Techniques to Improve the Parsing of Unstructured Logs for AIOps

Issam SEDKI

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Ph.D. (Electrical and Computer Engineering) at

Concordia University

Montréal, Québec, Canada

March 2025

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Issam SEDKI

Entitled: Techniques to Improve the Parsing of Unstructured Logs for AIOps

and submitted in partial fulfillment of the requirements for the degree of

Doctor Of Philosophy Electrical and Computer Engineering

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. Manar Amayri — Chair

Dr. Domenico BIANCULLI — External Examiner

Dr. Hassan Rivaz — Arm's Length Examiner

Dr. Yann-Gaël Guéhéneuc — Examiner

Dr. Nawwaf Kharma — Examiner

Dr. Wahab Hamou-Lhadj — Thesis Supervisor (s)

Dr. Otmane Ait Mohamed

Approved by

Dr. Wei-Ping Zhu, Associate Chair — Chair of Department or Graduate Program Director

11/02/2025
Date of Defence

Mourad Debbabi — Dean, Faculty of Engineering and Computer Science

Abstract

Techniques to Improve the Parsing of Unstructured Logs for AIOps

Issam SEDKI

Artificial Intelligence for IT Operations (AIOps) is revolutionizing IT management by incorporating AI, machine learning, and big data analytics to automate and enhance system operations. Logs are the backbone of AIOps—they provide the fundamental data on system events, user activities, and performance metrics that AIOps needs for proactive monitoring, predictive analytics, and maintaining system health. Log management, especially log parsing, is therefore critical for identifying anomalies, diagnosing failures, and ensuring overall operational efficiency. However, challenges such as diverse log formats, insufficient logging guidelines, the sheer volume of logs, and the need for real-time insights significantly limit the precision, scalability, and effectiveness of AIOps.

This thesis develops novel methods for universal log parsing, introduces accurate evaluation metrics, proposes a comprehensive taxonomy of log characteristics, and addresses privacy compliance, collectively advancing the efficacy, scalability, and trustworthiness of AIOps.

Inaccurate log parsing can lead to inaccurate insights—misleading or incorrect conclusions drawn from analysis. Such errors may cause AIOps to misclassify events, overlook crucial anomalies, or generate noise that obscures genuine issues. To address this, the first major contribution of this thesis is the development of ULP (Universal Log Parser), which leverages a frequent token counting method to identify recurring patterns and extract log templates efficiently. By reducing computational complexity, ULP enables faster, more accurate log parsing, making it highly effective for large-scale IT environments—a key capability for the automation and responsiveness required in AIOps.

The second contribution is AML (Accuracy Metric for Log Parsing), a structured framework

for evaluating the accuracy of log parsing. Traditional metrics are insufficient for heterogeneous log formats, leading to errors and subsequent misguided AIOps decisions. AML offers nuanced metrics that account for both omission and commission errors, enabling detailed and reliable comparisons across different log parsers.

The third contribution is a taxonomy of log characteristics, categorizing logs based on their structural and contextual properties. This taxonomy not only guides parser design by clarifying how logs differ across applications but also informs logging practices, helping practitioners tailor log writing strategies for better analytics.

The fourth contribution focuses on enhancing log privacy compliance, a crucial aspect in AIOps —especially as automated processes handle large volumes of sensitive log data. The thesis provides guidelines for evaluating and managing privacy risks associated with log data, ensuring that the automation capabilities of AIOps remain compliant with stringent privacy regulations and best practices.

Together, these contributions form a framework for advancing log parsing within AIOps. It is a holistic approach—encompassing efficient universal parsing, robust accuracy evaluation, a guiding taxonomy, and built-in privacy compliance—that addresses the entire life cycle of log-based analytics. This framework ultimately strengthens the capabilities of IT operations to be more proactive, responsive, and compliant, paving the way for the next generation of AIOps-driven IT management.

Acknowledgments

I wish to express my deepest gratitude to my family. To my parents, Ahmed Sedki and Rachida Bsibissa, your unwavering faith and unconditional love laid the foundation for every step of this journey. You taught me the values of persistence and resilience, and I carry them with me in everything I do. To my wife, Wafaa El Bachri, words cannot capture the depth of my gratitude. This journey has been as emotionally challenging for you as it has been for me. You stood by me through the countless emotional burnouts, offering not just encouragement but also unwavering patience and understanding. You bore the weight of this journey with me, and without your strength, none of this would have been possible. To my children, thank you for your love and forbearance. I know I wasn't always there mentally, as my mind was often preoccupied with shaping up my research. Your smiles and laughter reminded me what truly matters, and I hope this achievement brings us all closer.

I am profoundly grateful to my main supervisor, Dr. Wahab Hamou-Lhadj, for his exceptional guidance and mentorship throughout my Ph.D. Wahab, I deeply value our many late-night discussions, where we brainstormed, challenged ideas, and envisioned the future of AIOps AIOps together. You pushed me to think critically and boldly, and your support extended far beyond just research; you became a sincere friend and mentor in every sense. Your influence on this work and on me personally is immeasurable.

My co-supervisor, Dr. Otmane Ait Mohamed, has been an indispensable part of this journey. Otmane, your patience, calm wisdom, and constant mental support provided a crucial balance through the highs and lows. Your steady encouragement helped me stay grounded, and I am deeply thankful for your guidance and friendship.

I would also like to extend my heartfelt thanks to Dr. Yann-Gaël Guéhéneuc for his invaluable

suggestions and sharp insights, which consistently pushed me to think deeper and improve my work. To my examination committee members, Dr. Nawwaf Kharma and Dr. Hassan Rivaz, your thoughtful feedback refined this thesis, and your perspectives helped shape it into something stronger and more focused.

I would also like to thank the Innovation for Defence Excellence and Security (IDEaS) program from the Canadian Department of National Defence (DND), the Natural Sciences and Engineering Research Council of Canada (NSERC), the Gina Cody School of Engineering and Computer Science, and the Faculty of Graduate Studies for their generous financial support.

Special thanks go to Dr. Mohamed Shehab, whose collaboration and insights were instrumental in shaping key elements of this research. I am grateful for your partnership and intellectual contributions.

I would also like to thank Audrey Veilleux for her immense patience and tireless support in navigating the administrative challenges throughout my time at Concordia University. Your willingness to help and your kindness never went unnoticed.

To my colleagues and collaborators, thank you for being a part of this journey. Your camaraderie and shared knowledge enriched my academic experience in countless ways.

Finally, to all those whose names are not mentioned here but who have contributed to my personal and professional growth—thank you from the bottom of my heart.

Co-Authorship

My contributions throughout the chapters and related publications in this thesis are summarized as follows: proposing the research ideas, investigating background knowledge and conducting literature reviews, formulating initial research methodologies, collecting datasets for evaluation, conducting experiments, analyzing the experimental results, and writing and refining manuscripts.

The invaluable suggestions and guidance from my co-authors played a crucial role in refining my research ideas and methods, ensuring the reliability and comprehensiveness of the experimental results, and enhancing the clarity and quality of the written works.

The following publications have resulted from the research conducted during my Ph.D. journey.

These studys demonstrate various aspects of my contributions to the field of Log Parsing and AIOps in general.

- I. Sedki, A. Hamou-Lhadj, O. Ait-Mohamed and M. A. Shehab, "An Effective Approach for Parsing Large Log Files," in Proc. of the 38th IEEE International Conference on Software Maintenance and Evolution (ICSME), Limassol, Cyprus, 2022, pp. 1-12, doi: 10.1109/ICSME55016.2022.00009.

- I. Sedki, A. Hamou-Lhadj, O. Ait-Mohamed and N. Ezzati-Jivan, "Towards a Classification of Log Parsing Errors," in Proc. of the 31st International Conference on Program Comprehension (ICPC), Melbourne, Australia, 2023, pp. 84-88, doi: 10.1109/ICPC58990.2023.00023.

- I. Sedki. "Privacy-Centric Log Parsing for Timely, Proactive Personal Data Protection," In Proc. of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), San Fransisco, USA, 2023 pp. 2204–2206, doi:https://doi.org/10.1145/3611643.3617847

- I. Sedki, A. Hamou-Lhadj, O. Ait-Mohamed, "AML: An accuracy metric model for effective evaluation of log parsing techniques", In Proc. of the Journal of Systems and Software (JSS), Volume 216, 2024, 112154, ISSN 0164-1212, https://doi.org/10.1016/j.jss.2024.112154.

- I. Sedki, A. Hamou-Lhadj, O. Ait-Mohamed and N. Ezzati-Jivan, and M. Shehab. "Decoding Log Parsing Challenges: A Comprehensive Taxonomy for Actionable Solutions," In Proc. of the 46th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion '24). pp. 392–393. doi: https://doi.org/10.1145/3639478.3643523

- I. Sedki. "A Preliminary Study on the Privacy Concerns of Using IP Addresses in Log Data". In Proc. of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE 2024 Ideas, Visions and Reflections), New York, NY, USA, pp. 527–531. doi: https://doi.org/10.1145/3663529.3663791

- I. Sedki, A. Hamou-Lhadj, O. Ait-Mohamed and N. Ezzati-Jivan, "Developing a Taxonomy for Advanced Parsing Techniques," In Proc. of the 33rd ACM/IEEE International Conference of Program Comprehension (ICPC), Ottawa, ON, Canada, April 2025

- **Ranked 1st at FSE student competition, San Fransisco, December 2023**

Statement of Originality

I, Issam SEDKI, hereby declare that all the intellectual content of this thesis, and the research to which it refers, is my own work. All received assistance during the preparation of this thesis has been properly acknowledged.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Artificial Intelligence for IT Operations (AIOps) represents a transformative approach to IT management, combining machine learning (ML), big data, and analytics to optimize and automate IT services. AIOps is designed to enhance operational workflows by providing proactive monitoring, predictive insights, and data-driven recommendations that help reduce failures, improve mean-time-to-recovery (MTTR), and allocate resources more effectively [1, 2]. As a rapidly evolving field, AIOps holds substantial potential in a variety of IT tasks, ranging from anomaly detection [3, 4, 2**?**] to resource optimization, making it increasingly vital for managing complex IT environments. [5]

The software industry has shifted from delivering boxed products to offering online applications and cloud-based solutions. This transition has led to a fundamentally different approach to building, releasing, and maintaining software, with a growing emphasis on operational efficiency. DevOps has emerged as a popular methodology to support the continuous development and release of these services. However, with the proliferation of cloud computing, the scale and complexity of services have increased dramatically, making it challenging for engineers to efficiently manage and operate these systems [6]. In response to these challenges, the concept of AIOps was introduced by ”Gartner” to leverage artificial intelligence in addressing DevOps challenges, enhancing both efficiency and reliability.

The core of AIOps is its ability to process vast amounts of operational data in real time to detect patterns, anticipate issues, and provide insights for IT administrators [7]. The state-of-the-art research in AIOps covers resource allocation, fault prediction [8][**?**], and automation in log analytics [9][10], which together form the backbone of modern IT service management. These branches are vital for understanding how AIOps improves the reliability and efficiency of IT services and why ongoing research in these areas is critical [11].

Logs, the foundation of much of AIOps, serve as data sources for understanding system behavior. Logs are generated by inserting log lines into the source code, creating a collection of messages that chronologically record system events. These log entries, whether single- or multi-line, capture specific runtime events in various computing systems, including large-scale distributed clusters, self-contained applications, and user devices. Each log entry typically contains fields such as timestamps, severity levels, process IDs, and log messages that provide context about the system's state. While some fields, like timestamps, follow standard formats, the log message field is often free-text, defined by developers, and varies across systems.

For instance, Figure 1.1 illustrates a logging statement from the Hadoop Distributed File System (HDFS). The log event consists of a log header with contextual information (e.g., time, process ID, log level) and a log message detailing the specific event.

Log management involves the collection, storage, and preprocessing of logs for subsequent analysis. Effective log management is crucial for ensuring that valuable information is not lost in the massive influx of data produced by complex IT systems. It also involves creating structured representations of logs, which often contain unstructured free-text messages written by developers. Given the heterogeneity and volume of log formats, manual log analysis is impractical, leading to the need for automated solutions that can extract actionable insights efficiently. Delays in identifying anomalies or diagnosing system failures can lead to cascading issues, such as prolonged downtime or reduced system reliability. Moreover, manual inspection requires deep system expertise and often fails to capture subtle patterns that may signal emerging problems, exacerbating the difficulty of preventing incidents before they escalate.

Logging statement: LOG.info("Received Block"+ block + "of size" + block_size + " from" + sender_ip)

Raw log line: 081109 283519 147 INFO dfs.DataNodePacketResponder: Received block blk_-1680999687919862986 of size 91178 from /10.250.14.224

Log template: Received Block <*> of size <*> from <*>

Figure 1.1: An example of a logging statement, the generated Log Event, and the expected log template

Automated log analysis has become a critical component of modern software engineering, playing a pivotal role in enabling tasks such as anomaly detection, root cause analysis, system debugging, and performance monitoring [3, 4, 2**?**]. Such methods must not only process the vast high-throughput of log data efficiently but also adapt to the dynamic, evolving structures of logs across diverse systems. When logs originate from a wide array of applications, platforms, or microservices, their structure and format naturally evolve over time. Updates to software versions, changes in configuration, or the introduction of new services can alter the number and arrangement of fields, the tokens used to describe events, or even the semantics of specific log messages. Consequently, static parsing approaches can become outdated when these log templates change, impeding accurate detection of anomalies and other insights.

The core goal of log parsing is to transform raw, unstructured log events into a structured form, typically represented by log templates. These templates separate the static components of a log event (known as the "log template") from the dynamic tokens, which vary in each log event. For instance, consider the log event depicted in Figure 1.1; the corresponding log template is "Received Block * of Size * from ," where the "" symbol represents dynamic tokens. Log parsing aims to extract these templates, providing a more structured representation of log data that aids in understanding system behavior and facilitates efficient analysis. The growing importance of automated log parsing has led to the development of a wide array of log parsers, employing diverse methodologies such as machine learning, pattern mining, natural language processing (NLP) [12] [13, 14] as well as LLMs [15, 16, 17, 18] with a very growing interest in the LLM world. A comprehensive survey of these tools is presented by El-Masri et al. [19]. Some notable examples of log parsers include Drain [20],

Spell [21], SHISO [22], AEL [23], Lenma [24], LogSig [25], SLCT [26], IPLoM [27], LKE [28], LogMine [29], Logram [13], and ULP [30].

Due to the absence of standardized logging guidelines and best practices [31, 32], log structures often vary widely between different systems [33]. Although there are occasional efforts from standards bodies and open-source communities to provide logging frameworks (e.g., for specific programming languages or platforms), there is currently no universally mandated authority or widely adopted global standard that prescribes how logs should be written or organized. As a result, each organization—or even each development team—tends to define log templates based on internal conventions, leading to significant variability. This variability, coupled with the continually expanding size of log files [34, 35] [36], significantly complicates log parsing, hindering efficient system analysis and troubleshooting [37, 38]. Consequently, log parsers often yield incomplete or incorrect results, which can propagate inaccuracies to downstream analyses, reducing the effectiveness of AIOps tools and compromising the reliability of the entire workflow.

Key challenges in Log Parsing include:

- Heterogeneity of log formats: Logs are generated from a wide variety of systems, each with its own structure, syntax, and dynamic elements, such as session IDs and timestamps. This heterogeneity complicates the process of extracting accurate log templates and distinguishing between static and dynamic components. Having a dedicated log parser for each log type might seem appealing in small, uniform environments, but several practical issues make this approach challenging in real-world, large-scale systems. Modern infrastructures often integrate dozens of services—ranging from microservices and third-party APIs to legacy systems—each generating its own unique log structure. Maintaining a distinct parser for every format quickly becomes unmanageable. Moreover, log structures are not static; they evolve as software applications are updated, so even a single service may generate different log formats across versions or environments. Keeping separate parsers current for each variation adds significant overhead and delays responses to new formats. In addition, many enterprise environments rely on correlating logs from multiple sources—such as network equipment, databases, and cloud platforms—requiring a more general or flexible parsing approach that

avoids siloed analysis. Building and running multiple specialized parsers also strains computing resources, as each parser may demand its own configuration and processing pipeline. Finally, as the number of applications scales, the burden of maintaining an expanding suite of specialized parsers grows exponentially, undermining manageability and increasing costs. Consequently, although one-to-one parsers may yield highly accurate results in limited or controlled settings, such an approach is impractical at scale and can hinder efficient, adaptable log analysis.

- Dynamic nature of logs: Logs often contain dynamic elements such as session IDs or timestamps, making it challenging to distinguish between static templates and dynamic tokens.

- Lack of logging guidelines: Developers often create log statements without clear guidelines, leading to inconsistent logging practices across different systems. This variability in log levels, structure, and detail adds complexity to parsing and subsequent analysis.

- Noisy data and incorrect analysis: Logs can contain a significant amount of noise—irrelevant entries or redundant information—that makes effective analysis difficult. Additionally, errors in log parsing can introduce noise into subsequent analyses, reducing the reliability of insights derived from logs.

