40. Fanny, R.O., Liliana, P.: Automated modelling of security incidents to represent logging requirements in software systems. In: Proceedings of the 2020 International Conference on Availability, Reliability and Security, pp. 1–8 (2020)
41. Febrian, S., Erion, T., Fatima, S., Georgios, D., Tsakalidi, D., Panos, K.: Gpt-2c: A parser for honeypot logs using large pre-trained language models. In: Proceedings of the 2021 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, pp. 649–653 (2021)
42. Feng, Z., Xingshu, C., Yonggang, L., Tiemai, H., Zhihong, L., Keer, S.: Spray: Streaming log parser for real-time analysis. Security and Communication Networks (2022)
43. Florian, S., Markus, W., Max, L.: Smart Log Data Analytics. Springer (2021)
44. Guojun, C., Jingyu, W., Qi, Q., Haifeng, S., Shimin, T., Jianxin, L.: Prefix-graph: A versatile log parsing approach merging prefix tree with probabilistic graph. In: Proceedings of the 2021 IEEE International Conference on Data Engineering (ICDE), pp. 2411–2422 (2021)
45. Guoping, R., Qiuping, Z., Xinbei, L., Shenghiu, G.: A systematic review of logging practice in software engineering. In: Proceedings of the 2017 Asia-Pacific Software Engineering Conference (APSEC), pp. 534–539 (2017)
46. Guoping, R., Shenghui, G., He, Z., Dong, S., Wanggen, L.: How is logging practice implemented in open source software projects? a preliminary exploration. In: Proceedings of the 2018 Australasian Software Engineering Conference (ASWEC), pp. 171–180 (2018)
47. Guoping, R., Yangchen, X., Shenghui, G., He, Z., Dong, S.: Can you capture information as you intend to? a case study on logging practice in industry. In: Proceedings of the 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 171–180 (2018)
48. H, L.K., Xiangyu, Z., Dongyan, X.: Loggc: Garbage collecting audit log. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, pp. 1005–1016 (2013)
49. Han, A., Jie, C., Wenchang, S., Jianwei, H., Bin, L., Bo, Q.: An approach to recommendation of verbosity log levels based on logging intention. In: Proceedings of the 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)), pp. 125–134 (2019)
50. Hao, L., Jingyu, Z., Bin, Y., Minyi, G., Jie, L.: Cowic: A column-wise independent compression for log stream analysis. In: Proceedings of the 2015 IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 21–30 (2015)

51. Hao, W., Desheng, Y., Nian, D., Yang, G., Lu, Z.: Medusa: Blockchain powered log storage system. In: Proceedings of the 2018 IEEE International Conference on Software Engineering and Service Science (ICSESS), pp. 518–521 (2018)
52. Haonan, Z., Yiming, T., Maxime, L., Heng, L., Weiyi, S.: Studying logging practice in test code. Empirical Software Engineering p. 83 (2022)
53. Harshit, G., Abhinav, S., Sangeeta, L., Amanpreet, K., A, K., Ashish, S.: Empirical analysis of the logging questions on the stack overflow website. In: Proceedings of the 2018 Conference On Software Engineering & Data Sciences (CoSEDS) (2018)
54. Harshit, G., Abhinav, S., Sangeeta, L., Lov, K.: A three dimensional empirical study of logging questions from six popular q & a websites. e-Informatica Software Engineering Journal pp. 105–139 (2019)
55. Harshit, G., Sangeeta, L., Heng, L.: An exploratory semantic analysis of logging questions. Journal of Software: Evolution and Process p. e2361 (2020)
56. Hen, A., Lingfeng, B., Nimrod, B., David, L., Shahar, M.: Using finite-state models for log differencing. In: Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 49–59 (2018)
57. Heng, L., P., C.T.H., Weiyi, S., E, H.A.: Studying software logging using topic models. Empirical Software Engineering pp. 2655–2694 (2018)
58. Heng, L., Weiyi, S., Bram, A., Mohammed, S., E, H.A.: A qualitative study of the benefits and costs of logging from developers' perspectives. IEEE Transactions on Software Engineering (2020)
59. Heng, L., Weiyi, S., E, H.A.: Which log level should developers choose for a new logging statement? Empirical Software Engineering pp. 1684–1716 (2017)
60. Heng, L., Weiyi, S., Ying, Z., E, H.A.: Towards just-in-time suggestions for log changes. Empirical Software Engineering pp. 1831–1865 (2017)
61. Hetong, D., Heng, L., Shao, C.C., Weiyi, S., Tse-Hsun, C.: Logram: Efficient log parsing using n-gram dictionaries. IEEE Transactions on Software Engineering (2020)
62. Hossein, H., Biplob, D., Jianwu, X., Hui, Z., Guofei, J., Abdullah, M.: Logmine: Fast pattern recognition for log analytics. In: Proceedings of the 2016 ACM International on Conference on Information and Knowledge Management, pp. 1573–1582 (2016)
63. Ilias, M., Helen, K.: Performance evaluation of cloud-based log file analysis with apache hadoop and apache spark. Journal of Systems and Software pp. 133–151 (2017)
64. Issam, S., Abdelwahab, H.L., Otmane, A.M., A, S.M.: An effective approach for parsing large log files. In: Proceedings of the 2022 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 1–12 (2022)
65. Jackson, R., Brooklynn, S., Jan, S., Connor, W., Tomas, C., Pavel, T.: Multi-source log clustering in distributed systems. In: Proceedings of the

