An Empirical Study of Log Analysis at Microsoft

Shilin He
Xu Zhang
Microsoft Research
Beijing, China

Pinjia He
The Chinese University of Hong Kong, Shenzhen
Shenzhen, China

Yong Xu
Liqun Li
Microsoft Research
Beijing, China

Yu Kang
Minghua Ma
Microsoft Research
Beijing, China

Yining Wei
Microsoft Azure
Shanghai, China

Yingnong Dang
Microsoft Azure
Redmond, USA

Saravanakumar Rajmohan
Microsoft 365
Redmond, USA

Qingwei Lin*
Microsoft Research
Beijing, China

ABSTRACT

Logs are crucial to the management and maintenance of software systems. In recent years, log analysis research has achieved notable progress on various topics such as log parsing and log-based anomaly detection. However, the real voices from front-line practitioners are seldom heard. For example, what are the pain points of log analysis in practice? In this work, we conduct a comprehensive survey study on log analysis at Microsoft. We collected feedback from 105 employees through a questionnaire of 13 questions and individual interviews with 12 employees. We summarize the format, scenario, method, tool, and pain points of log analysis. Additionally, by comparing the industrial practices with academic research, we discuss the gaps between academia and industry, and future opportunities on log analysis with four inspiring findings. Particularly, we observe a huge gap exists between log anomaly detection research and failure alerting practices regarding the goal, technique, efficiency, *etc.* Moreover, data-driven log parsing, which has been widely studied in recent research, can be alternatively achieved by simply logging template IDs during software development. We hope this paper could uncover the real needs of industrial practitioners and the unnoticed yet significant gap between industry and academia, and inspire interesting future directions that converge efforts from both sides.

CCS CONCEPTS

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

KEYWORDS

Log Analysis, Empirical Study, Software Reliability

ACM Reference Format:
Shilin He, Xu Zhang, Pinjia He, Yong Xu, Liqun Li, Yu Kang, Minghua Ma, Yining Wei, Yingnong Dang, Saravanakumar Rajmohan, Qingwei Lin. 2022. An Empirical Study of Log Analysis at Microsoft. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22), November 14–18, 2022, Singapore, Singapore.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3540250.3558963

*Qingwei Lin is the corresponding author of this work.

ESEC/FSE '22, November 14–18, 2022, Singapore, Singapore

ACM ISBN 978-1-4503-9413-0/22/11…$15.00
https://doi.org/10.1145/3540250.3558963

1 INTRODUCTION

Logs, as a well-acknowledged cornerstone for system observability [36, 38], record important runtime information. In industry, logs have been widely applied in the management and maintenance of software systems to ensure their reliability and availability, especially in large-scale systems. Moreover, log plays a vital role in security and auditing [13], workflow analysis [20], and making business decisions [3]. Nowadays, most software companies have strong demands and have devoted many efforts to managing logs, which also fuels the rise of the log management market with companies such as Splunk [35] and Datadog [10].

Considering the huge volume and great complexity of log data in large-scale systems, automated log analysis research aims to relieve the unaffordable manual efforts of log inspection with more intelligent approaches. Throughout the years, automated log analysis has also attracted a lot of attention from the research community [13, 39, 44, 49], especially with the popularization of machine learning and deep learning. As reviewed by He et al. [16], over 100 log research studies were presented in top conferences or journals from 2015 to 2020, where topics include logging [21], log compression [25], log parsing [49], anomaly detection [39], failure prediction[9], diagnosis [17, 45], *etc.*

Although log analysis has been extensively studied with growing prosperity, the real voices from front-line practitioners are seldom heard. For example, what kind of logs do they analyze? What tools do they currently use and expect to use in the future? What are the pain points during log analysis? *etc.* With these questions in mind, we conducted a comprehensive survey study on log analysis by empirically inquiring and interviewing employees at Microsoft, one of the largest software companies in the world. Additionally, based on our years of both industrial and academic experience, several follow-up questions were raised: What are the current practices of log analysis in the industry? Do these practices differ from academic research? What are the opportunities?

In this work, we first collected feedback from 105 employees in Microsoft through a questionnaire of 13 questions. We followed the guideline [31] and recent practices [1, 8] to design and distribute the survey. The survey participants come from a variety of products

in Microsoft, including complex cloud systems like Azure, productivity applications like Microsoft 365 (M365), operating systems like Windows, search engines like Bing, social media like LinkedIn, *etc.* Moreover, the participants are of different roles, such as software developer and site reliability engineer. To unearth more details and hidden insights behind the survey responses, we further invited 12 participants to an individual interview. It took around four months to collect the feedback through the questionnaire and interview.

We organized and summarized the survey responses by the log format, log usage scenario, analysis method and tool, and challenges of log analysis. These results uncover the first-hand experience of log practitioners in real-world production environments. For example, engineers in Microsoft mostly use structured and semi-structured logs instead of unstructured logs. Moreover, we present six practical challenges worth further exploring, for example, *too many logs and missing logs*, *hard to find failure-related logs* and *correlating logs from various sources is challenging*. These results provide an industrial view of what engineers care about most in log analysis, which may provide insights for future research studies.

We then present a brief review of log analysis research [16] in recent years, which consists of logging, log parsing, anomaly detection, and failure diagnosis. For each topic, we discussed the industrial practices, presented the gap between academic and industrial studies, and shared the opportunities for future research. We further concluded four inspiring findings from the discussion, for instance, "*log parsing is generally achieved by lightweight instrumentation, while the after-the-fact data-driven log parsing are used in some restricted scenarios such as third-party libraries* and "*a huge gap exists between anomaly detection in academia and failure alerting in the industry regarding the goal, techniques, interpretability, etc.*"

In recent years, there have been a few empirical survey studies [3, 40, 47] on log analysis in the industry. Barik et al. [3] presented a study on logging to understand the organizational implications and challenges of various roles at Microsoft. Yang et al. [40] studied how developers analyze logs in embedded software engineering via interviews. Unlike them, we focus on the current practices, challenges, gaps and opportunities of log analysis in a large software company. Additionally, Zhao et al. [47] conducted an empirical study on log anomaly detection, while our work considers the broader log analysis (such as logging) life cycle.

We summarize our main contributions as follows:

- We present a comprehensive survey study on log analysis at Microsoft and summarize the survey responses from perspectives including format, scenario, pain points, *etc.*
- To our best knowledge, we are the first to summarize the gaps between academia and industry and present some promising opportunities in log analysis domain.

We hope the experience of front-line engineers can help researchers better understand the industrial needs. Besides, we hope the unnoticed yet significant gap between industry and academia can inspire interesting future directions that converge both sides efforts and benefit the log analysis community.

2 BACKGROUND

In this section, we brief the background of logs, i.e., typical log formats and how logs are usually managed in the industry.

Structured Log

Timestamp	Cluster	Node	RackId	API	Version
2021-01-01 02:51:26	C01	N01	2_17	GetMessage	**v1**
2021-01-01 02:51:28	C01	N03	2_18	GetMessage	**v2**

Unstructured Log

Message
2021-01-01 02:51:26 COM Receive request 3754984 through GetMessage
2021-01-01 02:51:28 COM Executing the request takes 0.47356277 seconds

Semi-Structured Log

Timestamp	Cluster	Node	Message
2021-01-01 02:51:26	C01	N01	{"Version": "v1", "Message": Exception when processing request 89953}
2021-01-01 02:51:26	C01	N03	{"Version": "v2", "Message": Exception when processing request 46352}

Figure 1: Samples of structured, unstructured and semi-structured logs.