- Real-time analysis: Logs are generated at enormous scales, often producing millions of entries in short timeframes. Real-time or near real-time analysis is crucial for modern IT operations because any delay in parsing logs can hinder prompt detection and response to system issues. However, this requirement becomes significantly more challenging when systems generate logs at enormous scales,potentially millions of entries within very short timeframes. Processing and parsing such high,throughput data in real time places a heavy computational burden on logging frameworks, creating performance bottlenecks that can slow or even halt downstream analytics. If a log parser cannot keep pace, critical information—such as early warnings of potential failures or performance degradation—may be delayed or missed entirely. This not only increases the risk of outages but also lengthens the time to investigate

and resolve incidents, which can have severe financial and reputational consequences for organizations dependent on uninterrupted services. Solving this bottleneck calls for log parsing methods that are both highly scalable and efficient, ensuring that mission,critical insights remain timely and actionable. Real-time parsing of such volumes is computationally intensive, prone to inefficiencies, and can become a bottleneck in operational analysis.

- Lack of governance in logging practices: There is often no formal process to ensure that log data adheres to standardized practices, leading to variability in quality, structure, and the inclusion of sensitive information, further complicating parsing and analysis.

- Specialized Nature of Log Data: Unlike natural language, log data is composed of semi-structured, system-generated messages that lack the grammatical conventions of human language. As a result, Large Language Models (LLMs) trained on natural language text struggle to adapt effectively to log data, leading to inaccurate or incomplete analysis. Additionally, the underlying syntax and structure of logs are not well understood, with limited research exploring the fundamental composition of log messages, which further complicates their analysis and utilization.

- Introduction of sensitive information: Developers may inadvertently log sensitive user data, including personally identifiable information (PII), due to a lack of governance processes for detecting and managing such content, which can lead to compliance risks and privacy breaches.

- Privacy compliance issues: Developers may not be well-versed in privacy regulations, leading to the inclusion of sensitive information in logs that should not be recorded or made publicly accessible. This can result in non-compliance with privacy laws like The General Data Protection Regulation(GDPR), potentially leading to severe legal consequences.

This thesis closely addresses the challenges inherent in log parsing for AIOps environments, focusing on enhancing the accuracy, efficiency, scalability, and privacy of log data management. Below, we outline the specific solutions introduced in this research to respond to each identified challenge:

- **Heterogeneity of log formats:** The development of ULP (Universal Log Parser) utilizes a frequent token counting approach, effectively distinguishing between static and dynamic elements across diverse log structures, thereby overcoming inconsistencies in log formats.

- **Dynamic nature of logs:** The proposed AML (Accuracy Metric for Log Parsing) is a metric specifically designed to quantify the accuracy of log parsing outputs. In other words, it measures how well a parser reconstructs the correct log templates and accurately identifies dynamic tokens from raw log data, capturing both omission (missing information) and commission (extraneous information) errors. This level of detail makes AML parser-agnostic, allowing it to be applied across different systems without being tied to a single parser's architecture. As a result, AML serves as a more universal measure for comparing various log parsing approaches, selecting the most appropriate parser for a given scenario, or guiding the design of a universal parser. By effectively differentiating between static templates and dynamic tokens, AML provides actionable insights on how to improve log parsing for diverse and evolving AIOps environments

- **Lack of logging guidelines:** The detailed taxonomy of log characteristics provides a reference framework for developers, offering guidelines to standardize logging practices, thereby reducing variability in log levels, structure, and content.

- **Introduction of sensitive information:** This research also contributes to privacy-aware log management by introducing practices to identify and mitigate the logging of sensitive information, promoting compliance with privacy regulations like GDPR.

- **Noisy data and incorrect analysis:** The structured parsing approach of ULP, combined with the evaluative precision of AML, aims to reduce the noise introduced during log analysis, leading to more reliable and effective insights.

- **Scale:** The efficiency of ULP, with reduced computational complexity, enables effective real-time parsing of large-scale log data, minimizing inefficiencies and preventing bottlenecks in operational analysis.

- **Lack of governance in logging practices:** The introduction of a taxonomy and AML framework contributes to a more systematic governance process, ensuring consistency, compliance, and quality in log generation and analysis.

- **Specialized Nature of Log Data:** ULP is specifically tailored to handle the unique semi-structured nature of log data, providing deterministic parsing that LLMs cannot achieve due to their reliance on natural language training, thus bridging the gap in log data analysis capabilities.

- **Privacy awareness:** Through the development of standardized guidelines and a focus on privacy compliance, this research addresses the challenge of developers inadvertently logging sensitive data, contributing to better privacy management in IT operations.

Addressing these challenges is critical for improving the accuracy and effectiveness of log parsing, which in turn is essential for enabling reliable downstream analysis in AI-driven systems. The ability to parse logs consistently and accurately will improve operational insights, support real-time decision-making, and enhance overall system efficiency.

1.1 Contributions of the Thesis

This Ph.D. thesis introduces four key contributions that collectively address persistent challenges in modern log management for AIOps. By advancing the accuracy, performance, privacy, and evaluation of log parsing, these contributions form an integrated framework that not only tackles the limitations of current techniques but also meets the complex demands of diverse, large-scale IT environments.

The first contribution is the development of **ULP (Universal Log Parser)**, an efficient tool designed to overcome the performance bottlenecks found in existing parsers. ULP employs a frequent token count methodology to identify static and dynamic components in log messages with minimal computational overhead. Although large-scale processing and real-time requirements can often conflict since high throughput can slow performance while timely analysis demands speed, ULP strikes a balance by streamlining template extraction. This enables rapid, precise parsing even under heavy

workloads, making it especially valuable for large-scale AIOps environments that rely on timely insights for automated decision-making.

However, improved parsing alone is insufficient without a robust way to evaluate its effectiveness. The second contribution, **AML (Accuracy Metric for Log Parsing)**, fills this gap by providing a parser-agnostic framework for assessing log parsing performance. Traditional metrics often overlook nuanced errors, such as omissions or misclassifications, that can significantly affect downstream analyses. AML addresses this shortfall by offering detailed accuracy measurements at both the log template and log event levels, allowing parsers to be more effectively compared, tuned, and improved. By ensuring that parsing processes feeding AIOps are reliable and precise, AML minimizes false positives or negatives in system analysis.

Building on these foundational elements, the thesis next introduces a **taxonomy of Log Event Characteristics (LECs)** to guide the design and evaluation of robust parsing techniques. By categorizing structural and contextual log attributes, such as nested tokens, inconsistent delimiters, and variable-length fields, this taxonomy illuminates the recurring issues that lead to parsing inaccuracies. Armed with these insights, developers and practitioners can create more adaptable parsers, while also refining their logging practices to reduce ambiguity in data capture.

Finally, the fourth contribution addresses the often-overlooked issue of **privacy compliance** in log data. Modern logs can contain sensitive information like IP addresses, usernames, or other personal identifiers. Through an examination of real-world logs and current privacy regulations, such as GDPR, this research highlights the discrepancies between policy requirements and actual logging practices. The resulting guidelines help ensure that automated log analysis remains aligned with legal and ethical standards, thereby maintaining user trust and mitigating potential risks.

Taken together, these four contributions advance the field of log management by tackling core technical, methodological, and regulatory challenges in a cohesive manner. ULP lays the groundwork for scalable, real-time parsing, AML provides a reliable means of evaluating parser accuracy, the taxonomy of log characteristics offers targeted insights for parser resilience, and privacy compliance ensures ethical, lawful handling of sensitive data. This integrated framework ultimately enables AIOps practitioners to harness log data more effectively, leading to greater operational efficiency, reliability, and trustworthiness across IT environments.

The novelty of this thesis lies in its end-to-end approach to improving log parsing, from increasing efficiency and robustness to incorporating nuanced evaluation metrics and addressing privacy risks, thereby delivering a holistic solution that enhances AIOps at every stage of the log management lifecycle.

Table 1.1: Summary of Thesis Contributions

Category	Challenge	How we Solved the Challenge and Contribution to AIOps
Performance and Scalability	Performance limitations in existing parsers	• Developed a novel **frequent token count methodology** to reduce computational complexity. • Enabled faster, more efficient log parsing in large-scale environments under heavy workloads, which is crucial for AIOps that require rapid response to log-based events. • **Contribution to AIOps:** Enhanced real-time log processing capabilities, allowing Site Reliability Engineers (SREs) to make informed, timely decisions with lower latency, critical for maintaining IT infrastructure health.
Evaluation and Accuracy	Lack of robust metrics to evaluate parsing accuracy	• Introduced **Accuracy Metric for Log Parsing (AML)**, enabling comprehensive evaluation at both the log template and log event levels. • Addressed the limitations of current simplistic metrics, ensuring a more nuanced and accurate assessment of log parsers. • **Contribution to AIOps:** Improved the reliability of automated log analysis through more precise accuracy metrics, directly impacting the ability to identify and diagnose issues effectively.
Log Structure and Parsing Complexity	Difficulty in identifying structural attributes causing parsing failures	• Developed a **taxonomy of Log Event Characteristics (LECs)** that categorizes challenging log attributes and identifies structural complexities. • Provided insights into why certain logs are difficult to parse, enabling the development of more resilient parsers. • **Contribution to AIOps:** Improved log parsing resilience, making AIOps systems more adaptable and effective in analyzing heterogeneous logs from diverse IT environments, leading to better anomaly detection and operational insights.
Privacy Compliance	Risks of handling personal data within log files	• Conducted an evaluation of privacy practices in log management against policies and regulations like GDPR. • Provided guidelines to minimize the capture of sensitive data and ensure responsible log management. • **Contribution to AIOps:** Integrated privacy-aware log management practices within AIOps, enabling organizations to leverage logs for insights while ensuring compliance with privacy regulations. This is critical for maintaining trust and avoiding legal ramifications in the handling of personal data.

1.2 Thesis Structure

This thesis is organized as follows:

- Chapter 2: This chapter provides an in-depth review of existing log parsing tools, examining their capabilities and the inherent limitations within current log management practices. It also emphasizes the need for more robust and comprehensive evaluation metrics to accurately assess log parsing performance.
- Chapter 3: Introduces ULP, detailing its architecture, methodology, and performance evaluation against other state-of-the-art log parsers.
- Chapter 4: Presents the AML accuracy metric, including its theoretical foundation, methodology, and a comparative analysis of 14 log parsers using AML across 16 diverse log datasets.
- Chapter 5: Explores the LECs, providing a detailed analysis of the factors that contribute to parsing errors and recommendations for improving parsing algorithms.
- Chapter 6: Discusses the challenges and contributions related to privacy compliance in log management, including evaluating privacy practices against policies like GDPR, and offering guidelines for responsible log management.
- Chapter 7: Summarizes the key findings and discusses potential future research directions for advancing log management and AIOps.

Chapter 2

Literature Review

The emergence of AIOps can be traced to the need to adapt traditional IT methodologies, such as DevOps, to the complexities of modern infrastructures. DevOps aims to bridge the gap between development and operations by ensuring faster software delivery with high quality through agile practices [6]. However, with new paradigms like the operationalization of AI models and big data analytics [39], traditional DevOps approaches have faced limitations in handling the broader life-cycle and unique requirements of AI projects. AIOps seeks to address these gaps by providing a systematic approach to managing the increasing complexity of IT operations in AI-driven environments [40].

Software logging plays a critical role within AIOps, serving as the primary means of capturing system events during software execution. These log entries provide valuable insights into system behavior, aiding in debugging, anomaly detection, failure analysis, and auditing [41]. Effective logging practices directly contribute to improving AIOps capabilities by providing the raw data needed for effective analysis [42].

Logs play a crucial role in numerous software engineering tasks, including debugging [3, 43, 4], debugging and comprehension of system failures [44][45][46][47][8], testing and performance analysis [48][37] [49], operational intelligence [8] [37][50][5], data leakage detection [51], failure detection [46], failure prediction [8][**?**],anomaly detection [3, 4, 2**?**], and AI-driven IT operations [1, 2]. Their versatility makes logs an indispensable resource for maintaining the reliability, performance, and security of software systems [**?**].

Table 2.1: Summary and Classification of Research Work on Software Logging and Log Parsing.

Study Focus	Characterisation Study	Automation Technique	Machine Learning Model	Evaluation Study
Failure Diagnosis	[61]			
Logging Practice	[32]			
Log Analysis and Failure Diagnosis	[**?**]			
Logging for Anomaly Detection			[62]	
Machine Learning in Logging Practice		[9]		
Log Compression (Parallelization)		[63]		
Log Parsing (Rule-based)		[64, 61]		[61]
Log Parsing (Frequent Token Mining)		[25**?**]		
Log Parsing (Natural Language Processing)		[12]	[13, 14]	
Log Parsing (Clustering/Heuristic)		[65, 66, 29]		
Logging Code Analysis	[59, 60, 67]			
Automation in Logging		[68]		
Logging Configuration	[69]			
Log Statements (Code Level)	[70, 71]	[72]	[73]	
Log Parsing and Metrics		[10]		
Logging Issues (Performance)				[74, 70]

The importance of software logging has driven significant research focused on understanding how logging is implemented in practice. Studies have revealed several shortcomings, including the presence of logging anti-patterns, insufficient guidance on log levels, and inadequate practices regarding log placement [52, 53, 54, 55]. This has led to the development of automation tools that assist in various aspects of logging, such as determining what variables to log, choosing appropriate log levels, and employing machine learning models for automated logging suggestions [56, 57, 58].

Recent years have also seen research leveraging machine learning and deep learning for failure and anomaly detection through log messages [59, 60]. More recently, the focus has shifted towards improving the practice of writing and evolving logging code. Research in this area can be broadly categorized into characterization studies, automation techniques, decision-making frameworks, machine learning models, and evaluation studies, as shown in Table 2.1.

Characterization studies aim to derive insights about logging practices that are beneficial for both researchers and practitioners. On the other hand, automation techniques and decision-making frameworks provide systematic methods to assist developers in generating and managing logging statements. Machine learning-based approaches further contribute by leveraging existing data to make informed suggestions or predictions related to logging. Evaluation studies offer comparisons across different techniques and models to assess their effectiveness. Earlier works, such as those by Jiang et al. [61] and Yuan et al. [32], laid the foundation for automated log analysis, which has since expanded to include modern deep learning approaches like Deeplog for anomaly detection [62].

2.1 Log Parsing Techniques and Their Evolution

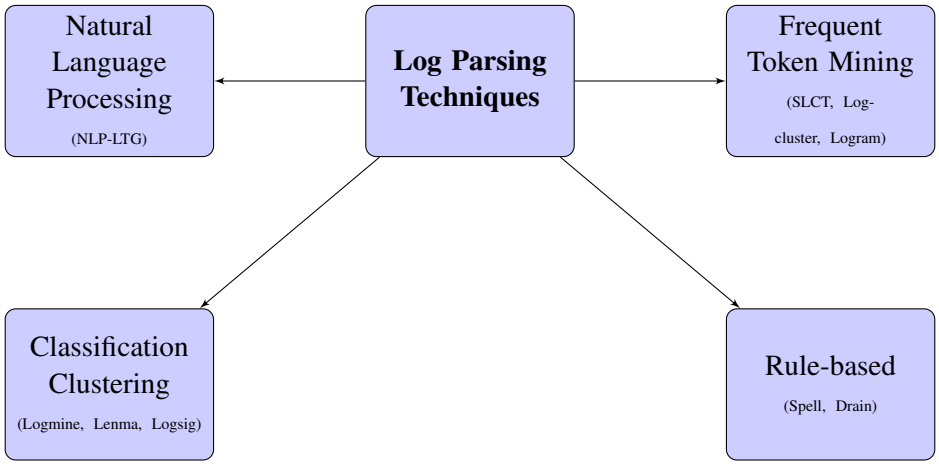

Figure 2.1: Overview of Log Parsing Techniques

The increased reliance on logs for system monitoring and troubleshooting has led to the development of a diverse range of log parsing tools. These tools employ methodologies such as machine learning, rule-based techniques, natural language processing (NLP), and clustering methods. Figure 2.1 illustrates an overview of these techniques.

One of the most comprehensive surveys on log parsing techniques is presented by El-Masri et al. [33], where the authors propose a quality model for classifying log parsing techniques and examine 17 different log parsing tools using this model.

The following subsections discuss several notable parsing methods across these different categories, providing insights into the methodologies employed and their respective challenges.

2.1.1 Rule-based parsing tools

Hamooni et al. proposed **LogMine** [29], which uses MapReduce to abstract heterogeneous log messages generated from various systems. The LogMine algorithm consists of a hierarchical clustering module combined with pattern recognition. It uses regular expressions based on domain knowledge to detect a set of dynamic variables. Then, it replaces the real value of each field with its name. It then clusters similar log messages with the friends-of-friends clustering algorithm.