2021 Information Science and Applications ICISA, pp. 31–41 (2021)
66. Jan, S., Jackson, R., Connor, W., Brooklynn, S., Tomas, C., Miroslav, B., Dongwan, S., Karel, F., Pavel, T.: On vulnerability and security log analysis: A systematic literature review on recent trends. In: Proceedings of the 2020 International Conference on Research in Adaptive and Convergent Systems, pp. 175–180 (2020)
67. Jared, R., Andriy, M.: On automatic parsing of log records. In: Proceedings of the 2021 IEEE/ACM International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER), pp. 41–45 (2021)
68. Jason, K., Jon, S., Maria, R., Laurie, W.: To log, or not to log: Using heuristics to identify mandatory log events–a controlled experiment. Empirical Software Engineering pp. 2684–2717 (2017)
69. Jason, K., Rahul, P., Laurie, W.: Enabling forensics by proposing heuristics to identify mandatory log events. In: Proceedings of the 2015 Symposium and Bootcamp on the Science of Security, pp. 1–11 (2015)
70. Jeanderson, C., Jan, H., Maurício, A., Arie, V.D.: An exploratory study of log placement recommendation in an enterprise system. In: Proceedings of the 2021 IEEE/ACM International Conference on Mining Software Repositories (MSR), pp. 143–154 (2021)
71. Jia, C., Peng, W., Fan, Q., Shi-Qing, D., Wei, W.: Plq: An efficient approach to processing pattern-based log queries. Journal of Computer Science and Technology pp. 1239–1254 (2022)
72. Jie, Y., Yanshen, Z., Shuo, Z., Dazhong, H.: Mass flow logs analysis system based on hadoop. In: Proceedings of the 2013 IEEE International Conference on Broadband Network & Multimedia Technology, pp. 115–118 (2013)
73. Jieming, Z., Pinjia, H., Qiang, F., Hongyu, Z., R, L.M., Dongmei, Z.: Learning to log: Helping developers make informed logging decisions. In: Proceedings of the 2015 IEEE/ACM IEEE International Conference on Software Engineering, pp. 415–425 (2015)
74. Jieming, Z., Shilin, H., Jinyang, L., Pinjia, H., Qi, X., Zibin, Z., R, L.M.: Tools and benchmarks for automated log parsing. In: Proceedings of the 2019 IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), pp. 121–130 (2019)
75. Jing, Z., Guoping, R., Guocheng, H., Shenghui, G., He, Z., Dong, S.: Jllar: A logging recommendation plug-in tool for java. In: Proceedings of the 2019 Asia-Pacific Symposium on Internetware, pp. 1–6 (2019)
76. Jing, Z., Zezhou, L., Xianbo, Z., Feng, L., Chao, W., Xingye, C.: Posbert: Log classification via modified bert based on part-of-speech weight. In: Proceedings of the 2022 International Conference on Pattern Recognition and Artificial Intelligence (PRAI), pp. 979–983 (2022)
77. Jingyu, S., Bingyu, L., Yuan, H.: Logbug: Generating adversarial system logs in real time. In: Proceedings of the 2020 ACM International Conference on Information & Knowledge Management, pp. 2229–2232 (2020)

78. Jinyang, L., Jieming, Z., Shilin, H., Pinjia, H., Zibin, Z., R, L.M.: Logzip: Extracting hidden structures via iterative clustering for log compression. In: Proceedings of the 2019 IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 863–873 (2019)
79. Josef, S.: Comparison and model of compression techniques for smart cloud log file handling. In: Proceedings of the 2020 International Conference on Communications, Computing, Cybersecurity, and Informatics (CCCI), pp. 1–6 (2020)
80. Julian, H., Haonan, Z., Lili, W., Luca, P., Mauricio, A., Weiyi, S.: Logging practices with mobile analytics: An empirical study on firebase. In: Proceedings of the 2021 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MobileSoft), pp. 56–60 (2021)
81. Junyu, W., Guangyan, Z., Junchao, C., Yang, W., Weimin, Z., Tingtao, S., Jiesheng, W., Jiangwei, J.: Loggrep: Fast and cheap cloud log storage by exploiting both static and runtime patterns. IEEE Transactions on Software Engineering (2023)
82. Junyu, W., Guangyan, Z., Yang, W., Zhiwei, L., Zhanyang, Z., Junchao, C., Tingtao, S., Qi, Z.: On the feasibility of parser-based log compression in large-scale cloud systems. In: FAST, pp. 249–262 (2021)
83. Justin, B., Kristal, C., Mark, G., Ram, S.: A mixture modeling approach for clustering log files with coreset and user feedback. Pattern Recognition Letters pp. 74–80 (2022)
84. Kabul, K., Andreas, E., Elmar, K., Dietmar, W., Gerald, Q., Min, T.A.: Vlograph: a virtual knowledge graph framework for distributed security log analysis. Machine Learning and Knowledge Extraction (2022)
85. Keyur, P., João, F., Abdelwahab, H.L., Ingrid, N.: The sense of logging in the linux kernel. Empirical Software Engineering p. 153 (2022)
86. Kirk, R., Yu, L., Ding, Y.: Clp: Efficient and scalable search on compressed text logs. In: Proceedings of the 2021 OSDI, pp. 183–198 (2021)
87. Klaus, K.: Computing krippendorff's alpha-reliability (2011)
88. Kundi, Y., Heng, L., Weiyi, S., E, H.A.: A study of the performance of general compressors on log files. Empirical Software Engineering pp. 3043–3085 (2020)
89. Kundi, Y., Mohammed, S., Weiyi, S., E, H.A.: Improving state-of-the-art compression techniques for log management tools. IEEE Transactions on Software Engineering (2021)