2.1 Log and Its Format

Recent years have witnessed the increasing scale and complexity of software systems, which leads to their more powerful functions and broader adoption while, at the same time, inevitably making them hard to manage and maintain. To tackle the challenge, engineers resort to enhancing the system observability [36, 38] with logs. Logs capture various system run-time information such as events, transactions, and messages. A typical piece of log is an *immutable, time-stamped record* that happened over time (e.g., software update events or received messages). Logs are usually generated when a system executes the corresponding logging code snippets. A system with mature logs essentially facilitates the system behavior understanding, health monitoring, failure diagnosis, *etc.*

Generally, there are three typical log formats, i.e., structured, semi-structured and unstructured logs. These logs share the same components: a timestamp and a payload content. We present an example of each log format in Figure 1 with details as follows:

(1) *Unstructured Log* is a common log type written in free-form plain text. It is human-readable and easy-to-instrument.
(2) *Structured Log* is generated by following a predetermined log format (e.g., key-value pairs). Structured logs are machine-friendly due to the ease of storage and statistical analyses.
(3) *Semi-structured Log* demonstrates a mixture of both structured and unstructured characteristics.

Structured logs usually keep a consistent format within the log data and are easy to manage. Specifically, the well-structured format allows easy storing, indexing, searching and aggregation in a relational database. However, the structured format falls short in recording heterogeneous but correlated events in a single file or database. It is because one predetermined log format may not fit all log events from various system components. An alternative is to use unstructured (or semi-structured logs), whose free-form format allows the ingestion of different log events to the same place. However, the unstructured log data achieves its high flexibility at the expense of the ease of machine processing. The characteristic of free-form text becomes a major obstacle for efficient query and analysis on unstructured or semi-structured logs. For instance, to count how often an API version appears in unstructured logs, engineers need to design a complex query with ad-hoc regular expressions to extract the desired information. Clearly, the manual process takes lots of time and effort and is not scalable.

2.2 Log Management System

This section briefly introduces how a log management system is commonly designed and implemented in practice, which is seldom covered in the literature. Usually, logs are managed through several phases: collection, ingestion, pre-processing, and consumption.

Log collection. Logs are collected from various sources, including the software application, infrastructure, host OS, *etc.* To enable the emission of the log data, engineers write logging statements during development by leveraging various instrumentation frameworks, usually in the form of SDK (e.g., log4j) or API. The framework is designed to run alongside the program being monitored and gather the log data automatically.

Log ingestion. The gathered data is then sent to intermediate storage outside the program. An agent is deployed to listen on the storage and send the collected data to an ingestion component. The logs are then persisted into durable storage systems for indexing and querying. The storage systems usually adopt cost-efficient and reliable storage solutions (e.g., distributed storage with replication) and may vary by the log format and usage. For instance, the log data for monitoring usually requires low latency, and the storage should allow rapid ingestion and query of log data.

Log pre-processing. Predefined pre-processing tasks are triggered after the log data arrives in storage, including content extraction, removing personal-identifiable information, reformatting, merging multiple data sources, *etc.* Due to the enormous volume of log data in Microsoft, pre-processing is often conducted on a Hadoop-like big data platform. Since there are many downstream log related workloads, these pre-processing tasks usually require sophisticated scheduling to accomplish specific SLAs, e.g., within the time limit.

Log consumption. Logs are consumed in various downstream scenarios, e.g., monitoring and diagnosis. For monitoring, a prompt alerting service is set up to detect anomalies based on near-real-time log data. Once detected, the alerting service will escalate the anomaly as an alert and notify On-Call engineers to investigate. Besides the monitoring, log data are also queried for on-demand diagnosis purposes; for example, log queries like keyword search or time-range filtering are often performed when a failure happens.

3 SURVEY METHOD

To investigate how logs are analyzed in practice, we surveyed 105 employees from different organizations in Microsoft. These organizations cover very diverse products, consisting of cloud systems with thousands of services like *Azure*, productivity applications like *M365*, operation systems like *Windows*, search engines like *Bing*, social media like *LinkedIn*, *etc.* These products represent a majority of software types in today's software market. To systematically conduct the survey, we follow the guideline proposed by Punter et al. [31] and survey practices presented in recent studies [1, 8].

Survey Design & Delivery. Unlike the recent study [40] which only took one-to-one interviews, we used a hybrid methodology to collect feedback, including a survey questionnaire containing 13 questions and a follow-up one-hour interview. We used *Microsoft Forms*[29] to design our survey questionnaire, collect and analyze feedback. We sent the questionnaire link to our participants via email. The invitation email first illustrates the purpose of the survey, followed by asking the email recipient to take the survey via clicking the link. The survey contains three sections: 1) The introduction shows a brief overview of the survey, an estimated completion time (10 minutes) and a consent form for ethics reasons. 2) The survey questions (Q1 - Q11 in Table 1) on log analysis. 3) Two additional questions (Q12 - Q13, not listed in Table 1): The Q12 is about whether the participant is willing to attend the follow-up interview, and its answers are *Yes*, *No* and *Maybe*. The Q13 asks for participants' suggestions to improve the survey question design. All questions are not optional except for Q13.

For the follow-up one-hour interview, we invited those participants who chose the answers of *Yes* and *Maybe* to attend a meeting via an internal communication tool. Considering the time zone difference, we scheduled each meeting after several rounds of email interactions. In the interview, we asked the interviewee to provide additional context to their response and made an in-depth discussion on it. The entire interview process is conducted interactively with questions based on the interviewee's response. We recorded the entire interview (e.g., screen sharing, video, auto-script), then replayed and analyzed the recorded video post the interview.

Table 1: A list of survey questions (excluding Q12 and Q13).

	ID	Question
Role	Q1	What is your profession (e.g., software engineer, data scientist)?
	Q2	What is your year of experience in analyzing logs?
Usage	Q3	Which log format do you analyze most frequently?
	Q4	Which scenarios do you usually use logs for?
	Q5	How many hours do you spend analyzing logs in an incident?
Tool	Q6	How do you usually analyze logs?
	Q7	What existing tools do you use to analyze logs?
	Q8	What future tools do you expect to utilize to analyze logs?
Challenges	Q9	How often do you analyze logs that you are unfamiliar with?
	Q10	Failure-related logs may be overwhelmed by too many logs and hard to identify. To what extent do you agree or disagree?
	Q11	What are the pain points/challenges when analyzing logs?

Survey Questions Design. To start, we designed the first version of survey questions based on our years of industrial experience and observations. We then performed a pilot study to evaluate the survey questions by inviting 10 experienced developers to provide feedback. After receiving the feedback, we revise the wording of both questions and answers (if the question is closed-ended). For questions that are not easy to understand, we provided illustrating examples to demonstrate the concepts. For example, in Q3, we attached the example in Figure 1 under the question. After this process, we finalized the survey questions.

These questions include the role of the survey participants (Q1, Q2), the log analysis usage scenario (Q3 - Q6), the toolsets (Q6 - Q8) and challenges of analyzing logs in practice (Q9 - Q11), and the two additional questions (Q12, Q13) mentioned above. Among these questions, Q1 - Q5 and Q9 - Q10 are closed-ended questions whose answers are concluded based on our domain experience and feedback from pilot participants. For the closed-ended questions, to ensure the comprehensiveness of the answers, we provide an additional option named "Others". Participants can enter a free-text answer if their answer is not listed in the predefined answers. Q4

is a question with multiple-choice answers. Apart from these questions, the remaining questions are open-ended, and participants can enter a free text. It is worth noting that Q10 was designed after we collected some feedback from the pilot participants. We realized that Q10 might be a common question for most engineers and would like to know to what extent they agree on it.