Drain [64] starts by Pre-processing raw log messages using regular expressions to identify

trivial dynamic tokens. Then, it builds a parse-tree using the number of tokens in log events. Drain assumes that tokens that appears at the beginning of a log message are most likely Static Tokens. It uses a similarity metric that compares leaf nodes to event types to identify log groups.

2.1.2 Frequent tokens mining

Tang et al. proposed **LogSig** [25], which considers the words present in a Log Event as signatures of event types. LogSig identifies log events using a set of predefined message signatures. First, it converts log messages into pairs of terms. Then, it forms log-message clusters using a local search strategy. LogSig selects the terms that appear frequently in each cluster and use them to build the event templates. **Spell** (Streaming Parser for Event Logs using an LCS)[75] is a log parser, which converts log messages into event types. Spell relies on the idea that log messages that are produced by the same logging statement can be assigned a type, which represents their longest common sequence. The LCS of the two messages is likely to be static fields.

2.1.3 Natural language processing tools

Logram, a recent approach proposed by Dai et al. [13], is an automated log parsing approach developed to address the growing size of logs, and the need for low-Latency log analysis tools. It leverages n-gram dictionaries to achieve efficient log parsing. Logram stores the frequencies of n-grams in logs and relies on the n-gram dictionaries to distinguish between the Static Tokens and dynamic variables. Moreover, as the n-gram dictionaries can be constructed concurrently and aggregated efficiently, Logram can achieve high scalability when deployed in a multi-core environment without sacrificing parsing accuracy. In Logram, the identification of dynamic and static tokens depends on a threshold applied to the number of times the n-grams occur. An automated approach to estimating this threshold was proposed.

Kobayashi et al. proposed **NLP-LTG** (Natural Language Processing–Log Template Generation) [12], which considers event template extraction from log messages as a problem of labelling sequential data in natural language. It uses Conditional Random Fields (CRF) [24] to classify terms in log messages as static or dynamic. To construct the labelled data (the ground truth), it uses human knowledge and regular expressions.

Thaler et al.[14] proposed **NLM-FSE** (Neural language Model For Signature Extraction), which trains a character-based neural network to classify static and dynamic parts of log messages. The approach constructs the training model through four layers. (1) The embedding layer transforms the categorical character input into a feature vector. (2) The Bidirectional-LSTM layer allows each prediction to be conditioned on the complete past and future context of a sequence. (3) The dropout layer avoids over-fitting by concatenating the results of the bi-LSTM layer, and (4) the fully connected, feed-forward neural network layer predicts the event template using the Softmax activation function.

2.1.4 Classification/clustering approaches

Jiang et al. [61] introduced **AEL** (Abstracting Execution Logs), which is a log parsing method that relies on textual similarity to group log events together. AEL starts by detecting trivial dynamic variables using hard-coded heuristics based on system knowledge (e.g., IP addresses, numbers, memory addresses). The resulting log events are then tokenized and assigned to bins based on the number of terms they contain. This grouping is used to compare log messages in each bin and abstract them into templates. The problem with AEL is that it assumes that events that contain the same number of terms should be grouped together, resulting in many false positives.

Vaarandi et al. [76] proposed **SLCT** (Simple Logfile Clustering Tool). The authors used clustering techniques to identify log templates. SLCT groups log events together based on their most common frequent terms. To this end, the approach relies on a density-based clustering algorithm to recognize the Dynamic Tokens, SLCT uses frequency analysis across all log lines in the log file.

LogCluster [**?**] is an improved version of SLCT proposed by the same authors. LogCluster extracts all frequent terms from the log messages and arranges them into tuples. Then, it splits the log file into clusters that contain at least a certain number of log messages ensuring that all log events in the same cluster match the pattern constructed by the frequent words and the wildcards, which replace the dynamic variables.

Another clustering approach is that proposed by Makanju et al., which is called **IPLOM** (Iterative Partitioning Log Mining) [65]. IPLOM employs a heuristic-based hierarchical clustering algorithm. Using this approach, the first step is to partition log messages. For this, the authors used

heuristics considering the size of log events. The next step is to further divide each partition based on the highest number of similar terms. The resulting leaf nodes of the hierarchical partitioning as clusters and event types.

Fu et al. proposed **LKE** (Log Key Extraction) [66], another clustering-based approach, using a distance-based clustering technique. Log events are grouped together using weighted edit distance, giving more weight to terms that appear at the beginning of log events. Then, LKE splits the clusters until each raw log level in the same cluster belongs to the same log key and extracts the common parts of the raw log key from each cluster to generate event types.

2.1.5 Challenges for Log Parsing

Despite the progress in automated log parsing, challenges remain due to the lack of standardized guidelines for logging. This variability in log structures across different systems, combined with the continuously increasing volume of log data, complicates log parsing, often limiting the effectiveness of system diagnostics and maintenance [31, 32, 33, 37, 38]. Achieving high parsing accuracy across diverse log formats remains difficult, largely because much of the research has focused on developing parsing algorithms without sufficient emphasis on understanding the causes of parsing errors [19, 77, 20, 13].

Inaccurate log parsing or a high number of parsing errors can lead to misinterpretations of system behavior, causing delays in diagnosing issues, increased maintenance costs, and, in some cases, detrimental consequences for the overall performance and reliability of the software system [78]. Even marginal inaccuracies in parsing, as slight as 4% of the total, can have profound implications on performance, potentially leading to repercussions magnified by an entire order of magnitude [79].

Le et al. [80] illustrated the adverse effects of log parsing errors on log-based Anomaly Detection. Fu et al. [81] delved deeper into this, examining how different log parser errors affect anomaly detection, emphasizing the importance of parsing accuracy and the event template count in choosing the right log parsers.

2.2 Logs privacy

Despite their utility, logs frequently capture personal data, including user IDs, email addresses, IP and MAC addresses. This inadvertent capture during routine operations raises privacy concerns, especially when logs are shared externally [82]. Developers may occasionally log information, often for debugging purposes, and inadvertently leave this information within the code base. This poor coding practice poses a potential risk to data privacy since logs may contain personal data without due diligence[51]. The outsourcing of log streams to cloud vendors introduces additional privacy concerns due to potential access by third parties and the need for increased data privacy measures[82].

The concept of personal data in privacy law is broadly defined to include any information that can identify an individual, either directly or indirectly [83, 84].The General Data Protection Regulation (GDPR), defines personal data as `any information relating to an identified or identifiable natural person` (Article 4, GDPR). This definition supports the scope of our analysis, ensuring consistency and compliance with widely recognized standards [85]. Introduced by the European Union (EU) in 2018, GDPR is perhaps the most comprehensive data privacy law in existence today [86, 85, 87]. GDPR contains several provisions that regulate the handling of personal data of EU residents. Companies operating outside the EU zone are also subject to GDPR regulations when handling the personal data of EU residents [83]. Non-compliance with GDPR can result in severe fines [87, 86, 85]. GDPR consists of a data privacy legal framework guided by principles, such as purpose limitation, data minimization, accuracy, storage limitations, integrity, confidentiality, accountability, lawfulness, fairness, and transparency [88, 85]. Incorporating these legal obligations into software development is a complex task[89, 90], particularly when it comes to data logging. As noted by Portillo et al. [82], organizations must address the challenging problem of deciding on the types of data that can be logged, justifications for logging, and how to protect individuals' privacy rights while leveraging log data insights. This is further complicated by the lack of awareness within the software development teams on the importance of regulatory compliance [91, 92].

This definition was highlighted in the landmark Breyer case, where IP addresses were recognized as personal data when linked with other identifiable information [93]. This ruling clarifies the privacy considerations surrounding IP addresses in logs, affirming their status as personal data. User perceptions of IP addresses as highly sensitive are documented in studies [94], with concerns exacerbated by prevalent IP tracking practices [95, 96]. The treatment of IP addresses in logs, thus, becomes a critical aspect of maintaining user privacy.

Several studies have explored the security aspects of system logging and proposed guidelines for secure logging practices. Zeng et al. [97] discuss a range of security vulnerabilities that arise from system logging and emphasize the necessity for robust logging mechanisms to prevent data breaches and maintain system integrity.

In a more focused study, Patel et al. [98] delve into the specifics of logging within the Linux kernel suggesting improvements for effective logging that enhances both security and performance monitoring [98]. Lyons et al. [99] examine the extent of sensitive information logging within the Android ecosystem. Their study measures the prevalence of sensitive data in logs and discusses the implications for privacy and security, stressing the need for stringent controls on what is logged and how it is managed.

2.3 LLMs for AIOps

Large Language Models (LLMs) are AI systems designed to perform a wide range of natural language processing tasks, such as translation, summarization, and text generation, thanks to training on vast amounts of data [100].

Large Language Models (LLMs) have become crucial in AIOps for tasks such as log analytics, anomaly detection, and root cause analysis. LLMs are designed to handle a wide range of natural language processing (NLP) tasks, thanks to training on massive datasets, and their capabilities are being repurposed for IT management. In particular, LLMs are being used for automated logging statement generation [101], log parsing [102, 103], and root cause analysis [104].

LLMs can be categorized into two main types: autoregressive models, such as the GPT series, and masked token models like BERT [100]. Autoregressive models are particularly suited for log

parsing because they generate predictions sequentially, allowing them to understand the flow of information across log messages [105]. Masked token models, on the other hand, are valuable for anomaly detection because they predict missing components within a sequence, thereby providing insights into patterns of failures or anomalies [105, 106].

The use of general-purpose LLMs like GPT-4 has shown promising results in log analytics, particularly for identifying anomalies and understanding complex log structures [107]. However, integrating these models into existing log analysis pipelines can pose challenges. Issues such as tool compatibility, and the high cost associated with processing large log datasets [103] limit the broader adoption of such models in practical settings [108].

Another notable limitations of LLMs lies in their probabilistic nature, which often results in inconsistent and sometimes inaccurate outputs when handling structured log data. Prior studies have reported that pre-trained LLMs do not perform well on domain-specific tasks, underscoring the importance of understanding the unique vocabulary and structure of log messages to enhance log analysis [42]. Unlike natural language, logs are semi-structured, system-generated data that lack the grammatical conventions of human language. The inconsistency of LLMs in processing log data limits their effectiveness in performing deterministic tasks such as log parsing. Therefore, while LLMs are effective at extracting insights and performing high-level analytics, they struggle to achieve the structured accuracy needed for in-depth log parsing, often resulting in errors when processing complex and diverse logs at scale [102, 101].

Recent research has introduced specialized LLMs that focus on log data to overcome some of these challenges. For instance, [42] presents an LLM specifically trained on both public and proprietary log data, demonstrating superior performance in multiple downstream log analysis tasks. However, even these specialized LLMs face challenges when it comes to parsing heterogeneous log formats with high accuracy[109].

To address these challenges, my thesis introduces approaches that complement the capabilities of LLMs while mitigating their shortcomings in structured log analysis. Specifically, my research explores how Log Event Characteristics (LECs) can be used in conjunction with LLM training datasets to enhance the models' understanding of log data effectively. By embedding LECs into the training data, LLMs can be better informed about the underlying structure of log messages. This

guidance enhances LLMs' ability to generate more accurate and contextually relevant interpretations of log events. The structured nature of LECs complements the probabilistic capabilities of LLMs, thereby enhancing both the accuracy and reliability of log parsing and analysis.

Chapter 3

ULP - A Universal Log Parser

"Simplicity is the ultimate sophistication."

— Inspired by Leonardo Da Vinci

I. Sedki, A. Hamou-Lhadj, O. Ait-Mohamed and M. A. Shehab, "An Effective Approach for Parsing Large Log Files," in Proc. of the 38th IEEE International Conference on Software Maintenance and Evolution (ICSME), Limassol, Cyprus, 2022, pp. 1-12, doi: 10.1109/ICSME55016.2022.00009.

Abstract : Because of their contribution to the overall reliability assurance process, software logs have become important data assets for the analysis of software systems. Logs are often the only data points that can shed light on how a software system behaves once deployed. Unfortunately, logs are often unstructured data items, hindering viable analysis of their content. There are studies that aim to automatically parse large log files. The primary goal is to create templates from raw log data samples that can later be used to recognize future logs. In this paper, we propose ULP, a Unified Log Parsing tool, which is highly accurate and efficient. ULP combines string matching and local frequency analysis to parse large log files in an efficient manner. First, log events are organized into groups using a text processing method. Frequency analysis is then applied locally to instances of the same group to identify static and dynamic content of log events. When applied to 10 log datasets of the LogPai benchmark, ULP achieves an average accuracy of 89.2%, which outperforms the accuracy of four leading log parsing tools, namely Drain, Logram, SPELL and AEL. Additionally, ULP can parse up to four million log events in less than 3 minutes. ULP is available online as an open source and can be readily used by practitioners and researchers to parse effectively and efficiently large log files so as to support log analysis tasks.

3.1 Introduction

Logs are generated by logging statements inserted by developers in the source code. An example of a logging statement is shown in Figure 1, which is a code snippet extracted from a Hadoop Distributed File System (HDFS) Java source file. The generated log event (Figure 1) is composed mainly of two parts: the log header and log message. The log header typically contains the timestamp (e.g., 081109 283519), the process id (147), the log level (INFO), and the logging function

(dfs.DataNodePacketResponder). The log message contains static tokens (usually text) such as "Received block", "size of", "from" in the example of Figure 1 and dynamic tokens, which represent variable values (`blk_-1680999687919862986, 91178, /10.250.14.224`).

Log files include a plethora of information on the execution of a software system that is used to help with different software engineering activities, such as anomaly detection [110][111][112][113], debugging and comprehension of system failures [44][45][46][47][8], testing and performance analysis [48][37] [49], operational intelligence [8] [37][50][5], failure prediction [8][**?**], detection of data leakage [31], and more recently, AI for IT Operations (AIOps) [5][114].

Logs, on the other hand, have traditionally been difficult to work with. Typical log files may be considerably large (in the order of millions of events) [47][115][116]. Furthermore, the logging practice is known to be ad hoc, with no defined best practices [33]. To make matters worse, logs are mainly unstructured data files due to the lack of a standardized format [110][**?**][116]. As a result, extracting relevant information from large raw log files [36][39] can be a daunting and time-consuming task.

In this paper, we focus on the problem of log parsing, which consists of (semi-)automatically converting unstructured raw log events into a structured format that would facilitate future analysis. Log parsing is further reduced to the problem of parsing log messages. This is because log headers usually follow the same format within the same log file. Parsing a log message consists of automatically distinguishing the static text from the dynamic variables. The result of parsing the log event in Figure 1 is the extraction of the template in `Received Block <*> of size <*> from`

Logging statement: LOG.info("Received Block"+ block + "of size" + block_size + " from" + sender_ip)

Raw log line: 081109 283519 147 INFO dfs.DataNodePacketResponder: Received block blk_-1680999687919862986 of size 91178 from /10.250.14.224

Log template: Received Block <*> of size <*> from <*>

Figure 3.1: An example of a logging statement, the generated log event, and the expected log template

<*>, which identifies the log message structure. One way to parse log events would be to use regular expressions [117][61]. The problem is that typical industrial log files may contain hundreds of log templates as shown by Chow et al. [50] and Mi et al. [118]. Furthermore, as the system evolves, new log formats are produced due to the use of multiple logging libraries, necessitating the ongoing updating of the regular expressions. The use of regular expressions to parse various types of logs is simply impractical, which has led researchers to develop more intelligent log parsing techniques (see [33] for a good survey). Existing approaches, however, suffer from many limitations [33] including their reliance on domain knowledge, inability to demarcate static content from dynamic variables for complex log files, and use of advanced machine learning algorithms, which require parameter tuning.

In this paper, we propose ULP (Unified Log Parser), a simple yet powerful approach that recognizes log structures from any log files without prior domain knowledge or the use of complex machine learning techniques. ULP relies on string matching and local frequency analysis. It begins by grouping similar log events into groups using a string matching similarity technique. It then uses frequency analysis on instances of each group to distinguish between static and dynamic log message tokens. We conjecture that tokens that are more often repeated in the same group of similar log events are more likely static tokens than dynamic tokens. This is not the first time that frequency analysis is used in log parsing. Other tools such as Drain [64] and Logram [13] also use frequency analysis. However, these tools apply frequency analysis to the entire log file, which makes it hard to find a clear demarcation between static and dynamic tokens. ULP, on the other hand, applies frequency analysis to log events that belong to the same group rather than to the entire log dataset, increasing the likelihood of distinguishing between static and dynamic tokens.

We compared ULP to four major log parsing tools, namely Drain [64], AEL [61], SPELL [75], and Logram [13] by applying them to 10 log datasets from the LogPai benchmark[1]. Our findings show that ULP outperforms existing tools in parsing all the log datasets. Our technique has an average accuracy of 89.2 %, while the second-best method, Drain, has an average accuracy of 73.7 %. Furthermore, ULP can parse big files containing up to 4 million log events in under 3 minutes.

[1] https://github.com/logpai

ULP is available as an open source tool and accessible online[2]. Practitioners can readily embed ULP into their log analytic suite and not have to worry about creating parsers that are tailored to specific log files, which we believe may result in improved productivity and better software maintenance.