90. Leticia, D., Daniel, L., Daniele, B.: Explainable log parsing and online interval granular classification from streams of words. In: Proceedings of the 2022 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), pp. 1–8 (2022)
91. Lin, Z., Xueshuo, X., Kunpeng, X., Zhi, W., Ye, L., Yujun, Z.: An efficient log parsing algorithm based on heuristic rules. In: Proceedings of the 2019 Advanced Parallel Processing Technologies: International Symposium, pp. 123–134 (2019)
92. Lingfeng, B., Nimrod, B., David, L., Shahar, M.: Statistical log differencing. In: Proceedings of the 2019 IEEE/ACM International Conference

on Automated Software Engineering (ASE), pp. 851–862 (2019)
93. Łukasz, K., Krzysztof, G.: Landscape of automated log analysis: A systematic literature review and mapping study. IEEE Access (2022)
94. Luyue, F., Xiaoqiang, D., Xu, L., Yiping, Q., Weiwu, R., Qiang, D.: Quicklogs: A quick log parsing algorithm based on template similarity. In: Proceedings of the 2021 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pp. 1085–1092 (2021)
95. Marc, P., Thomas, R., Benoit, P., Noel, D.P.: Logflow: Simplified log analysis for large scale systems. In: Proceedings of the 2021 International Conference on Distributed Computing and Networking, pp. 116–125 (2021)
96. Marcin, K., Janusz, S.: Multidimensional log analysis. In: Proceedings of the 2016 European Dependable Computing Conference (EDCC), pp. 193–196 (2016)
97. Marcin, K., Janusz, S.: Holistic processing and exploring event logs. In: Proceedings of the 2017 International Workshop of Software Engineering for Resilient Systems, pp. 184–200 (2017)
98. Martin, R.C., Leon, M.: Improving problem identification via automated log clustering using dimensionality reduction. In: Proceedings of the 2018 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 1–10 (2018)
99. Masayoshi, M.: Incremental mining of system log format. In: Proceedings of the 2013 IEEE International Conference on Services Computing, pp. 595–602 (2013)
100. Max, L., Markus, W., Florian, S., Giuseppe, S., Peter, F.: Dynamic log file analysis: An unsupervised cluster evolution approach for anomaly detection. Computers & Security pp. 94–116 (2018)
101. Maxime, R., Marc-Olivier, B., Georges, Q.: A novel pattern-based edit distance for automatic log parsing. In: Proceedings of the 2022 International Conference on Pattern Recognition (ICPR), pp. 1236–1242 (2022)
102. Megan, H., Dakota, F., Elisabeth, B., Sean, B., Hugh, G., William, J., Nathan, D.: Enhancing hpc system log analysis by identifying message origin in source code. In: Proceedings of the 2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 100–105 (2018)
103. Mehran, H., Weiyi, S., Emad, S., Nikolaos, T.: Studying and detecting log-related issues. Empirical Software Engineering pp. 3248–3280 (2018)
104. Meiyappan, N., A, V.M.: Abstracting log lines to log event types for mining software system logs. In: Proceedings of the 2010 Working Conference on Mining Software Repositories, pp. 71–81 (2017)
105. Mik, K., C, M.G.: Mylar: a degree-of-interest model for ides. In: Proceedings of the 2005 International Conference on Aspect-Oriented Software Development, pp. 159–168 (2005)
106. Mike, C., X, Z.A., Jim, L., I, J.M., Eric, B.: Failure diagnosis using decision trees. In: Proceedings of the 2004 International Conference on

Autonomic Computing, pp. 36–43 (2004)
107. Min, D., Feifei, L.: Spell: Streaming parsing of system event logs. In: Proceedings of the 2016 IEEE International Conference on Data Mining (ICDM), pp. 859–864 (2016)
108. Min, D., Feifei, L.: Spell: Online streaming parsing of large unstructured system logs. IEEE Transactions on Knowledge and Data Engineering pp. 2213–2227 (2018)
109. Min, D., Feifei, L., Guineng, Z., Vivek, S.: Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In: Proceedings of the 2017 ACM SIGSAC conference on Computer and Communications Security, pp. 1285–1298 (2017)
110. N, H.A., V, P.A., M, G.N., A, H.N., Nicholas, K.: On machine learning approaches for automated log management. J. Univers. Comput. Sci. pp. 925–945 (2019)
111. Nadine, R., Andreas, M.: Flexparser—the adaptive log file parser for continuous results in a changing world. Journal of Software: Evolution and Process p. e2426 (2022)
112. Nan, Y., Pieter, C., Dennis, H., Ramon, S., Johan, L., Alexander, S.: An interview study about the use of logs in embedded software engineering. Empirical Software Engineering p. 43 (2023)
113. Nane, K.: Cloud-native observability: The many-faceted benefits of structured and unified logging—a multi-case study. Future Internet p. 274 (2022)
114. Nathan, B., Jan, B.: Software logs for machine learning in a devops environment. In: Proceedings of the 2020 Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. 29–33 (2020)
115. Nazrin, A., C, G.M.: Log and execution trace analytics system. In: Proceedings of the 2021 International Conference on INnovations in Intelligent SysTems and Applications (INISTA), pp. 1–7 (2021)