Recruiting Participants. We recruited the survey participants in various ways after the pilot run. Firstly, we contacted several team leaders from different organizations at Microsoft with a systematic sampling approach [31]. We then asked them to distribute the survey link to their team members. Secondly, we adopt the self-recruited invitation, a type of non-systematic sampling approach [31], to recruit participants that might be interested in the survey topic. In detail, we posted the survey invitation to several internal communities (i.e., an interest group for employees from different organizations and teams to discuss a specific topic, such as telemetry, reliability engineering, AIOps) that contains hundreds or thousands of employees. The advantage is that those people are attracted by similar topics as the survey and may be eager to respond. At last, we adopted the snowball sampling method, where we asked the participants to forward the invitation email to their colleagues. In total, we sent more than 2,000 survey links and received 105 survey responses from various organizations including *Azure*, *M365*, *Windows*, *Bing*, *LinkedIn*, *etc.* For the follow-up interview, 12 participants attended the one-hour interview meeting.

Organizing Survey Results. We statistically summarized the answers for closed-ended questions by counting how many participants chose each answer (e.g., Table 2) or plotting the distribution (e.g., Figure 3) if it is not a multi-choice question. For open-ended questions, we adopted the widely-adopted open coding techniques [6] to summarize the results. For each question, the first two authors independently compiled a list of answer categories in two steps: first, each author iterated the responses and assigned rough labels; second, each author refined the labels by going through the responses again. Then, the two authors discussed their category lists, compiled the final list of answer categories and jointly relabeled each response. At last, for each question, the third and fourth authors rechecked the labels to ensure the correctness. We present these results by either showing the counts (e.g., Table 3) or statistical distributions. For the one-hour interview, the first two authors separately compiled the scripts for each recorded video and converged the scripts together with the fourth author.

Ethics. Before sending out the survey, we followed a strict compliance assessment process at Microsoft to ensure the privacy and security of the collected data. Additionally, similar to a recent study [8], we provided a consent form outlining the study purpose, scope, data usage, and retention policies to all the survey participants at the beginning of the survey questionnaire.

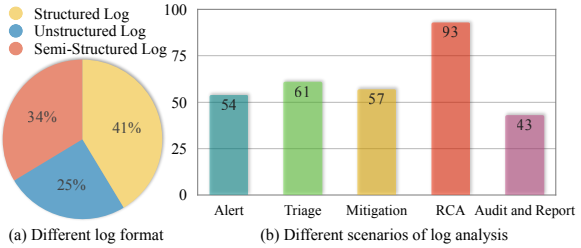
4 SURVEY RESULTS

This section first shows the distribution of survey participants (Q1-Q2) and the survey results in Section 4.1 (Q3-Q10). Then, Section 4.2 summarizes the pain points and challenges in log analysis (Q11). For Q12-Q13, we mainly used the answers to either ask for the follow-up interview or improve the question design, thereby not explicitly presenting the results.

Table 2: Role (left) and year of experience (right) for survey participants (SRE denotes site reliability engineer)

Role	#Resp
Support Engineer & SRE	37
Software Developer	46
Data Scientist	12
Program Manager	5
Others	5

Year	#Resp
1 – 3 years	28
3 – 5 years	21
5 – 7 years	13
> 7 years	43



(a) Different log format (b) Different scenarios of log analysis

Figure 2: Distribution of log formats and analysis scenarios.

Q1 & Q2: Role and experience of participants. As shown in Table 2 (left), the survey participants serve a variety of roles at Microsoft, including support engineer, site reliability engineer (SRE), software developers, data scientist, program manager, and others (e.g., consultant). Support engineers process customer support requests, while SREs mainly resolve live site incidents, and they work with logs frequently. Software developers mostly analyze logs during the on-call. Support engineers, SREs, and software developers account for a majority of our survey participants (around 79%). Data scientists, program managers and others also analyze logs in their scenarios, such as extracting diagnosis insights from logs and tracking the feature growth. Table 2 (right) presents the log analysis experience of engineers. About 73% of engineers have more than 3 years of experience in analyzing logs, and about 42% of them are very experienced engineers (i.e., with more than 7 years of experience). To conclude, the participants cover diverse roles in Microsoft, and most of them are experienced in log analysis.

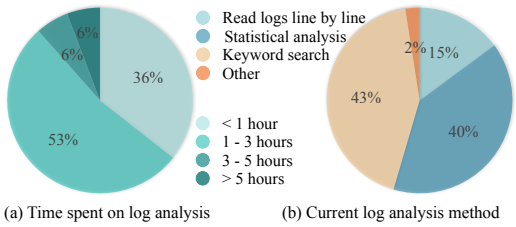
4.1 General Findings

We present the results of questions as follows, including log usage (Q3-Q5), log tool support (Q6-Q8), and log challenges (Q9-Q10).

4.1.1 *Log Usage.*

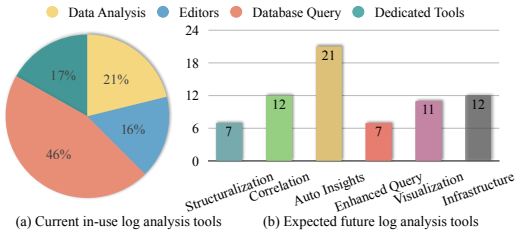
Q3: Log format. As introduced in Section 2.1 and presented in Figure 1, we identified three log formats, i.e., structured, semi-structured and unstructured logs. The distribution of log format usage is presented in Figure 2 (a). Structured and semi-structured logs are the most common (41% and 34% respectively) log types used by the participants, while unstructured logs (25%) are less used. Note that we only asked for the "most frequent" log format, while engineers may process logs of all three formats in practice.

Q4: Log analysis scenario. Engineers usually analyze logs for various purposes in different scenarios, such as software maintenance, auditing and reporting. To facilitate software maintenance,



(a) Time spent on log analysis (b) Current log analysis method

Figure 3: Distribution of log analysis time consumption and current log analysis methods.



(a) Current in-use log analysis tools (b) Expected future log analysis tools

Figure 4: Distribution of current in-use and expected future log analysis tools.

in Microsoft, we adopt a multi-phase procedure to process failures (often referred to as incidents [5]), consisting of alert, triage, mitigation and root cause analysis phases. In the *alert phase*, an alert would be triggered if there is an anomaly pattern in the time series of monitoring metrics. The alert then becomes an incident ticket that notifies On-Call engineers to engage immediately. In the *triage phase*, by inspecting logs or other related information, if engineers in team A find team B's service causes the incident of team A's service, they will transfer it to team B. In the *mitigation phase*, engineers have to quickly stop the incident from impacting customers by taking actions like the reboot. Although engineers mainly mitigate the incident by domain expertise, sometimes they need to analyze logs to decide the mitigation action. In the *root cause analysis (RCA) phase*, engineers will investigate the reasons behind the incident by inspecting logs and other information such as core dump and source code. Once the root cause is identified, engineers from the responsible team can fix the problem.

In terms of log analysis scenarios defined above, Figure 2 (b) illustrates the number of participants that chose each scenario. Note that the participants are allowed to select multiple scenarios simultaneously. It aligns with our expectation that RCA is the most common scenario of log analysis, as confirmed by a remarkably high number (93). Around half of the participants analyzed logs in alert, triage and mitigation scenarios with comparable numbers (50 ~ 60). Logs are also widely leveraged in *audit and report*. In *audit*, admins or auditors use logs to examine suspicious activities to assure security and prove compliance. In *report*, log data supports making business decisions such as product growth and feature development. In our study, around 40% of the participants selected audit and report as their log analysis scenario, demonstrating the strong potential in these areas.