The organization of this paper is as follows: Section II looks at the actual approaches used in log parsing and how they compare to our solution. We present the ULP approach in Section III. Section IV focuses on the evaluation of ULP using 10 log files. Section V highlights the distinctiveness of our technique and explores the reasons why ULP outperforms other comparable approaches, followed by threats to validity. We conclude the paper in Section VII and discuss future directions.

3.2 Related Work

In recent years, log analysis has received a great deal of attention from researchers and practitioners due to the increasing need to understand complex systems at runtime. One of the most comprehensive survey of log parsing techniques is the one proposed by El-Masri et al. [33] in which the authors proposed a quality model for classifying log parsing techniques and examined 17 different log parsing tools using this model. Existing tools can be categorized into groups based on the techniques they use, namely rule-based parsing tools, frequent token mining, natural language processing, and classification/clustering approaches. We discuss the main approaches in what follows and conclude with a general comparison of ULP with these techniques.

Jiang et al. [119] introduced AEL (Abstracting Execution Logs), which is a log parsing method that relies on textual similarity to group log events together. AEL starts by detecting trivial dynamic tokens using hard-coded heuristics based on system knowledge (e.g., IP addresses, numbers, memory addresses). The resulting log events are then tokenized and assigned to bins based on the number of terms they contain. This grouping is used to compare the log messages in each bin and abstracts them into templates. The problem with AEL is that it assumes that events that contain the same number of terms should be grouped together, resulting in many false positives.

Vaarandi et al. [26] [120] proposed SLCT (Simple Logfile Clustering Tool). The authors used

[2]http://zenodo.org/record/6425919

clustering techniques to identify log templates. SLCT groups log events together based on their most common frequent terms. To this end, the approach relies on a density-based clustering algorithm to recognize the dynamic tokens, SLCT uses frequency analysis across all log lines in the log file. LogCluster [114] is an improved version of SLCT proposed by the same authors. LogCluster extracts all frequent terms from the log messages and arranges them into tuples. Then, it splits the log file into clusters that contain at least a certain number of log messages.

Another clustering approach is the one proposed by Makanju et al., which is called IPLOM (Iterative Partitioning Log Mining) [65]. IPLOM employs a heuristic-based hierarchical clustering algorithm. In this approach, the first step is to partition the log messages. For this, the authors used heuristics considering the size of log events. The next step is to further divide each partition based on the highest number of similar terms. Fu et al. proposed LKE (Log Key Extraction)[121], another clustering-based approach, using a distance-based clustering technique. Log events are grouped together using weighted edit distance, giving more weight to the terms that appear at the beginning of log events. Then, LKE splits the clusters until each raw log level in the same cluster belongs to the same log key and extracts the common parts of the raw log key from each cluster to generate event types.

Hamooni et al. proposed LogMine [29], which uses MapReduce to abstract heterogeneous log messages generated from various systems. The LogMine algorithm consists of a hierarchical clustering module combined with pattern recognition. It uses regular expressions based on domain knowledge to detect a set of dynamic variables. Then, it replaces the real value of each field with its name. It then clusters similar log messages with the friends-of-friends clustering algorithm.

Natural Language Processing (NLP) techniques have also been used for log parsing. Logram, a recent approach proposed by Dai et al. [13], is an automated log parsing approach developed to address the growing size of logs, and the need for low-latency log analysis tools. It leverages n-gram dictionaries to achieve efficient log parsing. Logram stores the frequencies of n-grams in logs and relies on the n-gram dictionaries to distinguish between static tokens and dynamic variables. Moreover, as the n-gram dictionaries can be constructed concurrently and aggregated efficiently, Logram can achieve high scalability when deployed in a multi-core environment without sacrificing parsing

accuracy. Kobayashi et al. proposed NLP-LTG (Natural Language Processing–Log Template Generation) [12], which considers event template extraction from log messages as a problem of labeling sequential data in natural language. It uses Conditional Random Fields (CRF) [122] to classify terms in log messages as static or dynamic. To construct the labeled data (the ground truth), it uses human knowledge and regular expressions. Thaler et al.[14] proposed NLM-FSE (Neural language Model-For Signature Extraction) , which trains a character-based neural network to classify static and dynamic parts of log messages.

He et al. [64] proposed Drain, a tool that abstracts log messages into event types using a parse tree. Drain algorithm consists of five steps. Drain starts by preprocessing raw log messages using regular expressions to identify trivial dynamic tokens, just like ULP. Then, it builds a parse tree using the number of tokens in log events. Drain assumes that tokens that appear in the beginning of a log message are most likely static tokens. It uses a similarity metric that compares leaf nodes to event types to identify log groups.

Spell (Streaming Parser for Event Logs using an LCS)[75] is a log parser, which converts log messages into event types. Spell relies on the idea that log messages that are produced by the same logging statement can be assigned a type, which represents their longest common sequence. The LCS of the two messages is likely to be static fields.

The main difference between ULP and existing approaches lies in the way ULP is designed. ULP leverages the idea that static and dynamic tokens of log events can be better identified if we use frequency analysis locally on instances of log events that belong to the same group. To this end, it uses a string matching technique to create groups of similar events. This is contrasted with techniques that use clustering alone (e.g., AEL and IPLOM) and those that apply frequency analysis to the entire log file (e.g., Drain and Logram), i.e., globally. As we will see in the evaluation section, these design choices make ULP a very accurate and efficient log parser compared to leading open source log parsers.

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

2  081109 203807 222 INFO dfs.DataNode$PacketResponder: PacketResponder 0 for block
   blk_-6952295868487656571 terminating

3  081109 204005 35 INFO dfs.FsNameSystem: BLOCK* NameSystem.addStoredBlock: blockMap
   updated: 10.251.73.220:50010 is added to blk_7128370237687728475 size 67108864

4  081109 204015 308 INFO dfs.DataNode$PacketResponder: PacketResponder 2 for block
   blk_8229193803249955061 terminating

5  081109 208106 329 INFO dfs.DataNode$PacketResponder: PacketResponder 2 for block
   blk_-6670958622368987959 terminating

6  081109 204132 26 INFO dfs.FsNameSystem: BLOCK* NameSystem.addStoredBlock: blockMap
   updated: 10,251.43.115:50010 is added to blk_3050920557425079149 size 67105864

7  081109 204328 34 INFO dfs.FsNameSystem: BLOCK* NameSystem.addStoredBlock: blockMap
   updated: 10,251.203.80:50010 is added to blk_7688946331004732825 size 67105864

8  081109 201453 24 INFO dfs.FsNameSystem: BLOCK* NameSystem.addStoredBlock: blockMap
   updated: 10.250.11.85:50010 is added to blk_2377150260125000006 size 67108064

9  081109 204525 512 IFO dfs.DatatlodefPacketResponder: PacketResponder 2 for block
   blk_572492839787299681 terminating

10 081109 201655 556 INFO dfs.DataNode$PacketResponder: Received block
   blk_3587505140051952248 of size 67IO8864 from /10.251.42.84

11 081109 204722 567 INFO dfs.DataNode$PacketResponder: Received block
   blk_5407003568334525940 of size 67108864 from /10.251.214.112

12 081109 204815 653 INFO dfs.DataNode$PacketResponder: Received block
   blk_9212264480425680329 of size 67108864 from /10.251.214.111
```

Figure 3.2: HDFS log events used as a running example

3.3 Approach

ULP consists of the following steps: pre-processing, grouping of similar log events, and the generation of log templates using local frequency analysis. The first step is a pre-processing step where the header such as the timestamp, the log level, and the logging function are identified. We also detect trivial dynamic tokens such as IP and MAC addresses based on common regular expressions. The second step of ULP is to identify similar log events and group them together. To this end, we use text similarity measures. Once the groups of similar log events are formed, we apply frequency analysis to instances of each group to determine the static and dynamic tokens, and derive the various log templates that are then mapped back to each log event. Algorithm 1 shows the steps of ULP. We explain each step in more detail in the following subsections. To illustrate our approach, we use the sample log events from the HDFS log dataset (one of the datasets used to evaluate ULP) shown in Figure 2. We added a line number to each log event to help explain the approach.

3.3.1 Pre-processing

The pre-processing of log events begins by delineating the header information, including the timestamp, process ID, log level, and logging function (Lines 1–3, Algorithm 1). Prior research showed that this information can be readily identified using simple regular phrases [13]. Figure 2 shows that all the HDFS log events of the running example begin with a timestamp (e.g., 081109 203615), a process ID (148), a log level (INFO), and a logging function `dfs.DataNode$PacketResponder`. Another essential part of the pre-processing step is the identification of trivial dynamic variables such as IP and MAC addresses. Identifying such variables can improve the parsing accuracy as shown by He et al. [123] and all the tools studied in this paper (i.e., Drain [64], SPELL [75], Logram [13] and AEL [61]) include this step in their approach. For ULP, this step also increases the chances of identifying similar log events that should be instances of the same group. Grouping of similar events is discussed in more detail in the next subsection. We created regular expressions to detect the following trivial dynamic variables: Mac addresses, IPV6 addresses, URLs (beginning with HTTP and HTTPS), numerical in hexadecimal format, Dates such

Algorithm 1 Overall ULP Algorithm

Data: $LogEvents$
Result: $LogTemplates$, $GroupOfLogs$
foreach $LogEvent \in RawLogfile$ **do**
    $LogEvent \leftarrow RemoveHeader(LogEvent)$
        $LogEvent \leftarrow RemovePunctuation(LogEvent)$
        $LogEvent \leftarrow RemoveObviousDynamicTokens(LogEvent)$
        /* Generate an id for a Log event based on the tokens that are not deemed dynamic, and the count of tokens */
    $LogEvent.Id \leftarrow GenerateIds(LogEvent)$
end
$GroupOfLogs \leftarrow LogEvents.GroupBy(LogEvent.Id)$ $LogTemplates \leftarrow []$

foreach $g \in GroupOfLogs$ **do**
    /* Dimensions reduction to improve algorithm performance */
    $subGroup \leftarrow Sample(g)$
        /* The template will be constructed base on one of the Log events by removing dynamic tokens and replacing them by <*> */
    $template \leftarrow subGroup[0]$
        /* Get the group vocabulary */
    $Tokens \leftarrow GetVocabulary(subGroup)$
        /* Count occurences avoid duplicate for each Log event */
    $subGroup.CountTokensAvoidDuplicatePerEvent()$
     foreach $t \in Tokens$ **do**
        /* static tokens appear all the time in the subgroup, we remove dynamic tokens out of the template */
        if $t.count < subGroup.length$ **then**
            $p \leftarrow template.Remove(t)$
        end
        $LogTemplates.Append(template)$
         $g.template = template$
    end
    $GroupOfLogs \leftarrow mergeGroupWithSimilarTemplate(GroupOfLogs)$
end

as `2002-03-24` and `2002-03-24`, Time in the format `hh:mm:ss`. These regular expressions can be found in ULP code repository[3]. Furthermore, ULP allows users to define additional regular expressions to represent domain-specific variables. It is worth mentioning that all the regular expressions used by ULP are generic and independent from the log files to be parsed, unlike other tools that include regular expressions for detecting log-specific variables at the risk of limiting its generalization. For instance, `r'blk_(|-)[0-9]+'` is used in Logram to detect block ids within HDFS log files.

The result of applying the pre-processing step to the HDFS running example is shown in Figure 3.3, where the header information is detected and removed, and the IP addresses in Lines 3, 6, 7, 10, 11, and 12 are replaced by <*>, indicating that they are dynamic tokens.