116. Nicolas, A., Yohan, P., Sophie, C.: Improving performances of log mining for anomaly prediction through nlp-based log parsing. In: Proceedings of the 2018 IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 237–243 (2018)
117. Oihana, C., Josiane, M., Olivier, T., Xavier, B.: Meting: A robust log parser based on frequent n-gram mining. In: Proceedings of the 2020 IEEE International Conference on Web Services (ICWS), pp. 84–88 (2020)
118. Omar, P.D.A., Vanessa, A.R.: Towards an efficient log data protection in software systems through data minimization and anonymization. In: Proceedings of the 2019 International Conference in Software Engineering Research and Innovation (CONISOFT), pp. 107–115 (2019)
119. Omid, S., Christian, A., Matthew, J., H, L.K., Prashant, D.: Graalf: Supporting graphical analysis of audit logs for forensics. Software Impacts p. 100068 (2021)

120. Palash, D., G, S.: Logea: Log extraction and analysis tool to support forensic investigation of linux-based system. In: Proceedings of the 2021 International Conference on Trends in Electronics and Informatics (ICOEI), pp. 909–916 (2021)
121. Pattapon, P., Phond, P., Chibante, B.V.: A topic modeling for alice's log messages using latent dirichlet allocation. In: Proceedings of the 2022 IEEE International Conference on Knowledge Innovation and Invention (ICKII), pp. 75–82 (2022)
122. Peng, F., Zhou, L., Zhiying, W., Xiao, Y., Ding, L., Kangkook, J.: Seal: Storage-efficient causality analysis on enterprise logs with query-friendly compression. In: Proceedings of the 2021 USENIX Security Symposium, pp. 2987–3004 (2021)
123. Pengyu, W., Zhizhong, Z., Bingguang, D.: Olmpt: research on online log parsing method based on prefix tree. In: Proceedings of the 2020 International Conference on Information Technologies and Electrical Engineering, pp. 55–59 (2020)
124. Peter, B., Moises, G., Armando, F., B, W.D., Hans, A.: Fingerprinting the datacenter: Automated classification of performance crises. In: Proceedings of the 2010 European Conference on Computer Systems, pp. 111–124 (2010)
125. Péter, M., Péter, L.K., Attila, K.: The use of template miners and encryption in log message compression. Computers p. 83 (2021)
126. Péter, M., Péter, L.K., Attila, K.: A novel dictionary-based method to compress log files with different message frequency distributions. Applied Sciences p. 2044 (2022)
127. Pinjia, H.: An end-to-end log management framework for distributed systems. In: Proceedings of the 2017 IEEE Symposium on Reliable Distributed Systems (SRDS), pp. 266–267 (2017)
128. Pinjia, H., Jieming, Z., Shilin, H., Jian, L., R, L.M.: An evaluation study on log parsing and its use in log mining. In: Proceedings of the 2016 annual IEEE/IFIP international conference on dependable systems and networks (DSN), pp. 654–661 (2016)
129. Pinjia, H., Jieming, Z., Shilin, H., Jian, L., R, L.M.: Towards automated log parsing for large-scale log data analysis. IEEE Transactions on Dependable and Secure Computing pp. 931–944 (2017)
130. Pinjia, H., Jieming, Z., Zibin, Z., R, L.M.: Drain: An online log parsing approach with fixed depth tree. In: Proceedings of the 2017 IEEE international conference on web services (ICWS), pp. 33–40 (2017)
131. Pinjia, H., Zhuangbin, C., Shilin, H., R, L.M.: Characterizing the natural language descriptions in software logging statements. In: Proceedings of the 2018 IEEE/ACM International Conference on Automated Software Engineering (ASE)), pp. 178–189 (2018)
132. PWDC, J., NR, W., HKEP, H.: Automatic detection of multi-line templates in software log files. In: Proceedings of the 2017 International Conference on Advances in ICT for Emerging Regions (ICTer), pp. 1–8 (2017)

133. Qiang, F., Jieming, Z., Wenlu, H., Jian-Guang, L., Rui, D., Qingwei, L., Dongmei, Z., Tao, X.: Where do developers log? an empirical study on logging practices in industry. In: Proceedings of the 2014 International Conference on Software Engineering, pp. 24–33 (2014)
134. Qingwei, L., Hongyu, Z., Jian-Guang, L., Yu, Z., Xuewei, C.: Log clustering based problem identification for online service systems. In: Proceedings of the 2016 IEEE/ACM International Conference on Software Engineering Companion (ICSE-C), pp. 102–111 (2016)
135. Rafael, C., Jeff, S., Nur, Z.H., Malcolm, H.: Log abstraction for information security: Heuristics and reproducibility. In: Proceedings of the 2021 International Conference on Availability, Reliability and Security, pp. 1–10 (2021)
136. Ran, C.A., Tse-Hsun, C., Shaowei, W.: Demystifying the challenges and benefits of analyzing user-reported logs in bug reports. Empirical Software Engineering pp. 1–30 (2021)
137. Ran, T., Zulong, D., Haiyang, J., Gaogang, X.: Logdac: A universal efficient parser-based log compression approach. In: ICC 2022-IEEE International Conference on Communications, pp. 3679–3684 (2022)
138. Risto, V., Mauno, P.: Logcluster - a data clustering and pattern mining algorithm for event logs. In: Proceedings of the 2015 International conference on network and service management (CNSM), pp. 1–7 (2015)