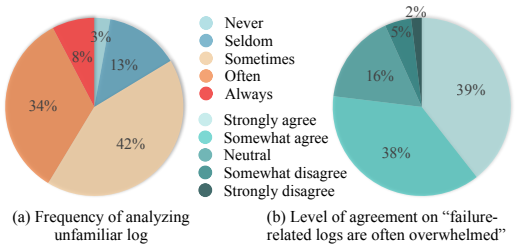
Q5: Log analysis time. The time spent analyzing logs reflects the difficulty of existing log analysis and potentially indicates the space for improvement if we adopt some automation tools. As Figure 3 (a) depicts, about 65% of participants spend more than an hour on log analysis for an incident. Note that most participants are very experienced engineers (shown in Section 3). Hence, it is a heavy workload to analyze logs, which desires more automated log analysis tools to relieve the manual efforts.

4.1.2 *Log Tool Support.*

Q6: Log analysis method. Figure 3 (b) presents the survey results regarding how engineers currently analyze logs. We observed that a large number of engineers (43%) search keywords to find related logs. Many participants (40%) use statistical analysis (e.g., filter, count, sum, trend) to understand logs. In essence, both keyword search and statistical analysis summarize key insights from a vast volume of logs. Nevertheless, a small portion of (15%) participants read logs line by line to understand log content. Since some participants analyze logs by combining two or three of the mentioned methods (such as "*statistical analysis on top of keyword search*"), we add the count to each method separately. Also, a participant analyzes logs by "*typical ML tools*" and another one did not share the specific method as it "*depends on the log file and problem symptom*" (denoted in "Other"). We can find that current-in-use log analysis methods are relatively simple, leaving spaces for more advanced yet easy-to-deploy tools to improve the analysis efficiency.

Q7: Log analysis tools. When applying the above-mentioned methods to analyze logs, participants usually leverage some tools. As depicted in Figure 4 (a), we categorized these tools into different types, including data analysis tools, editors, database queries, and dedicated tools. Around half of the participants analyze logs by writing various queries in our internal database products, including relational databases such as Azure Data Explorer and big data platforms (namely, a Hadoop-like platform) for log data of a large volume, *etc.* Additionally, some participants use data analysis tools such as Python and Spark scripts and other internal/external data analytics tools. The two remaining categories are text editors and dedicated tools. For the former, engineers use popular IDEs such as VS Code and Excel to load logs; for the latter, some dedicated tools are often used. For example, Windows Event Viewer and Process Monitor are often used to analyze Windows logs.

Q8: Future log analysis tools. Although plenty of log analysis tools are already in use, participants express their expectations on future tools. The summarized results are presented in Figure 4 (b). Most participants expected tools that could *automatically* help them analyze logs. For example, it would be beneficial to have a "super AI" that provides "*automated insights*" based on the given logs. They also want the tool to "*point out the failure indicating logs*" with interpretation and "*highlight log importance*". In addition, tools are expected to merge logs from different sources into one place for exploration. Participants also want a good visualization tool to better "*understand logs and the behind workflow*". To retrieve logs from databases, a tool that can automatically generate the query would be useful. Moreover, engineers often need to structurize the unstructured log data into a structured format, demanding an *off-the-shelf* tool to achieve so automatically. There are also some



(a) Frequency of analyzing unfamiliar log

(b) Level of agreement on "failure-related logs are often overwhelmed"

Figure 5: Frequency of analyzing unfamiliar logs (a) and level of agreement on overwhelmed failure logs (b).

expectations on the overall log infrastructure (e.g., collaborative log data exploration and better log organization). To summarize, participants have great expectations for the future log analysis tools, which we hope could attract more attention from the research community to contribute more.

4.1.3 Log Challenges. Here we only present the result of Q9-Q10 and leave the results of Q11 and the interview in Section 4.2.

Q9: Analyzing unfamiliar logs. Intuitively, it is easy to analyze logs for an engineer if she is the author of the corresponding logging statements. However, the industry embraces cross-team collaborations, and a practical challenge is dealing with unfamiliar logs (e.g., from other teams or components). Figure 5 (a) shows how frequent the participant processes unfamiliar logs. It is observed that a large portion (i.e., 34% of "often" and 8% of "always") of participants frequently work on unfamiliar logs, while only 16% of participants seldom or even never work on unfamiliar logs. Interestingly, we found that SRE and support engineers are more likely to analyze unfamiliar logs than software developers. 92% of SRE and support engineers chose the frequency higher than "often", while 74% of software engineers chose the same. The reason is that developers are more likely to investigate incidents related to them or their team. To summarize, it is worth exploring how to ease the burden of understanding unfamiliar logs for engineers.

Q10: Overwhelming logs. Through our pilot study, we found many participants complaining about dealing with overwhelming logs (i.e., hard to distinguish failure-related logs from normal logs). To quantitatively measure how serious the issue is, we asked to what extent the participant would agree or disagree with the complaint. As presented in Figure 5 (b), around 77% of participants hold a positive view for the question, while only 7% of participants do not think so. Intuitively, the log size is tremendous for industry-scale services, even after we scope down the period for investigation. How to shrink down the significant volume of logs to a reasonable size would be a challenging but valuable topic to explore.

4.2 Real-World Pain Points and Challenges

In this section, we present the pain points and challenges that the participants have during log analysis. As presented in Table 3, we list the most-voted and interesting pain points collected from our survey participants. The pain points and challenges of free-text are grouped in various categories by the systematic result organization method we introduced in Section 3. Note that a participant may mention multiple pain points, we classify each pain point to the best-fit category for easy summarization.

Table 3: Pain points that participants encountered when analyzing logs (#Par shows the number of participants).

Too many logs and missing logs	**#Par**
Too many logs	15
Missing Logs	18
Log file is too large to open/join	3
Logs are hard to structurize	
Logs are unformatted	4
Inefficient on extracting data from free-form fields	3
Cannot summarize or count for some log	2
Logs are difficult to understand	
Hard to understand what log means	5
Do not understand log column name	5
Do not understand the logic and normal pattern behind logs	4
Lack of a good visualization tool	4
Finding failure-related logs is non-trivial	
Hard to find failure-related logs	15
Logs are noisy	6
Difficult to compare "bad" logs with "good" logs	4
Correlating logs from various sources is challenging	
Difficulty in correlating multiple log sources	17
Terminology differences make cross-team logs not align	14
Logs of various origins, sizes, formats and patterns	4
Others	
Access control	7
Automated query generation for log analysis	3
Enrich incidents with automated log query/search result	2

1) **"Too many logs" and "missing logs"**: The huge volume and incompleteness of logs have been two long-standing problems that bother engineers [40] and motivate many research studies [24, 39, 43, 46]. The log data is often too large to understand completely: "*there is too much information to read, and it is often hard to find the key point*" and "*do not read logs as too many logs for an incident*". On the other hand, the log data sometimes do not provide sufficient information: "*most useful logs are not enabled by default*" and "*the information from the given log file is limited*". Another vivid example is: "*we try to reconstruct why a failure happened through logs, which takes a lot of time and may sometimes be impossible if there is inadequate logging in the intermediate code paths.*" Although "too many logs" and "missing logs" look like a paradox, they are deeply tangled for complex software systems. Intuitively, more logs provide more detailed information about a failure. However, it may hurt the system operating efficiency and considerably lengthen the time for diagnosis, making it like finding the needle in a haystack.

2) **Logs are hard to structurize**: If all logs are well-structured, it is straightforward to gain insights by setting filters or statistical analysis (e.g., count and sum). However, as presented in Section 2 and Figure 2 (a), a large amount of logs are still unstructured. These unstructured logs are hard for direct analysis due to the free-form text representation. For instance, it is reported that "*difficult to code around the various log formats*", "*log data is poorly formatted, and summarization is hard because of unique identifiers in log messages, but dropping those require lots of work*".