```
1  PacketResponder 1 for block blk_38865049064139660 terminating
2  PacketResponder 0 for block blk_-6952295868487656571 terminating
3  BLOCK* NameSystem.addStoredBlock: blockMap updated: <*> is added to
   blk_7128370237687728475 size 67108864
4  PacketResponder 2 for block blk_8229193803249955061 terminating
5  PacketResponder 2 for block blk_-6670958622368987959 terminating
6  BLOCK* NameSystem.addStoredBlock: blockMap updated: <*> is added to
   blk_3050920557425079149 size 67105864
7  BLOCK* NameSystem.addStoredBlock: blockMap updated: <*> is added to
   blk_7688946331004732825 size 67105864
8  BLOCK* NameSystem.addStoredBlock: blockMap updated: <*> is added to
   blk_2377150260125000006 size 67108064
9  PacketResponder 2 for block blk_572492839787299681 terminating
10 Received block blk_3587505140051952248 of size 67I08864 from <*>
11 Received block blk_5407003568334525940 of size 67108864 from <*>
12 Received block blk_9212264480425680329 of size 67108864 from <*>
```

Figure 3.3: Results of pre-processing the HDFS log events example

3.3.2 Grouping similar log events

The second step of ULP is to group similar log events together (Lines 4 to 6 of Algorithm 1). This grouping will help later distinguish between the static and dynamic tokens by applying frequency analysis to instances of each group. For example, the log messages of Lines 1, 2, 4, 5,

[3]http://zenodo.org/record/6425919

and 9 all deal with terminating `PacketResponder` (used by HDFS to manage the processing of data into a series of pipeline nodes) and only vary in terms of the task_number and the block_id. So if we can put these log messages into the same group, we can easily see that the static tokens (`PacketResponder, for, block, terminating`) appear in all the log events of that group, and that the dynamic tokens, i.e., the `task_number` and the `block_id`, vary from one log event to another, hence the idea of using frequency analysis on instances of the same group instead of applying it to the entire log file, which may not lead to such a clear demarcation.

Our grouping strategy relies on a string matching technique. We measure the similarity of two log events based on the number of tokens they contain combined with the tokens that are most likely static tokens (i.e., tokens that do not contain digits and/or special characters). To do this, for each log event, we first count the number of words it contains. A word is defined as any token that is delimited by a space character. Note we do not use any other delimiters to prevent splitting a dynamic variable into multiple tokens. For example, the token `04:09:19.989` is considered by ULP as one token. Then, we identify tokens that only contain alphabetical letters. In other words, we ignore tokens that contain digits and/or special characters, which are most likely dynamic tokens. Finally, we convert the log event into a string that results from concatenating the alphabetical tokens and the total number of tokens. Two log events are grouped together if there is an exact match (i.e., 100%) between their corresponding strings. For example, the two following log events: "`block x23 from 125.12.1.1 allocated to block x45`" and "`block x23 from 125.125.123.144 allocated to block x125`" will be grouped together since they contain the same number of tokens (8) and the same alphabetical tokens and that also appear in the same order `block from allocated to block`.

Another alternative design would be to consider similar but not necessarily identical strings. For this, we would need to define a threshold beyond which two log events can be deemed similar. Setting such a threshold may be a difficult task since it may vary from one group of events to another. We deliberately opted for a grouping technique that requires an exact match to prevent the use of thresholds, which may necessitate human intervention or advanced statistical methods to determine the right threshold.

Applying the grouping approach to the log messages of Figure 3.3 results in three groups. The

first one consists of log messages 1, 2, 4, 5, and 9, which contain the static token `PacketResponder`. The second pattern consists of log messages 3, 6, 7, and 8, representing the message `BLOCK* NameSystem.addStoredBlock: blockMap update`. The last one contains log messages 10, 11, and 12 for the `Received block` log event. At this stage, we have only identified the groups. The next step will leverage local frequency analysis to detect the dynamic tokens and associate a template to log events within each group.

3.3.3 Generation of log templates using local frequency analysis

In this step, we analyze the occurrences of the tokens of each group of log events by counting the number of times each token appears in the log events that belong to the same group (Lines 7–16 of Algorithm 1). As explained earlier, we expect to see static tokens appear all the time in each log event of the same group (because of the way our grouping technique works), while the dynamic tokens are expected to occur only in some log events and not all of them. Therefore, we consider anything that occurs less that the maximum frequency as a dynamic token. It should be noted that in our approach, we are more interested in the occurrences of a dynamic token over several log events than in a single log event. Counting the same token twice in the same log event raises its frequency across several log events in the same group, which introduces a bias. Duplicates in the same log event are counted only once to prevent this bias. For example, if the token `block` occurs twice in the same log event, it will be counted as one occurrence in this event, not two. Table 1 displays the frequency of the tokens of log events of the first group, which consists of the following events:

(1) `PacketResponder 1 for block blk_38865049064139660 terminating`

(2) `PacketResponder 0 for block blk_-6952295868487656571 terminating`

(3) `PacketResponder 2 for block blk_8229193803249955061 terminating`

(4) `PacketResponder 2 for block`

Table 3.1: Example of a frequency analysis result applied to the first group of log events

Term	Frequency	Classification
PacketResponder	5 out of 5	static token
0	1 out of 5	dynamic token
1	1 out of 5	dynamic token
2	3 out of 5	dynamic token
for	5 out of 5	static token
block	5 out of 5	static token
blk_38865049064139660	1 out of 5	dynamic token
blk_-6952295868487656571	1 out of 5	dynamic token
blk_8229193803249955061	1 out of 5	dynamic token
blk_-6670958622368987959	1 out of 5	dynamic token
blk_572492839287299681	1 out of 5	dynamic token
terminating	5 out of 5	static token

`blk_-6670958622368987959 terminating`

(5) `PacketResponder 2 for block`

`blk_572492839287299681 terminating`

In this group, the terms `PacketResponder, for, block, terminating` appear five times (the maximum frequency). The other tokens (task numbers 0, 1, 2, and block ids `blk_38865049064139660`, `blk_-6952295868487656571`, `blk_8229193803249955061`, `blk_-6670958622368987959`, `blk_572492839287299681`) appear less than five times. ULP classifies them as dynamic tokens.

The resulting template from applying local frequency analysis to this group of events is: `PacketResponder <*> for block <*> terminating`. The generated log templates when applying local frequency analysis to the log events of the running example are shown below. Except for `67108864` (the size of a data block in HDFS), ULP was able to uncover all the static and dynamic tokens. ULP was no able to detect the dynamic token `67108864` because we are examining a small sample of log events. In practice, the application of ULP to large HDFS log files should be able to detect this variable at some point in time when a different variable appears in another log event of the same group.

(1) `PacketResponder <*> for block <*> terminating`

(2) `BLOCK* NameSystem.addStoredBlock:`

`blockMap updated: <*> is added to <*> size 67108864`

(3) `Received block <*> of size 67108864 from <*>`

3.4 Evaluation

In this section, we evaluate the effectiveness of ULP in parsing logs of 10 log datasets of the LogPai benchmark [117] available online[4]. The datasets consist of a collection of log files, generated from various systems including Apache, HPC, HDFS as shown in Table 3.2. They are used extensively in the literature (e.g., Drain [64], SPELL[75], and Logram[13]).

This evaluation aims to answer the following two research questions:

(1) RQ1. What is the accuracy of ULP in parsing the 10 log files of the LogPai benchmark and how does it compare to leading log parsing tools?

(2) RQ2. What is the efficiency of ULP and how does it compare to leading log parsing tools?

Table 3.2: LogPai datasets

Datasets	**Description**	**Size**
Apache	Apache server error log	5.1MB
BGL	Blue Gene/L supercomputer log	743MB
HDFS	Hadoop distributed file system log	1.47GB
Hadoop	Hadoop mapreduce job log	2MB
HPC	High performance cluster	32MB
Proxifier	Proxifier software log	2.42MB
Spark	Spark job log	166MB
Thunderbird	Thunderbird supercomputer log	29.60GB
Openstack	OpenStack software log	41MB
Zookeeper	ZooKeeper service log	10MB

We evaluated ULP using accuracy and efficiency. We also compared ULP to four leading log parsing tools, namely Drain[64], AEL [61], SPELL[75] and Logram[13]. We selected these tools

[4]https://zenodo.org/record/3227177#.YUqmXtNPFRE

because prior studies [117][13] [123] showed that these tools yield the highest accuracy and efficiency compared to other log parsing tools such as SLCT. We ran the same experiments with the selected log parsers using the most recent version of their publicly accessible source code.

3.4.1 Accuracy

Each log dataset of the LogPai benchmark used in this study comes with a subset of 2,000 log events that have been parsed manually by the LogPai team. The log templates were identified and each log event out of the 2,000 events was associated with a specific log template. This ground truth dataset is meant for researchers to test their Log parsers and has been used by many log parsing tools such as Drain [64], AEL[61], Lenma [124], IPLoM[65], and Logram [13]. We also use it in this study to evaluate ULP and to compare ULP with existing tools. Table 4.9 shows an example of events from the Apache 2,000 labelled log events where a log event (represented here by an id starting with "E") is mapped to a template that was extracted manually by the LogPai team.

Table 3.3: An example of manually labeled log events from the Apache log dataset

Event ID	**Event Template**
E1	jk2_init() Found child <*>in scoreboard slot <*>
E2	workerEnv.init() ok <*>
E3	mod_jk child workerEnv in error state <*>
E4	[client <*>] Directory index forbidden by rule: <*>
E5	jk2_init() Can't find child <*>in scoreboard
E6	mod_jk child init <*><*>

The way accuracy is measured in related studies is based on the work of Zhu et al. [117] where the authors compared the accuracy of 13 log parsing tools including some of the tools used in this paper (Drain, AEL, and Spell). Logram, which was published after the work of Zhu et al. [117], also uses the same approach. Zhu et al.'s accuracy metric is based on the number of log events that are correctly identified as belonging to the same template when compared to the ground truth. This metric, however, does not check if the template in question is the same as the one in the ground truth. In our opinion, this metric is misleading since it does not assess the ability of a log parser to extract the exact templates as the ones in the ground truth. In other words, Zhu et al.'s metric is necessarily but not sufficient. In this paper, we go one step further by not only comparing if the

Table 3.4: An example that shows how we measure the accuracy of ULP

Ground truth Template	**ULP Generated Template**	**Match or Not**	**Explanation**
Cannot open channel to <*>at election address /<*>:<*>	Cannot open channel to <*>at election address <*>]	1	Static text is detected correctly. Dynamic tokens are identified. the gap is that ground truth is interpreting IP address with port as 2 variables and the parsing tool as one, which is acceptable.
Expiring session <*>, timeout of <*>ms exceeded	[expiring session <*>timeout of 10000ms exceeded]	0	Only one dynamic token has been identified out of two. The parsing is then considered incorrect.

log events are correctly classified as having the same template, but also checking that the templates we extract are exactly the same as the ones in the ground truth. More precisely, to measure the accuracy of ULP, we compare the templates extracted by ULP to those provided by LogPai for the 2,000 log events of the log datasets shown in Table 3.2. The accuracy is the number of matches divided by 2,000. We apply the same procedure to other log parsing tools and compare our results to theirs. A match is considered if the following requirements are satisfied: (1) all static tokens are detected and displayed in the correct location in the ground truth file; (2) all dynamic variables were identified and replaced by <⋆>; (3) all dynamic variables are shown in the same order as the ground truth; (4) no extra static or dynamic tokens were added. Table 3.4 shows an example of the manual comparison we performed.

Results

Table 3.5 shows the results of ULP accuracy. **ULP has the best accuracy in parsing all the log datasets** in comparison to all the other tools assessed in this study (i.e., Logram, SPELL, AEL, and Drain). Additionally, our approach has an **average accuracy of 89.2%**, whereas the second most

Table 3.5: Accuracy of ULP compared to other Log parsers. highest results bold.

Name	Drain	Logram	Spell	AEL	ULP
HDFS	0.996	0.981	0.500	0.434	**0.999**
Apache	0.693	0.050	0.269	0.000	**0.699***
HPC	0.745	0.877	0.662	0.045	**0.944**
Proxifier	0.380	0.000	0.015	0.117	**0.979**
ThunderBird	0.868	0.114	0.781	0.021	**0.970**
Openstack	0.400	0.000	0.170	0.000	**0.801**
Spark	0.979	0.201	0.863	0.363	**0.995**
Zookeeper	0.962	0.722	0.918	0.046	**0.971**
Hadoop	0.546	0.125	0.293	0.000	**0.660***
BGL	0.810	0.457	0.702	0.004	**0.910**
AVG	0.737	0.352	0.517	0.103	**0.892**

Table 3.6: The Cliff's effect size test results

Algorithm name	Cliff's Delta
Drain	**0.39**
Logram	**0.74**
Spell	**0.76**
AEL	**1.00**

accurate method, Drain, has an average accuracy of 73.7%. Table 3.6 shows the effect size using Cliff's δ, which is a statistical test that indicates the magnitude of that difference [125]. The effect size is considered small when $0.147 \leq \delta < 0.33$, medium for $0.33 \leq \delta < 0.474$, and large for $\delta \geq 0.474$. [126]. Cliff's δ is defined using Equation(1). As shown in Table 3.6, the difference between the accuracy of ULP and that of Logram, SPELL, and AEL is significantly large (Cliff's $delat \geq 0.474$). It is medium in the case of Drain.

$$Cliff's\ \delta = \frac{\sum_i \sum_j sign(y_i - x_j)}{n_y.n_x} \tag{1}$$

We carefully examined the log templates that ULP missed to understand the causes. We found that some errors and inconsistencies in the manual labelling of the benchmark files misled the performance of ULP. For example, the token `CrazyIvan46` in the log event

`labeling NETClientConnection evaluate CrazyIvan46 CI46 Perform`

`CrazyIvan46` is considered as a static token, which is an error in the labelling of the data. This

Table 3.7: Examples of issues with the ground truth files for Apache and Hadoop log datasets

Log event	**Template from ground truth file**	**Explanation of the issue with the ground truth file**
Disk quotas dquot_6.5.1	Disk quotas dquot_<*>.<*>.<*>	Interpreting one dynamic token as many.
data_thread() got not answer from any [Thunderbird_A] datasource	data_thread() got not answer from any [Thunderbird_<*>] datasource	A dynamic variable is a whole, can not be split. The dynamic variable should replace the whole string Thunderbird_34.
Warning: we failed to resolve data source name dn910 dn911 dn912 dn913 dn914 dn915	Warning: we failed to resolve data source name <*>	Multiple dynamic variables are identified as one. This may be confusing since the practitioner loose the info about the number of dynamic tokens contained in the event.
workerEnv.init ok workers2.properties	workerEnv.init ok workers2.properties	The token workers2.properties does not change and does not have any other value, which make him be interpreted as static (occurrences 568).

should be labelled as a dynamic token because it refers to a username. Another cause of misclassification is the presence of dynamic variables whose values do not change over a large number of log events. For example, the token ”workers2.properties” was repeated 568 times in one of the datasets. ULP misclassified it as a static token. The good thing about ULP is that, unlike many existing tools including Drain, it does not make any assumptions about the order of static and dynamic tokens, which reduces significantly the number of false positives. Table 3.7 shows some other examples of errors in the ground truth files causing the low accuracy results of ULP when applied to Hadoop and Apache logs. This seems to affect the other parsers as well (see Table 3.5).

RQ1 Finding: The accuracy of ULP when applied to 10 log files of the LogPai benchmark is between 66% and 99.9% with a average of accuracy of 89.2%. ULP outperforms Drain, Logram, Spell, and AEL in the parsing of all the 10 log files.

Log Event:
`Receiving block blk_579248909 src: /10.251.30.6:335 dest: /10.251.30.6:500`
Template:
`Receiving block * src: * dest: *`

Figure 3.4: An example of a log event and its corresponding log template

Table 3.8: Number of log events of the sample log files used to measure efficiency

BGL	400	4,000	40,000	400,000	4,000,000
HPC	600	3,000	15,000	75,000	375,000
HDFS	1,000	5,000	10,000	100,000	1,000,000
Zookeeper	4,000	8,000	16,000	32,000	64,000
Spark	1,000	5,000	10,000	100,000	1,000,000

3.4.2 Efficiency

To evaluate ULP's efficiency, we record the execution time to complete the end-to-end parsing process. We repeated the experiment 10 times to avoid any bias and took the average of the execution times. We also run the same experiments for Drain, Logram, and SPELL on our computer and record the running parsing time for these programs in the same manner. We did not assess AEL's efficiency because it has a very poor accuracy (average accuracy of 10%, as indicated in Table 3.5)). We run the experiments using a MacBook Pro laptop running macOS Big Sur version 11.4 and equipped with a 6 Intel Core i7 CPU operating at 2.2GHz, 32GB 2400MHz DDR4 RAM, and a 256 GB SSD hard drive. We use the datasets indicated in Table 3.8, publicly accessible in the LogPai benchmark. We selected these datasets because they have previously been used to measure efficiency in other research studies [13] and [64]. We assess efficiency for each log dataset by running ULP and the other tools on random log samples of increasing size, as indicated by the number of log lines so as to examine how the parser's efficiency changes as the file become larger. This was by inspired by the way efficiency was assessed for Drain [64], which uses this log dataset splitting, except that Drain uses the file size rather than the number of log events.

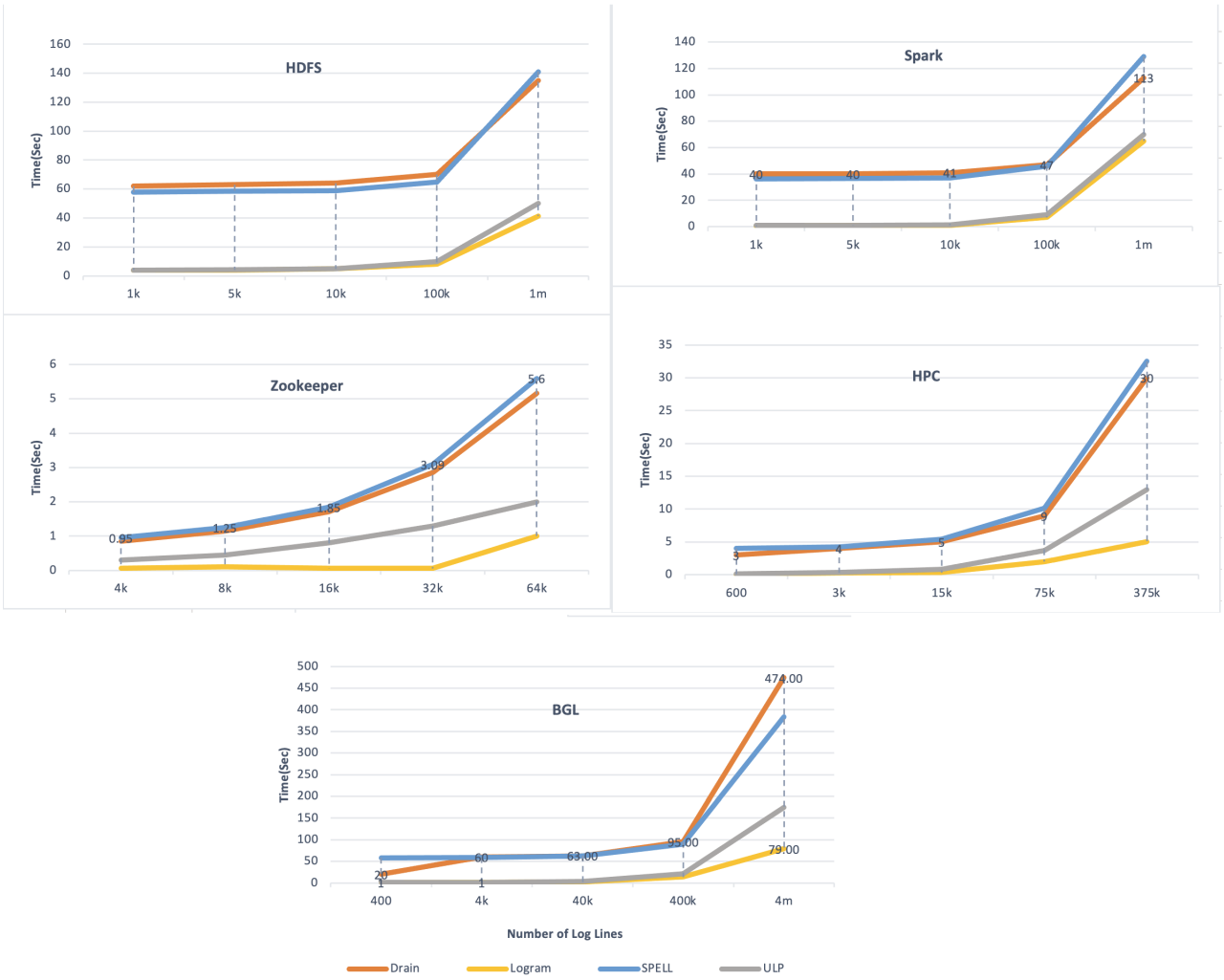

Figure 3.5: Results of ULP efficiency. The x-axis represents the number of log events and the y-axis represents the execution time in seconds

Results

Figure 3.5 shows the efficiency of ULP. ULP requires less than 50 seconds to parse one million log lines from the HDFS log file and less than three minutes to parse four million lines from the BGL log file. ULP's efficiency is comparable to that of Logram (the quickest log parsing tool evaluated in this paper). It is worth mentioning that Logram uses an upfront step to fine-tune the threshold for proper parsing. This step is not included in our analysis of Logram's efficiency because the corresponding code is not available in Logram's Github repository.

RQ2 Findings: ULP can parse up to 4 million log events in less than 3 minutes. It is more efficient than Drain and and Spell when applied to HDS, Spark, Zookepr, HPC, and BGL log files. It exhibits similar efficiency as Logram except for Zookeeper and for large HPC and BGL log files where Logram has a noticeably better efficiency.

3.5 Discussion