139. Rui, Z., Mohammad, H., Haipeng, C., Abdelwahab, H.L.: Mobilogleak: A preliminary study on data leakage caused by poor logging practices. In: Proceedings of the 2020 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 577–581 (2020)
140. Ruipeng, Y., Dan, Q., Yekui, Q., Yusheng, D., Shaowei, Z.: An online log template extraction method based on hierarchical clustering. EURASIP Journal on Wireless Communications and Networking pp. 1–12 (2019)
141. Sahan, S., Haifeng, S., David, C.: ilse: An intelligent web-based system for log structuring and extraction. In: Proceedings of the 2017 Asia-Pacific Software Engineering Conference (APSEC), pp. 588–593 (2017)
142. Salma, M., Annibale, P., Domenico, B., Lionel, B., Raimondas, S.: A search-based approach for accurate identification of log message formats. In: Proceedings of the 2018 IEEE/ACM International Conference on Program Comprehension (ICPC), pp. 167–16710 (2018)
143. Sangeeta, L., Ashish, S.: Logopt: Static feature extraction from source code for automated catch block logging prediction. In: Proceedings of the 2016 India Software Engineering Conference, pp. 151–155 (2016)
144. Sangeeta, L., Neetu, S., Ashish, S.: Two level empirical study of logging statements in open source java projects. International Journal of Open Source Software and Processes (IJOSSP) pp. 49–73 (2015)
145. Sangeeta, L., Neetu, S., Ashish, S.: Logoptplus: Learning to optimize logging in catch and if programming constructs. In: Proceedings of the 2016 IEEE Annual Computer Software and Applications Conference (COMPSAC), pp. 215–220 (2016)

146. Sangeeta, L., Neetu, S., Ashish, S.: Analysis and prediction of log statement in open source java projects. Buenos Aires, Argentina p. 65 (2017)
147. Sangeeta, L., Neetu, S., Ashish, S.: Three-level learning for improving cross-project logging prediction for if-blocks. Journal of King Saud University-Computer and Information Sciences pp. 481–496 (2019)
148. Sangeeta, L., Neetu, S., Ashish, S.: Improving logging prediction on imbalanced datasets: A case study on open source java projects. In: Cognitive Analytics: Concepts, Methodologies, Tools, and Applications, pp. 740–772 (2020)
149. Sasho, N., Jasmin, B., Alexander, A., Jorge, C., Odej, K.: Self-supervised log parsing. In: Proceedings of the 2021 European Conference on Machine Learning and Knowledge Discovery in Databases, pp. 122–138 (2021)
150. Satoru, K., Kensuke, F., Hiroshi, E.: Towards an nlp-based log template generation algorithm for system log analysis. In: Proceedings of the 2014 International Conference on Future Internet Technologies, pp. 1–4 (2014)
151. Satoru, K., Yuya, Y., Kazuki, O., Kensuke, F.: Amulog: A general log analysis framework for comparison and combination of diverse template generation methods. International Journal of Network Management p. e2195 (2022)
152. Sayalee, N., Tripti, B.: Hmr log analyzer: Analyze web application logs over hadoop mapreduce. International Journal of UbiComp p. 41 (2013)
153. Scott, L., Hironori, W., Nobukazu, Y., Yoshiaki, F.: Online log parsing: Preliminary literature review. In: Proceedings of the 2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 304–305 (2021)
154. Senna, T.P., Julia, L., Gilles, M.: 3l: Learning linux logging. In: Proceedings of the 2015 BElgian-NEtherlands software eVOLution seminar (BENEVOL 2015) (2015)
155. Shaiful, C., S, D.N., Abram, H., M, J.Z.: An exploratory study on assessing the energy impact of logging on android applications. Empirical Software Engineering pp. 1422–1456 (2018)
156. Shams, Z., K, D.A., Ragib, H.: Seclaas: Secure logging-as-a-service for cloud forensics. In: Proceedings of the 2013 ACM SIGSAC symposium on Information, computer and communications security, pp. 219–230 (2013)
157. Shanshan, L., Xu, N., Zhouyang, J., Ji, W., Haochen, H., Teng, W.: Logtracker: Learning log revision behaviors proactively from software evolution history. In: Proceedings of the 2018 Conference on Program Comprehension, pp. 178–188 (2018)
158. Shanshan, L., Xu, N., Zhouyang, J., Xiangke, L., Ji, W., Tao, L.: Guiding log revisions by learning from software evolution history. Empirical Software Engineering pp. 2302–2340 (2020)
159. Shaohan, H., Yi, L., Carol, F., Rong, H., Yining, Z., Hailong, Y., Zhongzhi, L.: Paddy: An event log parsing approach using dynamic dictionary. In: Proceedings of the 2020 IEEE/IFIP Network Operations and Management Symposium, pp. 1–8 (2020)

160. Shaozhi, D., Zhongzhi, L., Shaohan, H., Carol, F., He, W., Hailong, Y., Depei, Q.: Reval: Recommend which variables to log with pre-trained model and graph neural network. IEEE Transactions on Network and Service Management (2022)
161. Shayan, H., Mika, M.: Sialog: Detecting anomalies in software execution logs using the siamese network. Automated Software Engineering p. 61 (2022)
162. Sheng, D., Rinku, G., Marc, S., Eric, P., Franck, C.: Logaider: A tool for mining potential correlations of hpc log events. In: Proceedings of the 2017 IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), pp. 442–451 (2017)