3) **Logs are difficult to understand**: Logs are initially designed for human interpretation with natural language-like messages.

However, log understanding is not as easy as reading natural language statements due to multiple challenges: log message readiness, understanding the logic behind logs and log field meaning, and the lack of tools for understanding. For example, it was reported that "*not familiar with how things work due to lack of readiness*" and "*do not understand which line means what*". A common way to analyze logs is to learn the execution logic behind logs, which is non-trivial with many interleaving logs from various sources and components. An example is as follows: "*for me, a lot of time is spent on figuring the software logic behind the log records, and then, eventually, understand why it behaves in a certain way.*" Sometimes, logs are not sufficient for developers to form a good understanding, requiring additional digging on source code. For instance, "*understanding the context in which the log is being recorded, usually requires looking through code*". Moreover, for structured logs or semi-structured logs, the log field (column) name often indicates the physical meaning of that column. Since the column name is usually short in length, understanding its meaning is a challenge: "*log column is not easily understandable*" and "*understand the significance of different fields*". Several participants also pointed out that "*there is a lack of visualization tools*".

4) **Finding failure-related logs is non-trivial**: As pointed out by our participants that "*the log size is usually large even when scoping down to a limited time span*", it is challenging to identify a small number of *key* log messages that could indicate the failure during diagnosis. Many survey participants complained that "*hard to find logs that could indicate the failure*", which motivated us to design the Q10. To identify failure-indicating logs, a common practice is to compare logs generated when a failure occurs with logs generated when there is no failure. Feedback from an engineer is "*quickly identifying common patterns is hard and would be helpful. For example, how many times we see log B following Log A would be helpful to identify if what we are looking at is a normal pattern or an anomaly*". But it is hard to achieve so since "*you're not really sure what baseline/normal behavior is*" and "*don't know whether the information is expected or caused the issue*". Additionally, logs are often very noisy. How to distinguish the failure-related logs from non-fatal logs then becomes hard, e.g., "*non-fatal errors that will pop up can significantly increase the analysis time*", and "*warning message is not warning, just informational*".

5) **Correlating logs from various sources is challenging**: Logs are often ingested from various sources to different files or databases. To deeply understand a failure, engineers have to inspect multiple log sources, which becomes a pain point for many survey participants. For instance, "*hard to correlate logs to understand the issue*" and "*have to interact with different log files to get a good understanding of the issue, and then correlate them separately*". Several obstacles are preventing the log correlation. First, different log sources have different terminologies and column names. It is mentioned that "*cross-team logs requires interpretation which isn't always self-evident and requires digging*" and "*schema is not consistent when dealing with complex systems*". Second, log data may vary by provenance, size, formats, and patterns. It is challenging to combine many heterogeneous logs to get a comprehensive view, e.g., "*do not know what the logs are about, where they come from and what columns mean if you are new to the issue*".

6) **Others**: Some pain points that do not belong to any of the categories mentioned above. A bunch of participants complained that they could not access logs easily. However, this is necessary since we must follow strict access control regulations to keep the information confidential. How to allow the feasibility for log analysis while keeping the information confidential might be an existing direction for exploration. Moreover, some engineers hope to involve more automation in log analysis, e.g., automatically generating the queries for analyzing logs, and the query results can later be provided as a support to resolve a failure.

5 LOG ANALYSIS RESEARCH

Over the last decade, log analysis has attracted much attention from academia [7, 13, 25, 37, 39, 49]. In this section, taking a similar structure as [16], we briefly introduce the mainstreaming log analysis topics, including logging, log parsing, log-based anomaly detection and failure diagnosis. We do not cover topics such as log compression and failure prediction since they are less prevalent in survey responses, which we leave for future work.

Logging: Logging refers to the instrumentation process of logging statements in the source code [16] to improve the software system diagnosability, maintenance, and performance (such as LogEnhancer [43], Log2 [12], and Errlog [41]). Most existing logging research [4] focuses on *where-to-log* and *what-to-log*. *Where-to-log* identifies the appropriate location in the source code to insert logging statements. *What-to-log* studies how to determine the verbosity level, static message, and variables in logging statements.

Log Parsing: Log parsing [15, 49] is a cornerstone task in the log analysis research, which aims to transform the unstructured logs into a well-structured format before feeding into the downstream applications (e.g., log-based anomaly detection). Specifically, a log parser identifies the static log template (constant logging text) and dynamic log parameters (variant runtime information such as variable values) from each raw log message. Almost all existing log parsers (e.g., Drain [15], Logram [7] and LenMa [34]) are based on the data-driven paradigm. Namely, they take a large volume of raw logs as the input and learn how to identify the common parts as templates and the dynamic parts as parameters.

Log-based Anomaly Detection: Log-based anomaly detection has attracted much attention in academia since the problem has been well-formulated by Xu et al. [39], and the benchmark datasets have been standardized and published in [18]. In general, a log sequence (a series of ordered logs that record the execution flow in a specific time-window/session/process) is the base unit of almost all existing log-based anomaly detectors. After the log collection and parsing, log sequences are represented as feature vectors using the parsed log events [47] (e.g., counting log event occurrences in a log sequence to construct an *event count vector*). Different anomaly detectors then adopt various technologies to determine whether an event count vector violates the normal behaviors. Existing log-based anomaly detectors can be roughly summarized into two categories, i.e., unsupervised approaches (e.g., PCA [39] and DeepLog [13]) and supervised approaches (e.g., SVM [22] and LogRobust [44]).

Log-based Failure Diagnosis: Logs have been widely leveraged to diagnose system failures. Broadly speaking, log-based diagnosis work can be categorized into two branches. One is reconstructing a failed execution or separating different execution flows based on log data [14, 27, 48]. The other branch is digging out the

root-cause related log messages when failures occur. The basic idea is to compare the difference between logs during the failure period and reference logs without the failure. Approaches following this paradigm include LogCluster [24], Log3C [17], and Onion [45].

6 GAP AND OPPORTUNITY

We have introduced industrial practices in Section 4 and academic achievements in Section 5 on log analysis. In this section, we attempt to present the gap observed when comparing actual needs in the industry with log research in academia and discuss some opportunities. We organize this section by following the log research topics in Section 5: logging, log parsing, anomaly detection and failure diagnosis. In each subsection, we first introduce the current practices and then share the gap and opportunity.

6.1 Logging

Logging is the process of instrumenting the source code with logging statements. As presented in Section 5, existing logging research concentrates on automating the decision making of where/what/how to log. However, developers in the industry implement logging primarily based on their domain expertise. Beyond it, engineers often consider the benefit of logging from the usage scenario of the generated logs, which we will discuss as follows.

Current Practices Typically, engineers first import the logging libraries (e.g., Log4j, Syslog) based on the usage scenario and programming languages. Following the logging grammar, engineers then decide the line where they want to insert a logging statement and the content to record, including verbosity level (Info/Warn/Debug, etc), static text and variables. The decision-making is usually based on their programming experience and understanding of the project. Logging statements could be updated if necessary, especially when logs are found insufficient for diagnosing a failure. Experienced engineers may also consider the cost of logging brought to the operation efficiency of software systems. There are some public references to logging practices [26, 28]. Besides, Yuan et al. [42] presented a summary of logging practices in open-source projects based on the logging and its modification history. Pecchia et al. [30] demonstrated a measurement study of logging practices in a critical industrial domain.