The primary difference between ULP and existing approaches lies in the way ULP is designed. ULP leverages the idea that static and dynamic tokens of log events can be better identified if we use frequency analysis locally on instances of log events that belong to the same group rather than in the entire log file. The similarity technique used to group log events is also unique. This design choice yields excellent results as shown in the previous subsections. It should be noted, however, that the sole reliance on local frequency analysis does not guarantee the detection of all dynamic tokens. If the same dynamic token is repeated as many times as static tokens, it will be misclassified by ULP. One way to address this is to improve the pre-processing step by targeting such variables.

Additionally, as opposed to other log parsers, ULP makes no assumptions about the position of a static or dynamic token. Drain, for example, assumes that a token that appears in the beginning of a log message is a static token, which is not always valid. Furthermore, ULP is able to detect dynamic tokens and log templates from a variety of unknown log files without using domain knowledge regular expressions during the pre-processing stage such it is the case for Drain [64] and Logram [13]. ULP leverages only generic regular expressions.

Another assumption made by Drain's authors is that log events with a similar length belong to the same group without necessarily checking the content of the events, which results in classifying very different log events into the same group. ULP overcomes this problem by applying a rigorous string matching technique to ensures that log events can only be grouped together if they share the same static tokens. In Table 3.9, we summarized some of the parsing errors in Drain caused by the assumptions in the design of the tool. As for Logram, one of its main limitations consists of the way it deals with log events that appear only once. For these events, the whole log template is considered by Logram to be composed of only dynamic variables. Another limitation of logram is related to the use of n-grams, which leads to the situation where n-gram sequences may be considered as

dynamic variables if they do not occur as frequently as other n-grams. For example, in the log event "Resolved 04DN8IQ.fareast.corp.microsoft.com to /default-rack", the 2-gram "Resolved 04DN8IQ.fareast.corp.microsoft.com" appears only twice in the log file as opposed to the 2-gram "to /default-rack" which appears more frequently, the template generated for this log event is "<*> <*> to /default-rack", which is not valid ("Revolved" should be detected as a static token).

In some ways, our approach is closer in principle to that of AEL. It is possible to categorize log events based on linguistic commonalities using the AEL approach. However, starting with simple dynamic patterns, AEL uses hard-coded algorithms based on system information (e.g., IP addresses, numbers, and memory locations) to identify more complex patterns. The generated log events are then tokenized and binned based on the number of words they contain. This categorization evaluates the log messages in each bin before abstracting them into templates for use elsewhere in the system. The difficulty with AEL is that it assumes that events with the same number of words belong to the same group, resulting in many false positives when analyzing log events. ULP makes use of string matching similarity, which combines static tokens and the number of tokens in a log event, to overcome this problem.

3.6 Threats to Validity

In this section, we discuss the threats to the validity of this study, which are organized as internal, external, conclusion, and reliability threats.

Internal validity: Internal validity risks are those factors that have the potential to influence our outcomes. It is possible that mistakes happened during the implementation and testing of ULP, despite our best efforts. In order to reduce this hazard, we tested the tool on a large number of log files and manually reviewed its results on a limited number of samples. In addition, we make the tool and data accessible on Zenodo so that researchers may run the tool and check the results. Finally, in order to determine the correctness of ULP, we had to look at the disparities between the findings provided by ULP and the results acquired by the ground truth. This was accomplished in part via scripts and in part through manual checks. All efforts were made to minimize the possibility

Table 3.9: Example of flaws introduced by Drain parsing assumptions

Log event	**Drain grouping**	**Ground truth grouping**	**ULP grouping**
attempt_123 TaskAttempt Transitioned from NEW to UNASSIGNED	<*><*>Transitioned from <*>to <*>	attempt_<*>TaskAttempt Transitioned from NEW to UNASSIGNED	<*>TaskAttempt Transitioned from NEW to UNASSIGNED
task_123 Task Transitioned from NEW to SCHEDULED	<*><*>Transitioned from <*>to <*>	task_<*>Task Transitioned from NEW to SCHEDULED	<*>Task Transitioned from NEW to SCHEDULED
task_123 Task Transitioned from SCHEDULED to RUNNING	<*><*>Transitioned from <*>to <*>	task_<*>Task Transitioned from SCHEDULED to RUNNING	<*>Task Transitioned from SCHEDULED to RUNNING
attempt_123TaskAttempt Transitioned from RUNNING to SUCCESS_CONTAINER_CLEANUP	<*><*>Transitioned from <*>to <*>	attempt_<*>TaskAttempt Transitioned from RUNNING to SUCCESS_CONTAINER_CLEANUP	<*>TaskAttempt Transitioned from RUNNING to SUCCESS_CONTAINER_CLEANUP
attempt_123 TaskAttempt Transitioned from RUNNING to FAIL_CONTAINER_CLEANUP	<*><*>Transitioned from <*>to <*>	attempt_<*>TaskAttempt Transitioned from RUNNING to FAIL_CONTAINER_CLEANUP	<*>TaskAttempt Transitioned from RUNNING to FAIL_CONTAINER_CLEANUP
kernel time sync disabled 12:56	kernel time sync <*><*>	kernel time sync disabled <*>	kernel time sync disabled <*>
kernel time sync enabled 09:45	kernel time sync <*><*>	kernel time sync enabled <*>	kernel time sync enabled <*>

of mistakes.

Reliability Validity:. The potential of reproducing this research is referred to as reliability validity. The study's evaluation, replication, and reproducibility are made easier by the use of an open-source program. ULP and all the data used in this paper are available online on Zenodo[5].

Conclusion validity. The accuracy of the collected findings corresponds to the validity risks associated with the conclusion. We used ULP to analyze ten log files that have been commonly used in comparable investigations in the past. The accuracy and efficiency experiments were thoroughly reviewed to verify that the findings were appropriately interpreted, and we made every effort to do so. The tool and data that were used in every phase of this study are made accessible online to enable the evaluation and replication of our findings.

External validity: The generalizability of the findings is what is meant by external validity. We tested our findings on a total of ten log files from a variety of different software systems. Ten log files from the LogPai benchmark were used to evaluate ULP's performance. As a result, we cannot say with certainty that ULP's accuracy would be the same if it were applied to other datasets. However, since these datasets span a wide range of software systems from a variety of areas, they serve as a useful testbed for log parsing and analysis techniques. We do not claim that our findings can be applied to all available log files, particularly industrial and proprietary logs, to which we did not have access when conducting this research. We are currently working with industrial partners to apply ULP to their logs.

3.7 Conclusion

We presented ULP, a powerful log parsing approach and tool. ULP differs from other tools in its design. It uses a novel way to distinguish between static and dynamic tokens of log events by applying string matching similarity to create groups of similar log events, and local frequency analysis to instances of the same group to distinguish between static and dynamic tokens. By doing so, ULP is capable with high accuracy to extract log templates that can be used to recognize and structure log events. Our approach confirms that is indeed possible to create an effective and efficient

[5]http://zenodo.org/record/6425919

universal log parsers, which eliminates the need to develop specific parsers for various types of log files. Moreover, ULP is readily usable by practitioners to support various maintenance tasks that rely on log analytics. ULP is more accurate in parsing a representative set of 10 log files of the LogPai project than four leading open source log parsers. ULP is very efficient too. It took 3 minutes to parse up to 4 million Log events. Future work should build on this work by focusing on the following aspects (a) apply ULP to more logs, especially those from industrial systems, (b) improve ULP by adding more regular expressions to identify other trivial dynamic variables such as domain-specific variables, and (c) conduct a time algorithm complexity analysis to determine with precision the best, worse, and expected running time of ULP, (d) improve the efficiency of the tool when applied to log files with a large number of templates, with high variability, and (e) adapt ULP to online parsing, which eliminates the need of a training set.

Chapter 4

AML: A Novel Accuracy Metric for Log Parsing Evaluation

"To measure is to know. If you cannot measure it, you cannot improve it."

— Adapted from Lord Kelvin

Bridging ULP to AML: Having established a high-performance approach to log parsing through ULP, the next critical step is to rigorously assess its effectiveness alongside other existing methods. Although ULP offers significant gains in both speed and scalability, meaningful comparisons require metrics that capture more than just computational efficiency. Traditional evaluation methods often overlook subtle but impactful mistakes, such as misclassified templates or missed tokens, that can dramatically affect downstream analytics in AIOps environments. Thus, while ULP addresses pressing performance needs, a dedicated, fine-grained measure of parsing accuracy remains essential to gauge its true contribution and guide further improvements. To fill this gap, the subsequent chapter introduces the Accuracy Metric for Log Parsing (AML), a framework capable of evaluating log parsers with the granularity and consistency necessary for modern, large-scale IT operations.

By quantifying both omission and commission errors, AML ensures that the benefits of ULP are not only theoretical but also verifiable in diverse and evolving real-world scenarios.

I. Sedki, A. Hamou-Lhadj, O. Ait-Mohamed, "AML: An accuracy metric model for effective evaluation of log parsing techniques", In Proc. of the Journal of Systems and Software (JSS), Volume 216, 2024, 112154, ISSN 0164-1212, https://doi.org/10.1016/j.jss.2024.112154.

Abstract: Logs are essential for the maintenance of large software systems. Software engineers often analyze logs for debugging, root cause analysis, and anomaly detection tasks. Logs, however, are partly structured, making the extraction of useful information from massive log files a challenging task. Recently, many log parsing techniques have been proposed to automatically extract log templates from unstructured log files. These parsers, however, are evaluated using different accuracy metrics. In this paper, we show that these metrics have several drawbacks, making it challenging to understand the strengths and limitations of existing parsers. To address this, we propose a novel accuracy metric, called AML (Accuracy Metric for Log Parsing). AML is a robust accuracy metric that is inspired by research in the field of remote sensing. It is based on measuring omission and commission errors. We use AML to assess the accuracy of 14 log parsing tools applied to the parsing of 16 log datasets. We also show how AML compares to existing accuracy metrics. Our findings demonstrate that AML is a promising accuracy metric for log parsing compared to alternative solutions, which enables a comprehensive evaluation of log parsing tools to help better decision-making in selecting and improving log parsing techniques.

4.1 Introduction

Logging is a common programming practice that developers use to reason about the runtime aspects of a software system. Logs are generated by inserting logging statements into the source code. Figure 4.1 shows an example of a logging statement taken from the Hadoop Distributed File System (HDFS). The figure also shows the generated log event after executing this statement. A log event typically has two sections, namely, the log header and the log message. The log header contains information about the execution context, including the timestamp (e.g., `071203 283349` in the log event of Figure 4.1), the process ID (e.g., `127`), the log level (e.g. `INFO`) and the logged component of the system (e.g., `dfs.DataNodeResponder`). The log message contains static text (e.g., `Received Block, of size, from`), and dynamic variables (e.g., `blk_-1680, 911, 10.234.11.201`). Because logs can sometimes be the only data available to analyze a software system's behaviour, they are essential for a variety of software engineering tasks such as anomaly detection and debugging [3, 4, 2**?**], understanding system faults [3, 43, 4], performance and quality analysis [127, 128, 43], operational intelligence [2**?** , 127, 129, 130], failure prediction[**?** 131], data leaks and security issues [51], and AI for IT Operations (AIOps) [1, 2]. Logs, however, are partially structured [3, 131], making the extraction of useful information from massive log files a challenging task [132, 133]. Log parsing techniques have been proposed to automatically extract log templates (i.e., log structures) from unstructured log files. For the log event example in Figure 4.1, the extracted log template is `Received Block * of Size * from *`, where the symbol `*` indicates the position of a dynamic parameter. Note that the log header structure is not included, since a log header usually follows the same format within a log file. Parsing a log header can be done with a simple regular expression. This is not the case for log messages. A typical log file can

Logging Statement: `LOG.info("Received Block"+ block_id + "of size" + block_size + "from" + ip)`
Log Event: `071203 283349 127 INFO dfs.DataNodeResponder: Received block blk_-1680 of size 911 from 10.234.11.201`
Log Template: `Received Block * of Size * from *`

Figure 4.1: An example of a logging statement, a log event, and the corresponding log template

have thousands of log templates [50, 118].

4.1.1 Problem Statement

Many log parsing techniques and tools have been proposed to automatically parse log events[1], such as LogSig [134], LogCluster [1], AEL [135], Drain [136], IpLom [137, 138], Lenma [139], LFA [140], LKE [28], LogMine [141], Molfi [142], Shiso [143], SLCT [144], Spell [21], Logram [145], and recently ULP [30]. These tools use a range of methods including heuristics, mining frequent patterns, natural language processing, clustering, or a mixture of many of these. However, these parsers have been evaluated using different accuracy metrics including Grouping Accuracy [146], Parsing Accuracy [145], and Template Accuracy [147]. These metrics have several drawbacks. For example, parsing accuracy is sensitive to the number of log events associated with log templates. A high parsing accuracy may not reflect the ability of a parser to detect every template of the log file. We discuss in more detail existing accuracy metrics and their limitations in the related work section. Without a reliable and consistent evaluation mechanism, it is difficult to understand the strengths and limitations of log parsing approaches, such as why these tools perform well on some log files but poorly on others. In addition, the absence of a reliable evaluation approach makes it hard to understand how well log parsing approaches perform relative to alternative solutions, thus hindering meaningful progress. Widespread adoption of log parsing tools will not be possible until convincing empirical evaluations are obtained using a sound accuracy measurement approach.

4.1.2 Motivation

In the domain of log parsing, consistent accuracy is not just about achieving high percentages; it is also about ensuring the reliability and robustness of the measurement methodologies. A significant concern arises when these accuracy measurements fail to articulate precisely how a log event is determined as correctly parsed. Often, such determinations are made manually, introducing potential biases and inconsistencies. This lack of standardized assessment criteria poses challenges when comparing different tools or methodologies. Even marginal inaccuracies in parsing, as slight as 4% of the total, can have profound implications on performance, potentially leading to repercussions

[1] In the remaining parts of the paper, we use the terms log messages and log events interchangeably.

magnified by an entire order of magnitude [79].

The difficulties in measuring the accuracy of log parsing techniques are underscored by the significant variations observed in evaluations of DRAIN, a widely used log parsing technique. The disparities in reported accuracy are not trivial, particularly as some assessments indicate an effectiveness gap exceeding 4%. For instance, when evaluating the parsing of the Proxifier system logs, there is a notable discrepancy in results. He et al. [136] and Dai et al. [145] cite commendable effectiveness reflected by 99% and 93% respectively, yet the same assessments from Zhu et al. [146] and Sedki et al. [30] paint a contrasting picture, indicating substantial room for improvement with accuracy scores of 53% and 38%, respectively. These variances highlight the complexity and challenges associated with obtaining consistent accuracy measurements in log parsing.

4.1.3 Contributions of the Paper

In this paper, we introduce AML (Accuracy Metric for Log Parsing), a reliable and yet simple accuracy metric for log parsing. AML can be used to assess the accuracy of a log parser at both the template and log file levels. Inspired by the concept of thematic accuracy in the area of remote sensing ([148], [149], [150]), AML is designed to measure omission and commission errors of a parser. At the template level, omission errors occur when a parser fails to detect log events associated with the template. Commission errors occur when the parser detects excessive log events. Omission and commission errors can be computed at the level of the entire log file by measuring the number of missing or excessive templates in the log file. The best parser would be the one that minimizes errors of commission and omission at both the template and log file levels. AML is agnostic to the distribution of log events across templates, a known problem with the parsing accuracy metric that is widely used in the literature. Moreover, AML and its components can be used to analyze the sources of parsing faults. This can help practitioners dig deeper into each template to understand the root causes of parsing errors, hence performing a critical evaluation of the parser's performance.

We use AML to measure the accuracy of 14 log parsers when applied to parse 16 log file datasets from the LogHub benchmark [151]. We also examine how AML compares to other accuracy metrics. Our results reveal that AML is a powerful and simple accuracy metric for log parsing that not only provides adequate insight into the performance of a log parser as opposed to alternative

accuracy metrics but can also guide root cause analysis of parsing errors with the ultimate objective of improving parsing tools.

4.1.4 Paper Organization

This paper is structured as follows: In Section 4.2, we provide an overview of existing log parsing accuracy metrics and discuss their limitations. Section 4.3 delves into the details of the novel AML, explaining its methodology and capabilities. In Section 4.4, we apply AML to assess the accuracy of 14 log parsing tools, presenting our methodology. Section 4.5 presents the results of our study and provides an in-depth analysis of the outcomes. Our insights into future research directions and potential improvements are discussed in Section 4.6. Finally, we summarize our findings and contributions in Section 4.10, concluding the paper.