163. Shijie, Z., Gang, W.: Efficient online log parsing with log punctuations signature. Applied Sciences p. 11974 (2021)
164. Shilin, H., Pinjia, H., Zhuangbin, C., Tianyi, Y., Yuxin, S., R, L.M.: A survey on automated log analysis for reliability engineering. ACM computing surveys (CSUR) pp. 1–37 (2021)
165. Shilin, H., Qingwei, L., Jian-Guang, L., Hongyu, Z., R, L.M., Dongmei, Z.: Identifying impactful service system problems via log analysis. In: Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 60–70 (2018)
166. Shilin, H., Xu, Z., Pinjia, H., Yong, X., Liqun, L., Yu, K., Minghua, M., Yining, W., Yingnong, D., Saravanakumar, R., et al.: An empirical study of log analysis at microsoft. In: Proceedings of the 2022 ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 1465–1476 (2022)
167. Shimin, T., Weibin, M., Yimeng, C., Yichen, Z., Ying, L., Chunning, D., Tao, H., Yongpeng, Z., Xiangguang, W., Hao, Y.: Logstamp: Automatic online log parsing based on sequence labelling. ACM SIGMETRICS Performance Evaluation Review pp. 93–98 (2022)
168. Shuqi, C., Shanshan, L., Yong, G., Wei, D., Zhouyang, J., Haochen, H., Qing, L.: Notonlylog: Mining patch-log associations from software evolution history to enhance failure diagnosis capability. In: Proceedings of the 2018 Asia-Pacific Software Engineering Conference (APSEC), pp. 189–198 (2018)
169. Shuting, G., Zheng, L., Wenyan, C., Tao, L.: Event extraction from streaming system logs. In: Proceedings of the 2019 Information Science and Applications (ICISA), pp. 465–474 (2019)
170. Sina, G.: Leveraging code clones and natural language processing for log statement prediction. In: Proceedings of the 2021 IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 1043–1047 (2021)
171. Sina, G., AS, W.P.: Logging statements' prediction based on source code clones. In: Proceedings of the 2020 Annual ACM Symposium on Applied Computing, pp. 82–91 (2020)

172. Staffs, K.: Guidelines for performing systematic literature reviews in software engineering (2007)
173. Stephen, Y., J, P.S., John, O.: Nanolog: A nanosecond scale logging system. In: Proceedings of the 2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18), pp. 335–350 (2018)
174. Steven, L., Heng, L., P., C.T.H., Weiyi, S., Wei, L.: Logassist: Assisting log analysis through log summarization. IEEE Transactions on Software Engineering (2021)
175. Steven, Y., Melody, M.: Intelligent log analysis using machine and deep learning. In: Research Anthology on Artificial Intelligence Applications in Security, pp. 1154–1182 (2021)
176. Suhas, K., Cor-Paul, B., Weiyi, S., D, S.M., E, H.A.: Examining the stability of logging statements. Empirical Software Engineering pp. 290–333 (2018)
177. Suhas, K., Cor-Paul, B., Weiyi, S., E, H.A.: Logging library migrations: A case study for the apache software foundation projects. In: Proceedings of the 2016 International Conference on Mining Software Repositories, pp. 154–164 (2016)
178. Taeyoung, K., Suntae, K., Cheol-Jung, Y., Soohwan, C., Sooyong, P.: An automatic approach to validating log levels in java. In: Proceedings of the 2018 Asia-Pacific Software Engineering Conference (APSEC), pp. 623–627 (2018)
179. Taeyoung, K., Suntae, K., Sooyong, P., YoungBeom, P.: Automatic recommendation to appropriate log levels. Software: Practice and Experience pp. 189–209 (2020)
180. Tal, W., Eric, S., Udi, W.: A sampling-based approach to accelerating queries in log management systems. In: Proceedings of the 2016 ACM SIGPLAN International Conference on Systems, Programming, Languages and Applications: Software for Humanity, pp. 37–38 (2016)
181. Tao, L., Yexi, J., Chunqiu, Z., Bin, X., Zheng, L., Wubai, Z., Xiaolong, Z., Wentao, W., Liang, Z., Jun, W., et al.: Flap: An end-to-end event log analysis platform for system management. In: Proceedings of the 2017 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1547–1556 (2017)
182. Theodoros, K., Kostas, K.: Schema independent reduction of streaming log data. In: Proceedings of the 2014 International Conference on Advanced Information Systems Engineering, pp. 394–408 (2014)
183. Tinnakorn, M., V.C, B., Phond, P.: A hyperparameter tuning approach for an online log parser. In: Proceedings of the 2021 International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), pp. 1036–1040 (2021)
184. Tong, J., Ying, L., Chengbo, Z., Wensheng, X., Jie, J., Yuhong, L.: Machine deserves better logging: a log enhancement approach for automatic fault diagnosis. In: Proceedings of the 2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 106–111 (2018)

185. Tong, X., Zhe, Q., Zhi-Jie, W., Kaiqi, Z., Xiangke, L.: Lpv: A log parser based on vectorization for offline and online log parsing. In: Proceedings of the 2020 IEEE International Conference on Data Mining (ICDM), pp. 1346–1351 (2020)
186. Tse-Hsun, C., W, T.S., Hassan, A.E.: A survey on the use of topic models when mining software repositories. Empirical Software Engineering pp. 1843–1919 (2016)