Gap As concluded by Chen et al. [4], logging research mainly focuses on how to insert the logging statements during the software development automatically. A challenge mentioned by our participant is to "find the balance between speed of finding the issue (e.g., more detail logs) and development effort". However, to our best knowledge, there is a **lack of *off-the-shelf* logging tools** or IDE extensions that can automate logging. For example, engineers would expect the logging tools "integrated with the code where I write/debug it every day, get that integrated into my IDE!" Besides that gap, through our survey and interview results, we found that engineers in the industry consider the logging issue more from how they would interpret the generated logs. We summarized some practical issues that are not well-covered in research: 1) **Terminology difference in logging**. It is found that "*different naming conventions across teams and services make it really hard to figure out what's going on*". For example, "*InciID*" and "*Incident ID*" refer to the sample concept. How to ensure different logging statements use consistent terms remains a practical problem, which aligns with the "*weak naming convention*" issue found by Pecchia et al. [30]. 2) **Logging that is hard to interpret**. There are two practical pain points found during the failure diagnosis, i.e., "*not familiar with how things work due to lack of readability*", and "*logs are typically written based on the developer's perspective when they should be based on helping the end-user (or whoever is reading the log) identify and correct the problem*". These issues are directly related to the logging quality (not readable). However, current logging research might rarely consider it from the view of log usage.

Opportunity To address the issue of lacking *off-the-shelf* logging tools, exploration on **automated logging with tool support** is needed. Along this direction, many practical needs should be considered, e.g., how to suggest the logging timely, how to avoid the privacy issues when asking developers to upload their code before suggesting logging statements. Another possible direction is to **produce readable and consistent logging**. It desires research efforts on how to produce logging statements such that they could be more readable for developers or can facilitate the diagnosis process. Meanwhile, it is important to check whether the naming in logging statements is consistent within a component and cross components or even address the inconsistency.

> *Finding 1*: There is a lack of off-the-shelf logging tools and current logging research is rarely approached from the log usage perspective, which demands practical logging tools (or IDE extensions) that could generate readable and consistent logging statements.

6.2 Log Parsing

Most log parsing research aims to transform the unstructured logs into a structured format by learning from a massive number of log messages. In industry, structuring logs (i.e., log parsing) is crucial, especially when summarizing the insights from logs.

Current Practices Log parsing in practice is often realized by a rule-based approach. Usually, engineers query the log database with either keyword search (e.g., contains "error code XX") or complex regular expressions to find the desired logs, as pointed out by Rodrigues et al.[33]. These methods are adopted in log companies such as Datadog [11] and Splunk [35]). However, manually structuring logs is time-consuming and does not scale up to different log sources. Our survey respondents pointed out the challenges that "*hard to extract data from free form fields*" and "*sometimes we try to parse the message string, extracting desired fields or variables from a fairly standard logging text*', as presented in Section 4.2.

Gap The first gap is that **log research and industry practice focus on logs of different formats**. As presented in Figure 2 (a), the unstructured log covered in most existing log research [16] takes up only 25% of log formats that participants often work with, while the majority of logs are either structured (41%) or semi-structured (34%). The second gap is that **log parsing is generally achieved by lightweight instrumentation, while the after-the-fact data-driven log parsing are used in some restricted scenarios (e.g., third-party libraries)**. In the 1-1 interview, several product teams at Microsoft demonstrated the lightweight instrumentation for log parsing, which we name as the

event pre-assignment. Specifically, after engineers instrument logging statements, commit the code and merge the pull request, a logging action (like the GitHub Action) will be automatically triggered. The action will profile the source code and insert a random-generated unique event ID string to each newly-added logging statement. In this way, the event ID (e.g., "Hzhtl7mk") would be printed together with the log message when executing each logging statement. Therefore, in their log data, log messages generated from the same logging statement share the same event ID. The template and variable parts can then be extracted effortlessly. Another benefit is that engineers can quickly locate the correct logging code and its context for troubleshooting by searching the event ID. Similarly, public logging frameworks such as Log4j [2] allow the print of *function name* and *log line number*, which can also be combined and utilized as an event ID to determine the unique logging statement.

Opportunity The above-mentioned *event pre-assignment* does not mean to be perfect. For example, how to ensure that many event IDs in a large system do not incur the collision, how to keep the event ID consistent even when updating the logging statements. There might be some simple solutions that should be further explored. Moreover, in scenarios such as the source code cannot be accessed (e.g., in third-party libraries) or it is costly to refactor the logging statements, how to assign the event IDs becomes an open question. For example, can we leverage some dynamic instrumentation techniques to attach the event ID during runtime? If not, data-driven log parsing might be the only possible solution for log structuring, which is exactly the direction that current research [19, 49] is working on.

> *Finding 2*: Research and industry focus on logs of distinct formats. Log parsing is generally achieved with lightweight instrumentation, while the after-the-effect data-driven methods are still promising in some restricted scenarios, such as parsing logs of third-party libraries.

6.3 Anomaly Detection

Log anomaly detection research aims to identify whether a log sequence (e.g., logs that record a workflow) is abnormal. However, when talking about *anomaly detection* in industry, it usually refers to the process of detecting failures in a software component or system, which we use the term *failure alerting* to denote it.

Current Practices In modern software systems, failure alerting is usually accomplished by an independent and reliable component for continuous monitoring and alerting. Generally, the alerting component consumes the metric data and raises alerts when detecting an anomaly in the *time series* of metrics. The metric is the numeric representation of data measured over time intervals, such as the failure rate, request latency or CPU utilization rate per five minutes. When metrics are sometimes insufficient, engineers may resort to logs for failure alerting. In detail, engineers would transform the logs into metrics by aggregation, such as counting the number of the desired log event within a period. If the logs are unstructured, log parsing is applied first to extract the log events. As pointed out by our survey participant, "*Our team needs to count the number of exception-E message by querying from our service logs and set up the alert. Once triggered, our on-call engineers will engage to find out why these exceptions happen and take actions.*"

Table 4: Comparison of log-based anomaly detection in academia and failure alerting in industry.

	Anomaly Detection	**Failure Alerting**
Goal	Identify abnormal behaviors in log sequences	Identify failures causing disruption to systems
Processing	Unstructured logs → Log sequences feature vectors	Unstructured Logs → Metrics
Technique	Invariant Mining, PCA, LSTM, etc (see Section 5)	Time series anomaly detection methods (e.g., k-sigma/thresholding)
Efficiency	Mostly not real-time	Real-time
Interpretability	Not easily understood	Easy to understand
Noise	More	Less

Gap In Table 4, we compared anomaly detection with unstructured logs in academia (in short, anomaly detection) and failure alerting in the industry (in short, failure alerting) from various perspectives, including the goal, processing method and technique, efficiency, interpretability, noise, and the generalization ability. **1) Goal**. Anomaly detection targets to distinguish "abnormal" log sequences (e.g., all logs of a user request) from normal log sequences. However, the industry is more concerned with whether any failure is observed and disrupts the system functionalities. It is crucial to note that the detected anomaly does not necessarily imply a real system failure, especially in large-scale systems with strong fault-tolerance ability. The anomalous log sequence may only indicate a different program execution flow. **2) Processing method and technique**. Anomaly detection usually transforms the logs into a high-dimension feature vector of log sequence, followed by machine learning or deep learning techniques to predict whether a log sequence is abnormal. On the contrary, failure alerting transforms unstructured logs into metrics as mentioned above and then uses methods such as k-sigma to detect time-series anomalies. **3) Efficiency**. Most existing anomaly detection studies do not consider the need for real-time processing, which however is indispensable for industrial systems. It is worthy to note that the metric signals are usually collected in real time, while log collection sometimes may incur a long latency delay. **4) Interpretability and Noise**. Anomaly detection results from a sophisticated model are often difficult to interpret, especially for engineers who do not understand machine learning. Instead, the metric like failure rate is a clear signal to measure the system health status with strong physical meaning. In addition, anomaly detection tends to identifies more false positives (noises), since an anomalous log sequence may not reflect a failure but instead an abnormal user behavior.