4.2 Related work

Log parsing is a prerequisite for log analysis tasks. In recent years, log analysis has received considerable attention from researchers and practitioners. El-Masri et al. [33] conducted an extensive review of 17 log parsing tools. The authors classified these tools using a quality model that focuses on the following aspects: coverage, delimiter independence, efficiency, system knowledge independence, execution mode, parameter tuning effort required, and scalability. Zhu et al. [146] presented another comprehensive survey of log parsing tools. The authors compared the performance of 13 parsing tools using grouping accuracy. However, both surveys did not compare the various accuracy metrics used by the log parsers.

More recently, Khan et al. [147] evaluates and compares techniques for log message template identification in real-world logs. The study proposes three guidelines: using appropriate accuracy metrics, performing oracle template correction, and analyzing incorrect templates. The analysis of incorrectly identified templates provides insights on the limitations of individual techniques and offers potential directions for improvement by identifying the types of incorrectly identified templates such as over-generalized (OG), under-generalized (UG), and mixed (MX). Similar to [147], our study identifies critical gaps in the existing log parsing accuracy metrics. However, we extend these

observations by offering a metric that not only identifies inaccuracies but also provides a methodological basis for addressing them, which was a limitation in the previous studies. The remaining part of the related work section is a review of existing log parsing accuracy metrics, namely Grouping Accuracy, Parsing Accuracy, Edit Distance, and Template Accuracy. To illustrate how these metrics work, we use the fictive ground truth log file shown in Table 4.1. This file consists of ten log events (E1, E2, ..., E10), which are parsed into three event templates (T1, T2, and T3). The ideal parser would be one that recognizes these templates and only these templates, as well as the log events and only the log events associated with each template.

4.2.1 Grouping Accuracy

Grouping accuracy (GA) is used mainly by tools such as Drain [136] and AEL [135] that treat the log parsing problem as a clustering problem. GA evaluates the accuracy of log template identification, conceptualized as a clustering process where log messages pertaining to different events are grouped into distinct templates. GA assesses whether the log messages grouped under a common identified template by the parsing tool match the grouping defined in the ground truth. Specifically, GA is quantified as the proportion of log messages that are ”correctly parsed” relative to the total number of log messages in the dataset. A log message is deemed ”correctly parsed” under the GA metric if it is grouped with other log messages in a manner that is consistent with the ground truth clustering. In practice, GA provides an indication of how well a log parsing tool can identify and

Table 4.1: Legend and Running example/Ground truth

Event	Ground-truth
E1	T3
E2	T2
E3	T3
E4	T1
E5	T2
E6	T3
E7	T1
E8	T2
E9	T1
E10	T2

Table 4.2: Scenario used to calculate GA

Event	Parsing Result
E1	T3
E2	T4
E3	T3
E4	T5
E5	T4
E6	T5
E7	T4
E8	T3
E9	T3
E10	T3

Table 4.3: Scenario used to calculate PA

Event	Parsing Result
E1	T5
E2	T2
E3	T1
E4	T1
E5	T3
E6	T3
E7	T1
E8	T2
E9	T1
E10	T2

Table 4.4: Scenario used to calculate PTA/RTA

Event	Parsing Result
E1	T3
E2	T4
E3	T3
E4	T1
E5	T2
E6	T3
E7	T1
E8	T5
E9	T1
E10	T6

group log messages into correct templates as defined by the ground truth, without necessarily considering the exact textual match or structure of the templates themselves. It focuses on the effective grouping of log messages into coherent clusters or templates, reflecting the tool's ability to accurately segregate logs based on their underlying events.GA focus on the number of clusters a parser can recover that are fully identical to the templates in the ground truth. GA is measured as shown in Equation (2). The metric consider only those groups that exactly match their corresponding ground truth groups in terms of content. Specifically, the formula ensures that each log message in an identified group is counted only if it forms a part of a group that exactly corresponds to a ground truth group:

$$GA = \frac{\sum_{i=1}^{n} \text{correctly\_grouped\_messages}_i}{\text{total\_number\_log\_messages}} \tag{2}$$

Where $\text{correctly\_grouped\_messages}_i$ represents the number of log messages in the i-th group that are grouped exactly as per the corresponding ground truth template.

For example, assume that parsing the fictive log file used as ground truth using a given log parser results in three clusters T4, T5, and T3, which contain the events shown in Table 4.2. From this table, we can see that the only template that was recovered is T3 by recognizing events E1 and E3. In this case, $GA = 1 * 5/10 = 50\%$. where 1 is the number of correctly identified log templates, 5 is the number of log events in cluster T3, and 10 is the total number of log events. Note that when using GA, it does not matter if E8, E9, and E10 were mistakenly detected as part of T3. The metric considers a template to be recovered as soon as it recognizes at least one of its log events. This metric has many drawbacks. First, it does not account for log events that are properly parsed but it just happens that the approach was not able to group these events in the right cluster, i.e., with other similar events (like E6, which should be with the group of log events associated with template T3). Another issue is related to the fact that this metric evaluates the parser without checking if the logs are associated with the correct log template (for example, E8, which is supposed to be associated with T2).

4.2.2 Parsing Accuracy

Several parsers such as Spell [21], Logpunk [152] define the accuracy of a parser as the ratio of the total number of log events that match the string representation of the template in the ground truth word by word (K) to the total number of events in the ground truth (N). This is known as Parsing Accuracy (PA) and is calculated as shown in Equation (3).

$$PA = \frac{K}{N} \tag{3}$$

The outcome of parsing a log event is considered correct if and only if it corresponds to the same template of log events as the ground truth. For example, assume that the result of parsing the log events shown in the fictive log file is the one shown in Table 4.3. We obtain $PA = 7/10 = 70\%$, because three log events, namely E1, E3, and E5, were not parsed correctly. PA is simple to compute but tends to be sensitive to the number of events in the log, which can be misleading when there are many repetitive events. hence, PA can be very highly sensitive to the distribution of log events across templates. In addition, PA does not show if the log parsing method generates more templates than the ones in the ground truth. Parsing Accuracy (PA) treats incorrectly adding extra templates or log entries and failing to recognize required templates or log entries similarly. This approach can obscure the specific weaknesses of a parser by not adequately distinguishing between fundamentally different types of errors. Consider a scenario where the ground truth for log messages specifies grouping into two templates:

- **Template T1:** Contains messages A, B, and C.
- **Template T2:** Contains messages D and E.

Assume two different outcomes from two parsers:

(1) **Parser 1 Outcome:**

- Correctly identifies T1 with messages A, B, C.
- Fails to identify T2, omitting messages D and E entirely (omission), and parse them as T1.

(2) **Parser 2 Outcome:**

- Correctly identifies T1 with messages A, B, C.
- Incorrectly groups messages D and E into a new, non-existent Template T3 instead of T2 (commission).

In this example, both parsers fail to handle all the log messages as per the ground truth:Parser 1 omits messages D and E entirely, failing to include them in any group. Parser 2 misplaces messages D and E into an incorrect template, creating an unnecessary grouping. PA would compute the accuracy for both scenarios by considering the proportion of correctly parsed messages to the total messages:

$$PA = \frac{\text{Number of Correctly Parsed Messages}}{\text{Total Number of Messages}} \tag{4}$$

Despite different types of errors, PA yields the same score for both scenarios. PA does not differentiate between the nature of the errors—omission versus commission. Both types of errors reduce the PA score, but the metric does not indicate whether the error was due to missing data or due to additional, unnecessary data. This equal treatment of fundamentally different errors could obscure important distinctions in a parser's performance. For systems where missing data is more critical than additional data , or vice versa, PA might not provide enough information to accurately assess the log parser's suitability or reliability.

4.2.3 Edit Distance

The edit distance metric assesses the alignment capability of a parsing method with respect to matching log templates to their respective log messages in a dataset [153]. The primary goal of this metric is to minimize the difference between the parsed and the ground truth log templates. The edit distance is the minimum number of operations (insertions, deletions, or substitutions) required to transform one string into another within a log event. Given two log lines $\log_1$ and $\log_2$ of lengths

m and n respectively, the edit distance d between them is defined as:

$$d(\log_1, \log_2) = \begin{cases} 1 + \min\{d(\log_1[1..m], \log_2[1..n-1]), \\ \qquad d(\log_1[1..m-1], \log_2[1..n]), \\ \qquad d(\log_1[1..m-1], \log_2[1..n-1])\} \end{cases}$$

$$d(\log_1, \log_2) = \begin{cases} m & \text{if } n = 0, \\ n & \text{if } m = 0, \\ d(\log_1[1..m-1], \log_2[1..n-1]), & \text{if} \\ \log_1[m] = \log_2[n] \end{cases}$$

For log1: "ERR: File not" and log2: "ERR: File found", the computation can be broken down as:

$$d(\text{"ERR: File not"}, \text{"ERR: File found"}) =$$
$$1 + d(\text{"ERR: File no"}, \text{"ERR: File foun"})$$
$$d(\text{"ERR: File no"}, \text{"ERR: File foun"}) =$$
$$1 + d(\text{"ERR: File n"}, \text{"ERR: File fou"})$$
$$d(\text{"ERR: File n"}, \text{"ERR: File fou"}) =$$
$$1 + d(\text{"ERR: File "}, \text{"ERR: File fo"})$$
$$d(\text{"ERR: File "}, \text{"ERR: File fo"}) =$$
$$2 + d(\text{"ERR: File "}, \text{"ERR: File "})$$

Further computations would result in a distance of 0 since the rest of the strings are identical. Thus, the edit distance between "ERR: File not" and "ERR: File found" is 3. This metric can be invaluable in template-based log matching, providing a flexible measure that can accommodate slight deviations from templates due to variable parts in log messages. However, the computational efficiency of employing the edit distance in this context is a critical consideration. For a log message

of length m and a template of length l, the traditional dynamic programming approach to compute the edit distance requires $O(m \times l)$ time.

Given the extensive volume of log messages generated in real-world systems and the potential diversity of log templates, this computational cost can become significant. For example, if a system produces thousands of log entries per second and uses hundreds of templates, real-time or near-real-time analysis could be challenging.

The authors haven't clearly specified a strategy to upscale this metric from individual log-template matches to a comprehensive file-level evaluation. A straightforward proposition could be averaging the edit distances across all log-template pairs in a file. However, this simplistic aggregation might not capture the nuanced variations and dependencies within logs, especially if there are a multitude of templates involved and the distribution is not balanced. Moreover, such an averaging approach would compound the computational cost even more.

4.2.4 Template Accuracy

Recently, Khan et al. [147] proposed a new metric, called Template Accuracy. In this novel metric, a template is correctly identified from log events if and only if it is token-for-token identical to the one found in the ground truth. The identified template must be the same as the template associated with the messages for which it was identified. The authors introduced two Template Accuracy metrics: Precision-TA (PTA) and Recall-TA (RTA), which are based on standard information retrieval metrics precision and recall. PTA is defined as the ratio of correctly identified templates (O) to the total number of identified templates (L), which indicates the precision of the parsing technique at the template level and is calculated as follows:

$$PTA = \frac{O}{L} \tag{5}$$

Recall of the parsing approach at the template level is indicated by RTA, which is defined as the ratio of correctly identified templates (O) over the total number of templates in the ground truth (N).

$$RTA = \frac{O}{N} \tag{6}$$

Consider the result of the parsing shown in Table 4.4. The number of correctly identified templates is calculated by looking at the correct associations made by the parser, here T1 and T3, because the log messages were parsed correctly and associated with the right log template. This means that the number of correctly identified templates (O) is 2. The total number of identified templates (L) is 6 (T1, T3, T4, T5, and T6), and the total number of templates in the ground truth (N) is 3 (T1, T2, and T3).

$$PTA = 2/6 = 33\%. \tag{7}$$

$$RTA = 2/3 = 66\% \tag{8}$$

The template accuracy metric does not consider the number of events correctly associated with each template. In fact, this number is not included in the calculation of the template's accuracy to avoid introducing any biases. This metric provides little information about why the accuracy is low or which log events were wrongly parsed.

4.2.5 Discussion

As we showed in this section, existing log parsing accuracy metrics have many shortcomings that affect the evaluation results of log parsing tools. GA does not consider whether log events are correctly parsed or not. It only looks at the overall number of templates that were identified. PA is sensitive to the distribution of log events across templates, which can be inflated due to repetitions of the same log events. This metric is also biased by the presence of predominant templates (templates with a considerably large number of events).

The focus of the edit distance is limited to the pairing of log templates, overlooking broader patterns within the entire log files. This narrow scope creates ambiguity in aggregating results for evaluating the accuracy of parsing an entire log file. Furthermore, the computational cost, especially for massive log outputs and varied templates, can be substantial. For these reasons, we decided to exclude the edit distance when comparing AML to existing log parsing accuracy metrics (see Section 4.5.4).

PTA and RTA take a different approach by focusing on the number of templates that are correctly identified. They do not provide sufficient insight as to the number of log events that are correctly

parsed. These metrics are also not easy to interpret. One has to decide whether to favor precision or recall to assess the accuracy of a parser. In addition, none of these metrics can help with root cause analysis of potential parsing errors.

AML addresses these shortcomings by providing a more comprehensive approach for evaluating log parsing accuracy. Unlike PA, GA, RTA, and PTA, AML measures the number of errors at both the template and file levels. AML is not sensitive to the distribution of log events across templates either. Moreover, AML can be used by developers to analyze the sources of parsing errors.

4.3 AML: The Proposed Accuracy Metric for Log Parsing

In this section, we present AML to assess the accuracy of a log parser. AML is inspired by the concept of thematic accuracy for remote sensing applications [148]. Remote sensing is a specialized field of study that uses satellites and cameras to produce maps and images of the Earth, enabling us to understand various phenomena such as environmental changes, and patterns of land areas, among others [148]. Remote sensing techniques resort mainly to classification algorithms to detect areas of land cover and represent them as maps. Assessing the accuracy of the produced maps is not a straightforward task and has been the topic of many studies in the remote sensing literature [148]. A typical map contains different categories of data. For example, a climatic map can show the temperature, cloud cover, rain precipitation, relative humidity, etc. A robust accuracy metric should measure not only the overall quality of the map but also the individual quality of each category it comprises. This multi-level classification problem is difficult to assess using traditional classification measures such as the F1-score, which focuses on one level of classification. Aggregating multiple category-level F1 scores through micro-averaging or similar data analytic methods in order to assess the overall quality of the entire map may result in a metric that is complex to interpret because of the different types of data used in each category. The introduction of the AML metric is prompted by the complexities inherent in evaluating multi-level classification systems, such as log parsing, which are not adequately addressed by traditional metrics like F1-score, precision, and recall. Traditional metrics typically assess performance at a singular classification level and may not effectively capture the nuances of hierarchical or multi-category systems. In log parsing, we

often deal with multiple categories and sub-categories of logs, each with potentially differing levels of importance and distinct characteristics. Traditional metrics such as F1-score, which combine precision and recall in a harmonic mean, focus primarily on a single level of classification accuracy. They do not account for the impact of correct or incorrect classifications across different log categories in a way that reflects their overall importance or contribution to system performance. AML provides a nuanced view by integrating performance across multiple levels of log classification. A key distinction of AML lies in its treatment of excessive templates. Traditional metrics like precision and recall do not penalize the presence of excessive, unnecessary templates generated by a log parser. AML, however, incorporates a penalty for such inaccuracies, which is critical for assessing the overall quality and usability of parsed logs. Excessive templates can lead to inefficiencies and increased computational costs in downstream processing, making their consideration essential in the metric. AML is designed to handle the variability and complexity of data seen in different categories of logs. It adjusts its evaluation based on the nature of the data in each category, something that micro-averaged F1 scores might oversimplify, leading to a metric that can be difficult to interpret in a multi-faceted analysis.

In addition, because maps are approximate representations of the real world, they are expected to contain errors. Map producers and users are more interested in knowing how accurate a map is by measuring the errors it contains [150]. To achieve this, many thematic accuracy metrics rely on the concepts of omission and commission errors. An omission error occurs when a map omits an element of a category. A commission error happens when the map contains more data than the actual category. Omission and commission errors can be computed at the level of specific categories or at the level of the entire map. Errors or omissions and commissions at the map level occur when the algorithm either excludes certain categories or detects an excessive number of categories. The best remote sensing algorithm is the one that minimizes omission and commission errors at all levels of assessment.

By analogy, we can think of a log file as a map and event templates as categories of the map. We can then compute the number of omissions and commissions at both the template and log file levels. AML uses a simple mechanism to aggregate errors of omission and commission at both levels into a single and intuitive metric. To explain how AML works, let us consider the example in Table