187. Tsuyoshi, M., Kazumasa, S., Takashi, I., Katsuro, I.: Padla: a dynamic log level adapter using online phase detection. In: Proceedings of the 2019 IEEE/ACM International Conference on Program Comprehension (ICPC), pp. 135–138 (2019)
188. Ulaş, A., Baha, Ş.: Gdpr compliant audit log management system with blockchain. In: Proceedings of the 2021 Turkish National Software Engineering Symposium (UYMS), pp. 1–3 (2021)
189. Vinayak, B., Anuja, T., Chinmaya, P., Anirudha, D., Harmeet, K.: Hadoop in action: Building a generic log analyzing system. In: Proceedings of the 2018 International Conference on Computing Communication Control and Automation (ICCUBEA), pp. 1–7 (2018)
190. Vincent, B., Russell, S., Jacob, C., Mark, D., Tomas, C., Karel, F., Miroslav, B., Pavel, T., Dongwan, S.: On matching log analysis to source code: A systematic mapping study. In: Proceedings of the 2020 International Conference on Research in Adaptive and Convergent Systems, pp. 181–187 (2020)
191. Wei, X., Ling, H., Armando, F., David, P., I, J.M.: Detecting large-scale system problems by mining console logs. In: Proceedings of the 2009 Symposium on Operating Systems Principles, pp. 117–132 (2009)
192. Weibin, M., Federico, Z., Yuheng, H., Ying, L., Shenglin, Z., Yuzhe, Z., Yichen, Z., Tianke, Z., En, W., Zuomin, R., et al.: Summarizing unstructured logs in online services. arXiv preprint arXiv:2012.08938 (2020)
193. Weibin, M., Federico, Z., Yuzhe, Z., Ying, L., Shenglin, Z., Shimin, T., Yichen, Z., Tao, H., Yongpeng, Z., En, W., et al.: Logsummary: Unstructured log summarization for software systems. IEEE Transactions on Network and Service Management (2023)
194. Weibin, M., Ying, L., Federico, Z., Shenglin, Z., Yihao, C., Yuzhe, Z., Yichen, Z., En, W., Ruizhi, Z., Shimin, T., et al.: Logparse: Making log parsing adaptive through word classification. In: Proceedings of the 2020 International Conference on Computer Communications and Networks (ICCCN), pp. 1–9 (2020)
195. Weibin, M., Ying, L., Yichen, Z., Shenglin, Z., Dan, P., Yuqing, L., Yihao, C., Ruizhi, Z., Shimin, T., Pei, S., et al.: Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In: Proceedings of the 2019 International Joint Conference on Artificial Intelligence, pp. 4739–4745 (2019)
196. Weibin, M., Ying, L., Yuheng, H., Shenglin, Z., Federico, Z., Bingjin, C., Dan, P.: A semantic-aware representation framework for online log analysis. In: Proceedings of the 2020 International Conference on Computer

Communications and Networks (ICCCN), pp. 1–7 (2020)
197. Weiyi, S., Meiyappan, N., E, H.A., Ming, J.Z.: Understanding log lines using development knowledge. In: Proceedings of the 2014 IEEE international conference on software maintenance and evolution, pp. 21–30 (2014)
198. William, P., Lei, Z., John, S., Tony, E., Andriy, M.: Immutable log storage as a service on private and public blockchains. IEEE Transactions on Services Computing (2021)
199. Xi, L., Ting, W., Shexiong, W.: Pattern-based deep learning method to extract information from the log dataset. Journal of Circuits, Systems and Computers p. 2150296 (2021)
200. Xia, N., Geoff, J., Haifeng, C., Kenji, Y.: Hlaer: A system for heterogeneous log analysis. In: Proceedings of the 2014 SDM Workshop on Heterogeneous Learning, p. 1 (2014)
201. Xiaotong, L., Tong, J., Ying, L., Hao, Y., Yang, Y., Chuanjia, H.: Automatically generating descriptive texts in logging statements: How far are we? In: Proceedings of the 2020 Programming Languages and Systems: Asian Symposium, pp. 251–269 (2020)
202. Xiaoyu, D., Shi, Y., Hailong, C., Wanli, Y., Xiang, Y.: Oilog: An online incremental log keyword extraction approach based on mdp-lstm neural network. Information Systems p. 101618 (2021)
203. Xinpeng, L., Yong, W., Hao, F., Wenlong, K.: A parallel host log analysis approach based on spark. In: Proceedings of the 2018 International Conference on Computational Intelligence and Security (CIS), pp. 301–305 (2018)
204. Xiuqin, L., Peng, W., Bin, W.: Log analysis in cloud computing environment with hadoop and spark. In: Proceedings of the 2013 IEEE International Conference on Broadband Network & Multimedia Technology, pp. 273–276 (2013)
205. Xu, N., Shanshan, L., Zhouyang, J., Shulin, Z., Wang, L., Xiangke, L.: Understanding the similarity of log revision behaviors in open source software. International Journal of Performability Engineering p. 1887 (2018)
206. Xu, Z., Kirk, R., Yu, L., Michael, S., Ding, Y., Yuanyuan, Z.: The game of twenty questions: Do you know where to log? In: Proceedings of the 2017 Workshop on Hot Topics in Operating Systems, pp. 125–131 (2017)
207. Xu, Z., Kirk, R., Yu, L., Michael, S., Ding, Y., Yuanyuan, Z.: Log20: Fully automated optimal placement of log printing statements under specified overhead threshold. In: Proceedings of the 2017 Symposium on Operating Systems Principles, pp. 565–581 (2017)
208. Xueshuo, X., Zhi, W., Xuhang, X., Ye, L., Shenwei, H., Tao, L.: A confidence-guided evaluation for log parsers inner quality. Mobile Networks and Applications pp. 1638–1649 (2021)
209. Ya, Z., Yuanbo, G., Chunhui, L.: Flp: a feature-based method for log parsing. Electronics letters pp. 1334–1336 (2018)