Opportunity Due to the huge differences between anomaly detection in industry and failure alerting in industry, many opportunities could arise when we try to **bridge the gap between the two sides**. For example, how to increase the efficiency of anomaly detection methods given a great volume of streaming logs? Also, regarding the interpretability, how to identify anomalies and provide the reasoning logic behind the identification, such as which logs in the log sequence are abnormal and what is the normal pattern? One of our survey participants also mentioned that "*quickly identifying common patterns is hard and would be helpful. For example, how many times do we see log B following Log A would be helpful to identify if what we are looking at is a normal pattern or an anomaly*".

It can act as a vivid example of what engineers would expect from the interpretation of anomaly detection results. From another point of view, to distinguish the abnormal and normal logs, anomaly detection could be utilized to **identify which subset of logs are more likely to reflect the root cause of a failure**.

> *Finding 3*: A huge gap exists between anomaly detection in academia and failure alerting in the industry regarding the goal, techniques, efficiency, interpretability, *etc.* Many opportunities could arise towards bridging the gap, e.g., interpretable anomaly detection.

6.4 Failure Diagnosis

Log-based failure diagnosis, which aims to find out hints towards the root cause, could occur in various scenarios, such as triage, mitigation, or root cause analysis.

Current Practices The diagnosis process could vary for different products, teams and even engineers. Here we presented a typical way of diagnosis. In general, engineers usually "*search for the symptom or related error messages in a knowledge base to see whether the failure happened before*". If so, they will follow the trouble-shooting guide (TSG) to diagnose the problem. Instead, if the failure is unknown, engineers have to "*manually inspect any necessary information like logs*" by methods such as keyword search and reading line by line (as presented in Figure 3). To diagnose these failures, engineers often need to understand the logic behind log messages by reconstructing the execution flow and figuring out the root cause. However, if there are too many logs or it is hard to construct the execution flow, a typical way is to compare logs generated in a failure with logs generated in normal status and then find the differentiating log messages. During this process, engineers may have to query many times to get the desired information.

Gap Many existing failure diagnosis research aims to extract clues or insights from logs generated by *a batch of failures*. Failures in such a batch share the same root cause and similar log patterns. A common solution is to make a contrast analysis. For example, Zhang et al. [45] identify the incident-indicating logs by contrasting logs on normal and failed nodes. The **batch failure diagnosis is close to the current industrial practices**, and one possible reason is that many diagnosis research was actually conducted in industrial scenarios (such as LogCluster [37], Onion [45], Log3C [17]). In these cases, the failure amount and log size reach a large scale, making applying various statistical analysis methods feasible. However, our survey and interview results also suggest the **strong demands for diagnosing an individual failure, but few research studies focused on it**. The individual failure does not happen as a batch (such as customer reported issues), which are often analyzed case by case, and the underlying root cause could vary.

Opportunity In diagnosing individual failures, engineers often record the analysis process and solution as a TSG for future reuse. For example, in the TSG, engineers could record which log data to inspect, which log messages to check against (in the form of log queries), root cause and resolution, *etc.* However, these TSGs currently do not follow a standard format and are often manually crafted. Additionally, it is often a manual searching and execution process before finding the right TSGs. Therefore, to make log-based diagnosis more efficient, a potential direction is to **make the TSG easy to manage and reuse**. In detail, we may set up a standard format for TSGs, transform the TSG into an executable one and even summarize the TSG intelligently. Then, we may automatically recommend TSG based on the logs and execute it.

> *Finding 4*: Batch failure diagnosis in industry aligns well with academia, but diagnosing individual failure is not well-explored. Opportunities such as automated execution of trouble-shooting guides might be promising.

7 LIMITATIONS

Survey Subject: Our survey participants are limited to employees at a single software company, hence the results may not generalize to other companies. To mitigate the threat to generalizabily, we recruited participants from products of diverse organizations, including Azure, M365, Windows, Bing and LinkedIn, as introduced in Section 3. These products cover most existing software types in the broad industry. Moreover, our participants are of different roles, such as software engineers and program managers. Besides, to mitigate threats to internal validity, we further recruited participants by various channels. For instance, we posted the survey to several communities consisting of employees with different backgrounds.

Research Work Survey: In the research survey section (Section5), following the latest literature by He et al. [16], we focus more on the unstructured log research work because it is the mainstream of the automated log data analysis field. Structured log analysis is also well studied in academia. Nevertheless, in the "structured log" setting, log parsing is no longer necessary due to its standardized format. Anomaly detection and diagnosis on top of structured logs can also be formulated into well-established problems, such as time series based anomaly detection [32] or attributes combination search problem [23]. Therefore, following He et al. [16], our research review does not incorporate the structured log analysis work.

Cognitive Bias: Participants are likely to hold different understandings of the same concept owning to their different backgrounds. For example, facing the question "Which log format do you analyze most frequently?" some respondents may misunderstand "semi-structured logs" as "structured" because the former is often organized as the table (structured) format. To mitigate this construct validity, we attached some explanations or examples (as shown in Figure 1) by the side of those questions.

8 CONCLUSION

This study presents a comprehensive survey study on log analysis in Microsoft with a questionnaire and individual interviews. Based on the study results, we summarized the log usage in practice from various perspectives, including log format, scenario, method, tool, and challenges. These results reveal the first-hand experience of log analysis practitioners in the production environment. Besides, we discussed the current practices, gaps between the industry and academia, and future opportunities for exploration. For example, there is a huge gap between log-based anomaly detection research and failure alerting practices from various perspectives. We hope these findings can make log analysis researchers and practitioners realize the significant gap between industry and academia and inspire interesting future directions that facilitate the convergence of these two lines of log analysis research.

REFERENCES

[1] Reem S Alsuhaibani, Christian D Newman, Michael J Decker, Michael L Collard, and Jonathan I Maletic. 2021. On the Naming of Methods: A Survey of Professional Developers. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 587–599. https://doi.org/10.1109/ICSE43902.2021.00061

[2] Apache. 2022. *Apache Log4j 2.* Retrieved May 18, 2022 from https://logging.apache.org/log4j/2.x/

[3] Titus Barik, Robert DeLine, Steven Drucker, and Danyel Fisher. 2016. The bones of the system: A case study of logging and telemetry at microsoft. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 92–101. https://doi.org/10.1145/2889160.2889231

[4] Boyuan Chen and Zhen Ming Jiang. 2021. A Survey of Software Log Instrumentation. *ACM Computing Surveys (CSUR)* 54, 4 (2021), 1–34. https://doi.org/10.1145/3448976

[5] Zhuangbin Chen, Yu Kang, Liqun Li, Xu Zhang, Hongyu Zhang, Hui Xu, Yangfan Zhou, Li Yang, Jeffrey Sun, Zhangwei Xu, et al. 2020. Towards intelligent incident management: why we need it and how we make it. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1487–1497. https://doi.org/10.1145/3368089.3417055

[6] Daniela S Cruzes and Tore Dyba. 2011. Recommended steps for thematic synthesis in software engineering. In *2011 international symposium on empirical software engineering and measurement*. IEEE, 275–284.

[7] Hetong Dai, Heng Li, Che Shao Chen, Weiyi Shang, and Tse-Hsun Chen. 2020. Logram: Efficient log parsing using n-gram dictionaries. *IEEE Transactions on Software Engineering* (2020). https://doi.org/10.1109/TSE.2020.3007554

[8] Anastasia Danilova, Alena Naiakshina, Stefan Horstmann, and Matthew Smith. 2021. Do you really code? designing and evaluating screening questions for online surveys with programmers. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 537–548. https://doi.org/10.1109/ICSE43902.2021.00057

[9] Anwesha Das, Frank Mueller, and Barry Rountree. 2020. Aarohi: Making real-time node failure prediction feasible. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 1092–1101. https://doi.org/10.1109/IPDPS47924.2020.00115

[10] Datadog. 2022. *Datadog.* Retrieved May 18, 2022 from https://www.datadoghq.com

[11] Datadog. 2022. *Datadog Log Parsing.* Retrieved May 18, 2022 from https://docs.datadoghq.com/logs/guide/log-parsing-best-practice/

[12] Rui Ding, Hucheng Zhou, Jian-Guang Lou, Hongyu Zhang, Qingwei Lin, Qiang Fu, Dongmei Zhang, and Tao Xie. 2015. Log2: A cost-aware logging mechanism for performance diagnosis. In *2015 {USENIX} Annual Technical Conference ({USENIX}{ATC} 15)*. 139–150.

[13] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1285–1298. https://doi.org/10.1145/3133956.3134015

[14] Xiaoyu Fu, Rui Ren, Sally A McKee, Jianfeng Zhan, and Ninghui Sun. 2014. Digging deeper into cluster system logs for failure prediction and root cause diagnosis. In *2014 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 103–112.

[15] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. 2017. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE international conference on web services (ICWS)*. IEEE, 33–40. https://doi.org/10.1109/ICWS.2017.13

[16] Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R Lyu. 2021. A survey on automated log analysis for reliability engineering. *ACM Computing Surveys (CSUR)* 54, 6 (2021), 1–37. https://doi.org/10.1145/3460345

[17] Shilin He, Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, Michael R Lyu, and Dongmei Zhang. 2018. Identifying impactful service system problems via log analysis. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 60–70. https://doi.org/10.1145/3236024.3236083

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

[19] Zanis Ali Khan, Donghwan Shin, Domenico Bianculli, and Lionel Briand. 2022. Guidelines for Assessing the Accuracy of Log Message Template Identification Techniques. In *Proceedings of the 44th International Conference on Software Engineering (ICSE'22)*. ACM. https://doi.org/10.1145/3510003.3510101

[20] Ralf Lämmel, Alvin Kerber, and Liane Praza. 2020. Understanding What Software Engineers Are Working on: The Work-Item Prediction Challenge. In *Proceedings of the 28th International Conference on Program Comprehension*. 416–424.

[21] Heng Li, Weiyi Shang, Bram Adams, Mohammed Sayagh, and Ahmed E Hassan. 2020. A qualitative study of the benefits and costs of logging from developers' perspectives. *IEEE Transactions on Software Engineering* (2020). https://doi.org/10.1109/TSE.2020.2970422

[22] Yinglung Liang, Yanyong Zhang, Hui Xiong, and Ramendra Sahoo. 2007. Failure prediction in ibm bluegene/l event logs. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*. IEEE, 583–588.

[23] Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, and Dongmei Zhang. 2016. iDice: problem identification for emerging issues. In *Proceedings of the 38th International Conference on Software Engineering*. 214–224. https://doi.org/10.1145/2884781.2884795

[24] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. 2016. Log clustering based problem identification for online service systems. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 102–111. https://doi.org/10.1145/2889160.2889232

[25] Jinyang Liu, Jieming Zhu, Shilin He, Pinjia He, Zibin Zheng, and Michael R Lyu. 2019. Logzip: extracting hidden structures via iterative clustering for log compression. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 863–873. https://doi.org/10.1109/ASE.2019.00085

[26] Loggly. 2022. *30 best practices for logging at scale.* Retrieved May 18, 2022 from https://www.loggly.com/blog/30-best-practices-logging-scale/

[27] Jian-Guang Lou, Qiang Fu, Shengqi Yang, Jiang Li, and Bin Wu. 2010. Mining program workflow from interleaved traces. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. 613–622.

[28] Medium. 2022. *13 best practices to improve your logging.* Retrieved May 18, 2022 from https://medium.com/cogito-engineering/13-best-practices-to-improve-your-logging-75665183f05b

[29] Microsoft. 2021. *Microsoft Forms.* Retrieved May 18, 2022 from https://www.microsoft.com/en-us/microsoft-365/online-surveys-polls-quizzes

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

[31] Teade Punter, Marcus Ciolkowski, Bernd Freimut, and Isabel John. 2003. Conducting on-line surveys in software engineering. In *2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings.* IEEE, 80–88.

[32] Hansheng Ren, Bixiong Xu, Yujing Wang, Chao Yi, Congrui Huang, Xiaoyu Kou, Tony Xing, Mao Yang, Jie Tong, and Qi Zhang. 2019. Time-series anomaly detection service at microsoft. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3009–3017. https://doi.org/10.1145/3292500.3330680

[33] Kirk Rodrigues, Yu Luo, and Ding Yuan. 2021. {CLP}: Efficient and Scalable Search on Compressed Text Logs. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*. 183–198.

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

[35] Splunk. 2022. *Splunk.* Retrieved May 18, 2022 from https://www.splunk.com

[36] Cindy Sridharan. 2018. *Distributed Systems Observability: A Guide to Building Robust Systems.* O'Reilly Media.

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

[38] Wikipedia. 2021. *Observability.* Retrieved September 20, 2021 from https://en.wikipedia.org/wiki/Observability

[39] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. 2009. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. 117–132.

[40] Nan Yang, Ramon Schiffelers, and Johan Lukkien. 2021. An interview study of how developers use execution logs in embedded software engineering. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 61–70.

[41] Ding Yuan, Soyeon Park, Peng Huang, Yang Liu, Michael M Lee, Xiaoming Tang, Yuanyuan Zhou, and Stefan Savage. 2012. Be conservative: Enhancing failure diagnosis with proactive logging. In *10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12)*. 293–306.

[42] Ding Yuan, Soyeon Park, and Yuanyuan Zhou. 2012. Characterizing logging practices in open-source software. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 102–112.

[43] Ding Yuan, Jing Zheng, Soyeon Park, Yuanyuan Zhou, and Stefan Savage. 2012. Improving software diagnosability via log enhancement. *ACM Transactions on Computer Systems (TOCS)* 30, 1 (2012), 1–28.

[44] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, et al. 2019. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 807–817. https://doi.org/10.1145/3338906.3338931

[45] Xu Zhang, Yong Xu, Si Qin, Shilin He, Bo Qiao, Ze Li, Hongyu Zhang, Xukun Li, Yingnong Dang, Qingwei Lin, et al. 2021. Onion: identifying incident-indicating logs for cloud systems. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software*

Engineering. 1253–1263. https://doi.org/10.1145/3468264.3473919

[46] Yongle Zhang, Kirk Rodrigues, Yu Luo, Michael Stumm, and Ding Yuan. 2019. The inflection point hypothesis: a principled debugging approach for locating the root cause of a failure. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 131–146. https://doi.org/10.1145/3341301.3359650

[47] Nengwen Zhao, Honglin Wang, Zeyan Li, Xiao Peng, Gang Wang, Zhu Pan, Yong Wu, Zhen Feng, Xidao Wen, Wenchi Zhang, et al. 2021. An empirical investigation of practical log anomaly detection for online service systems. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1404–1415. https://doi.org/10.1145/3468264.3473933

[48] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Wenhai Li, and Dan Ding. 2018. Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study. *IEEE Transactions on Software Engineering* (2018).

[49] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R Lyu. 2019. Tools and benchmarks for automated log parsing. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 121–130. https://doi.org/10.1109/ICSE-SEIP.2019.00021