210. Yi, Z., Jinfu, C., Weiyi, S., Tse-Hsun, C.: Studying the characteristics of logging practices in mobile apps: A case study on f-droid. Empirical Software Engineering pp. 3394–3434 (2019)
211. Yiming, T., Allan, S., Raffi, K., Mehdi, B.: Automated evolution of feature logging statement levels using git histories and degree of interest. Science of Computer Programming p. 102724 (2022)
212. Yiming, T., Allan, S., Raffi, K., Mehdi, B.: A tool for rejuvenating feature logging levels via git histories and degree of interest. In: Proceedings of the 2022 ACM/IEEE International Conference on Software Engineering: Companion Proceedings, pp. 21–25 (2022)
213. Ying, F., Meng, Y., Jian, X., Jianguo, L., Zhongxin, L., Xiaohong, Z., Dan, Y.: Investigating and improving log parsing in practice. In: Proceedings of the 2022 ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 1566–1577 (2022)
214. Ying, F., Meng, Y., Zhou, X., Xin, X., Xiaohong, Z., Dan, Y.: An empirical study of the impact of log parsers on the performance of log-based anomaly detection. Empirical Software Engineering pp. 1–39 (2023)
215. Yining, Z., Xiaodong, W., Haili, X., Xuebin, C.: Improvement of the log pattern extracting algorithm using text similarity. In: Proceedings of the 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 507–514 (2018)
216. Yintong, H., Yuxin, S., Michael, L.: Logvm: Variable semantics miner for log messages. In: Proceedings of the 2022 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 124–125 (2022)
217. Yu, B., Yongwei, C., Dan, Z.: Patcluster: A top-down log parsing method based on frequent words. IEEE Access pp. 8275–8282 (2023)
218. Yu-Qian, Z., Jia-Ying, D., Jia-Chen, P., Peng, W., Shen, L., Wei, W.: Mlparser: An efficient and accurate online log parser. Journal of Computer Science and Technology pp. 1412–1426 (2022)
219. Yudong, L., Xu, Z., Shilin, H., Hongyu, Z., Liqun, L., Yu, K., Yong, X., Minghua, M., Qingwei, L., Yingnong, D., et al.: Uniparser: A unified log parser for heterogeneous log data. In: Proceedings of the 2022 ACM Web Conference, pp. 1893–1901 (2022)
220. Yun, L., Yongyao, J., Juan, G., Mingyue, L., Manzhu, Y., M, A.E., Thomas, H., David, M., J, M.L., Greguska, F., et al.: A cloud-based framework for large-scale log mining through apache spark and elasticsearch. Applied Sciences p. 1114 (2019)
221. Yun, W., Qianhuizhi, Z.: A logging overhead optimization method based on anomaly detection model. In: Proceedings of the 2021 Human Centered Computing International Conference, pp. 349–359 (2021)
222. Yusufu, S., Robert, H.: Enhancements to language modeling techniques for adaptable log message classification. IEEE Transactions on Network and Service Management (2022)

223. Yuxia, X., Kai, Y., Pan, L.: Logm: Log analysis for multiple components of hadoop platform. IEEE Transactions on Software Engineering pp. 73522–73532 (2021)
224. Zhang, C., Xiaojing, M.: Log parser with one-to-one markup. In: Proceedings of the 2020 International Conference on Information and Computer Technologies (ICICT), pp. 251–257 (2020)
225. Zhenhao, L., Heng, L., Tse-Hsun, C., Weiyi, S.: Deeplv: Suggesting log levels using ordinal based neural networks. In: Proceedings of the 2021 IEEE/ACM International Conference on Software Engineering (ICSE), pp. 1461–1472 (2021)
226. Zhenhao, L., Tse-Hsun, C., Jinqiu, Y., Weiyi, S.: Dlfinder: Characterizing and detecting duplicate logging code smells. In: Proceedings of the 2019 IEEE/ACM International Conference on Software Engineering (ICSE), pp. 877–887 (2016)
227. Zhenhao, L., Tse-Hsun, C., Jinqiu, Y., Weiyi, S.: Studying duplicate logging statements and their relationships with code clones. Journal of Systems and Software pp. 2476–2494 (2021)
228. Zhenhao, L., Tse-Hsun, C., Weiyi, S.: Where shall we log? studying and suggesting logging locations in code blocks. In: Proceedings of the 2020 IEEE/ACM International Conference on Automated Software Engineering, pp. 361–372 (2020)
229. Zhiyuan, Z., Chenxu, W., Wei, R.: Slop: Towards an efficient and universal streaming log parser. In: Proceedings of the 2018 International Conference on Information and Communications Security, pp. 325–341 (2018)
230. Zhongxin, L., Xin, X., David, L., Zhenchang, X., E, H.A., Shanping, L.: Which variables should i log? IEEE Transactions on Software Engineering pp. 2012–2031 (2019)
231. Zhouyang, J., Shanshan, L., Xiaodong, L., Xiangke, L., Yunhuai, L.: Smartlog: Place error log statement by deep understanding of log intention. In: Proceedings of the 2018 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 61–71 (2018)
232. Zishuo, D., Heng, L., Weiyi, S.: Logentext: Automatically generating logging texts using neural machine translation. In: Proceedings of the 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 349–360 (2022)
233. Zuo, Y., Zhu, X., Qin, J., Yao, W.: Temporal relations extraction and analysis of log events for micro-service framework. In: Proceedings of the 2021 Chinese Control Conference (CCC), pp. 3391–3396 (2021)