Table 4.5: Ground truth and parsing outcome

Template	Ground Truth Log Events	Parser's Outcome
T1	E1, E2, E3	E1, E2, E3
T2	E4, E5, E6, E7, E8	E4, E5, E6
T3	E9, E10	E7, E8, E9
T4	E11, E12	-
T5 (excessive template)	-	E10, E11, E12

4.5. In this table, we show an example of a ground truth log file where each log event E1 to E12 is associated with a log template T1 to T4. We also show the results of a fictive parser when parsing the log file. In this example, the parser made the following errors:

- Omission errors at the template level: The parser did not detect E7 and E8 in Template T_2, and did not detect event E10 in T_3.
- Commission errors at the template level: The parser wrongly assigned E7 and E8 to T3, which are two excessive events.
- Omission errors at the file level: The parser omitted to detect the entire template T_4.
- Commission errors at the file level: The parser detected an excessive template T_5, which did not match any template in the ground truth.

Although both T1 and T5 templates show an ErrO of 0, their contexts differ significantly. T1 is an ideal scenario where the parser perfectly matches the ground truth. T5, termed as an 'excessive template', shows no ground truth events, and thus, by definition, there cannot be any omission; however, it falsely introduces events, hence the ErrC of 1. These examples highlight that ErrO alone does not capture the presence of invalid log events. Although T1 and T4 have both ErrC at 0, T1 demonstrates a scenario with perfect parsing (no errors), while T4 shows a complete failure to parse any events despite their presence in the ground truth. T4's critical issue is captured by ErrO (1), which indicates a total omission. This observation points out that neither ErrO nor ErrC individually provides complete error information and underscores the necessity of using these metrics together, complemented by additional measures such as the ICSI shown in the table, to provide a more nuanced understanding of parser performance.

Table 4.6: Computing errors of omission and commission for the example in Table

Template	$ErrO$	$ErrC$	$ICSI$
T1	0	0	1
T2	0.4	0	0.6
T3	0.5	0.66	-0.16
T4	1	0	0
T5	0	1	0

4.3.1 Measuring Errors of Omission and Commission

Errors of omission, $ErrO_{Ti}$, for a template T_i is the ratio of the number of events of T_i that the parser failed to detect as part of T_i to the total number of events in T_i. Equation 9 is used to calculate $ErrO_{Ti}$ where:

- False Negatives (FN) represent events from T_i that were detected by the parser as belonging to other templates.
- True Positives (TP) represent events from T_i that were correctly detected by the parser.

$$ErrO_{Ti} = \begin{cases} \frac{FN}{TP+FN} & TP + FN \neq 0 \\ \\ 0 & TP + FN = 0 \end{cases} \tag{9}$$

Errors of omissions are calculated for all the templates that are in the ground truth and for any excessive template that the parser has mistakenly identified (e.g., T_5 in the example of Table 4.5). ErrO for an excessive template equals zero. This is the special case where (TP+FN = 0).

$ErrO_{Ti}$ varies from 0 to 1. A value that is equal to 0 means that the parser detected all the events of the template T_i based on the ground truth or that the template T_i is an excessive template. A value of $ErrO_{Ti}$ equal to 1 means that the parser did not detect any events of the template T_i. Table 4.6 shows the results of omission errors for templates T_1 to T_5. $ErrO_{T1} = 0/3 = 0$ because the parser detected all events of T_1 (i.e., no errors of omission). $ErrO_{T2} = 2/5 = 0.4$ because the parser failed to detect E7 and E8. etc. For the special case of Template T_5, $ErrO_{T5} = 0$ because it is an excessive template.

The error of commission is the ratio of the number of events that were identified mistakenly as belonging to the template T_i (i.e., false positives) to the total number of events that were identified

as belonging to template T_i (the sum of true positives and false positives). A commission error ratio $ErrC_{Ti}$ for template T_i is calculated using Equation 10. Following the same logic as for errors of commission, $ErrC_{Ti} = 0$ means that the parser did not detect any excessive log events. $ErrC_{Ti}$ that is strictly less than 1 means that the parser has detected excessive log events. $ErrC_{Ti} = 1$ indicates an excessive template altogether. For the special case where a ground truth template was not detected by the parser (e.g., T_4) we assign $ErrC_{Ti} = 0$.

$$ErrC_{Ti} = \begin{cases} \frac{FP}{TP+FP} & TP + FP \neq 0 \\ \\ 0 & TP + FP = 0 \end{cases} \tag{10}$$

For the example in Table 4.5, $ErrC_{T1} = 0/3 = 0$ because the parser did not detect any excessive events. The same applies to T_2. $ErrC_{T3} = 2/3 = 0.66$ because out of the three events detected by the parser as associated to T_3, two of these (i.e., E7 and E8) do not belong to T_3. $ErrC_{T3} = 0$ because T_3 was not at all detected. Finally, all events of T_5 are considered commission errors since the entire T_5 template is an excessive template.

4.3.2 Combining Errors of Omission and Commission

To assess the accuracy of a parser in identifying each template T_i we need to combine commission and omission errors of T_i into one equation. One way to achieve this would be to use the harmonic mean,[2] similar to the way the popular F1-score is computed. The problem with the harmonic mean is that it is sensitive to extreme values of any of the ratios that are involved in the calculation. Consider for example a parser that results in ErrO = 50% and ErrC = 50% for a given template, and another parser that results in ErrO = 90% and ErrC = 10% for the same template. The harmonic mean will disproportionately penalize the second parser (H=0.18) when compared to the first one (H=0.5). Another issue with the harmonic mean is that it only applies when all the ratios are different from zero, which is not the case for ErrC and ErrO. Both ratios can be equal to zero to account for situations where the parser omits to detect some templates or detects excessive templates in the log file as discussed in the previous section.

[2]The harmonic mean H of n ratios x_1 to x_n is computed as $H = n/(1/x_1 + 1/x_2 + ...1/x_n)$

Our approach for combining errors of omission and commission is based on the Individual Classification Success Index (ICSI) that was proposed by Koukoulas et al. [149] in the field of remote sensing. ICSI is a composite index that combines linearly ErrO and ErrC ratios. Composite indices are used in other fields [154] to combine multiple ratios in order to assess the overall quality of the observed phenomenon. Examples include the heat index that combines temperature and relative humidity, stock exchange indices for investment forecasting, and so on.

ICSI of Template T_i is calculated using Equation 11. AML aggregates the ICSIs of all templates to measure the overall performance of a parser as we will show in the next subsection.

$$ICSI_{Ti} = 1 - (ErrO_{Ti} + ErrC_{Ti}) \tag{11}$$

$ICSI_{Ti}$ varies from -1 to 1. A value of $ICSI_{Ti}$ equal to 1 indicates best accuracy in the sense that the parser was able to detect the events and only the events of T_i. A value of $ICSI_{Ti}$ that converges to -1 indicates poor performance of the parser for template T_i and reflects the situation where both omission and commission error ratios converge to 1. When applied to the above example, both parsers will result in the same ICSI value (ICSI = 1 - (0.5 + 0.5) = 1 - (0.9 + 0.1) = 0).

It should be noted that, in this paper, we assign the same weight to ErrO and ErrC when computing ICSI. In practice, a developer may opt to assign varying weights based on the significance assigned to each type of error. This can help developers select parsers depending on the type of errors they make. For example, one may decide to choose parsers that make fewer omission errors than commission errors or vice versa. Further studies should be conducted to investigate the need for a weighted ICSI.

4.3.3 Calculating AML

The AML metric is calculated using Equation 12, where `N` represents the total number of templates in the ground truth, and `D` represents the number of detected templates. `max(N, D)` is used to account for the situation where `D` is superior to `N`, meaning that the parser detected more templates than needed (i.e., a commission error at the log file level).

$$AML = \frac{\sum_{1}^{\max(N,D)} ICSI_i}{\max(N, D)} \tag{12}$$

The AML score ranges from -1 to 1. A value of 1 indicates that the parser was able to parse all the log events and assign them to the appropriate templates, as well as detect the templates and only the expected templates. When the AML score is negative, it means that the parser performed poorly, and the negative value indicates the extent of the poor performance. For example, an AML score of -0.5 means that the parser has detected some expected templates and log events, but it has also missed some templates and/or detected some incorrect log templates and the extent of these errors is relatively significant. Note that if the ICSI values are mixed (some positive, some negative) such that their sum equals zero, then the overall AML value will be zero. This can happen when the parser has both omission and commission errors across multiple templates, and the errors cancel each other out in terms of their impact on ICSI. Essentially, this means that the parser has correctly classified some templates while making errors in others, resulting in a net neutral impact on the overall AML score.

$$AML = \frac{1 + 0.6 - 0.16 + 0 + 0}{5} = 0.35 \tag{13}$$

One strength of AML is that not only it considers errors of omission and commission at the individual template level, but also takes into account the overall accuracy of the parser in detecting the correct number of templates (as we saw in the example with the log template T_5). This means that AML can identify situations where the parser may detect too many templates (commission errors at the file level) or too few templates (omission errors at the file level), and it penalizes such errors by reducing the overall AML score.

4.4 Study Setup

4.4.1 Log Parsing Tools

There are many log parsing tools that have been proposed in the last decade. Two comprehensive surveys of these tools are provided by Zhu et al.[146] and EL-Masri et al. [33]. In this paper, we

Table 4.7: The log parser used in this study

Parser	Key characteristic	Reference
Lenma	Computes similarity to templates of existing clusters.	[139]
Shiso	Measure resemblance of new templates to clusters.	[143]
LKE	Distance-based clustering technique	[28]
LogSig	Identifies log events using a set of signatures	[134]
Molfi	Parsing as a multiple objective optimization problem	[142]
ULP	Pattern recognition technique based on text similarity	[30]
SLCT	Mining line patterns and outlier events from textual event logs	[144]
Logmine	Uses MapReduce to abstract heterogeneous logs	[141]
LogCluster	Extracts terms from the logs into tuples	[1]
Spell	Relies on the longest common sequence	[21]
Drain	Abstracts logs into event using a tree	[136]
AEL	Relies on textual similarity to group logs together	[135]
IpLom	Employs a heuristic-based hierarchical clustering	[137, 138]
Logram	Leverages n-gram dictionaries	[145]

evaluate the accuracy of 14 log parsing tools, which include the 12 best-performing tools that were surveyed by Zhu et al.[146] and EL-Masri et al. [33]. We add to this list Logram [145] and ULP [30], which were recently released. We used the same parameter settings described by Zhu et al. [146] and the authors of Logram [145] and ULP [30]. Note that many of these tools did not compile due to bugs and the use of outdated libraries. We had to fix and update many of them. Table 4.7 lists the tools used in this study and their main characteristics, with reference to the key publications that describe the tool. It should be noted that this list of parsers is not exhaustive and that we may have missed to include some parsers. We do not see this as a significant threat to validity because we believe that the selected parsers are representative of the state of the art.

4.4.2 Datasets

We evaluate the selected log parsing tools using 16 log datasets from the LogHub benchmark [151], which is available online[3]. The datasets consist of a collection of log files, generated from various systems, including Apache, HPC, and HDFS as shown in Table 4.8. They are used extensively in the literature to compare the performance of log parsers.

Each log dataset from the LogHub benchmark used in this study comes with a subset of 2,000

[3]https://zenodo.org/record/3227177#.YUqmXtNPFRE

log events that have been parsed manually. The log templates were identified and each log event out of the 2,000 events was associated with a specific log template. This ground truth dataset is meant for researchers to test their Log parsers and has been used by many studies (e.g., Drain [136], Logram [145], ULP [30], Khan et al.[147]). Table 4.9 shows examples of templates that the logHub creators have manually extracted from a subset of the 2,000 log events of the Apache system.

Table 4.8: Log datasets from LogHub benchmark used in this study

System	Description	Number of Templates
Distributed systems		
HDFS	Hadoop distributed file system log	14
Hadoop	Hadoop mapreduce job log	114
Spark	Spark job log	36
Zookeeper	ZooKeeper service log	50
OpenStack	OpenStack infrastructure log	43
Supercomputers		
BGL	Blue Gene/L supercomputer log	120
HPC	High performance cluster log	46
Thunderbird	Thunderbird supercomputer log	149
Operating systems		
Windows	Windows event log	50
Linux	Linux system log	118
Mac	Mac OS log	341
Mobile systems		
Android	Android framework log	166
HealthApp	Health app log	75
Server applications		
Apache	Apache web server error log	6
OpenSSH	OpenSSH server log	27
Standalone software logs		
Proxifier software log	Software logging tool	8

When measuring the accuracy of various log parsing techniques, we noticed some recurring parsing errors, regardless of the tool that was used. An in-depth analysis of the log datasets revealed that the labeling of the ground truth contained minor errors. For example, in the HDFS log file, the block id variable is divided into two sections. Thus, the log event `Received block blk_11234 of size 910 from /10.250.14.224` is assigned to the log template `Received Block blk_<*> of size <*> from <*>` instead of `Received Block <*> of size <*> from <*>`. This is because `blk_11234` is a dynamic variable. Therefore, we remove the

Table 4.9: An example of templates from the Apache log file ground truth

ID	Template
1	jk2_init() Found child * in scoreboard slot *
2	workerEnv.init() ok *
3	mod_jk child workerEnv in error state *
4	[client *] Directory index forbidden by rule: *
5	jk2_init() Can't find child * in scoreboard
6	mod_jk child init * *

blk_ string from the ground truth file. We made similar corrections to other files when the errors were straightforward. Khan et al. in [147] noticed the same problem. They proposed an automated approach based on a set of heuristics that uses regular expressions to correct the datasets. However, their approach is heavily based on log parsing to automatically fix the dataset, which can lead to errors and introduce serious internal threats to validity.

4.4.3 Research questions

We evaluate the accuracy of 14 log parsing tools using AML. The objective is to answer five research questions:

- **RQ1:** How do existing log parsing tools perform using AML?

The answer to RQ1 provides insights into the performance of log parsing tools using AML. We also examine the omission and commission errors made by these parsers.

- **RQ2:** How do dynamic variables and log message density impact the performance of log parsing tools?

RQ2 investigates the impact of dynamic variables and log message density on log parsing tool performance. Understanding these factors can help improve log parsing in diverse environments.

- **RQ3:** How does the performance of log parsing tools vary across different log datasets, and which datasets pose unique challenges?

RQ3 explores the variation in log parsing tool performance across different datasets. Identifying datasets with unique challenges can guide tool selection for specific applications.

- **RQ4:** How do these tools perform using AML compared to other accuracy metrics?

Table 4.10: Results of parsing the 16 datasets of the LogHub benchmark in terms of average AML, omission, and commission, and the total number of templates generated.

Log parser	Omission(%)	Commission(%)	AML score(%)	Number of Templates
ULP	26.40	38.09	35.72	1591
Drain	36.21	35.52	26.12	1920
SHISO	35.64	39.18	25.10	1483
AEL	34.41	41.14	24.51	1593
Iplom	38.37	36.66	23.52	1212
Spell	29.54	49.86	22.46	2238
Lenma	25.96	54.05	19.97	3171
LKE	46.65	36.50	16.85	2561
Logmine	29.60	53.67	16.76	3375
SLCT	55.61	28.80	15.57	777
Logsig	51.96	34.85	14.52	916
Molfi	35.72	54.59	9.70	3083
Logram	47.86	42.36	8.57	1889
Logcluster	19.26	74.99	5.68	4553

RQ4 compares log parsing tool performance using AML against other accuracy metrics. This analysis highlights the advantages of AML in providing a more reliable view of a parser's effectiveness.

- **RQ5:** How can we use AML to analyze the sources of parsing errors?

RQ5 focuses on utilizing AML to identify the sources of parsing errors. By pinpointing these sources, we gain valuable insights into the issues behind parsing errors, which may be challenging to determine using traditional metrics.

4.5 Results

4.5.1 RQ1.How do existing log parsing tools perform using AML?

Table 4.10 shows the average AML accuracy of the 14 log parsing tools used in this study when applied to the 16 datasets of the LogHub benchmark. Note that all the results of this study are available online.[4]

We observe that ULP consistently performs better than the other log parsers achieving the highest average AML score of 35.72% across all log parsers, followed by Drain (26.12%), SHISO

[4]The detailed results of this study are available on https://zenodo.org/record/7872794#.ZEsHxezMJhE

(25.10%) and AEL (24.51%). These tools are better than all tools in identifying the right templates and reducing the number of log events that are assigned to the wrong templates. Logcluster, Molfi and Logram achieve the lowest average AML (less than 10%).

Table 4.10 also shows the average error ratios of omission and commission, as well as the total number of templates for all dataset generated by each tool. Note that the expected total number of templates for all datasets is 1,363. Logcluster with an average omission error ratio equal to 19.26%, Lenma (25.96%), ULP (26.40%) and Spell (29.54%) have the lowest omission error ratios, while LKE (46.65%), Logram (47.86%), Logsig (51.96%), and SLCT (55.61%) have the highest omission error ratios (see also Figure 4.2 for a ranking of log parsers based on their omission error ratio). Except for Logram, these tools (i.e. SLCT, LogSig, and LKE) rely on clustering techniques, which may explain the high omission error ratio. Clustering-based approaches aim to group log events that are similar into clusters that do not necessarily match the templates in the ground truth. As for Logram, one of the main limitations that results in a high omission error ratio consists of the way the tool deals with log events that appear only once. For these events, the whole template is considered by Logram to be composed of only dynamic variables. Another major issue with the use of n-grams in Logram is that an n-gram sequence may be considered a dynamic variable and hence removed from the final pattern.

Take, for example, the following log event: `Resolved 04DN8IQ.microsoft.com to /default-rack`. Because the 2-gram `Resolved04DN8IQ.microsoft.com` has only 2 occurrences, whereas the 2-gram `to /default-rack`, which appears more frequently, the template generated for this log event is not valid: `* * to /default-rack`. The expected template is `resolved * to *`.

The ranking of the log parsing tools using the commission error ratio is shown in Figure 4.3. LogCluster (average commission error ratio = 74.99%) is by far the technique with the highest commission error ratio followed by Molfi (54.59%), Lenma (54.05%), and Logmine (53.67%).

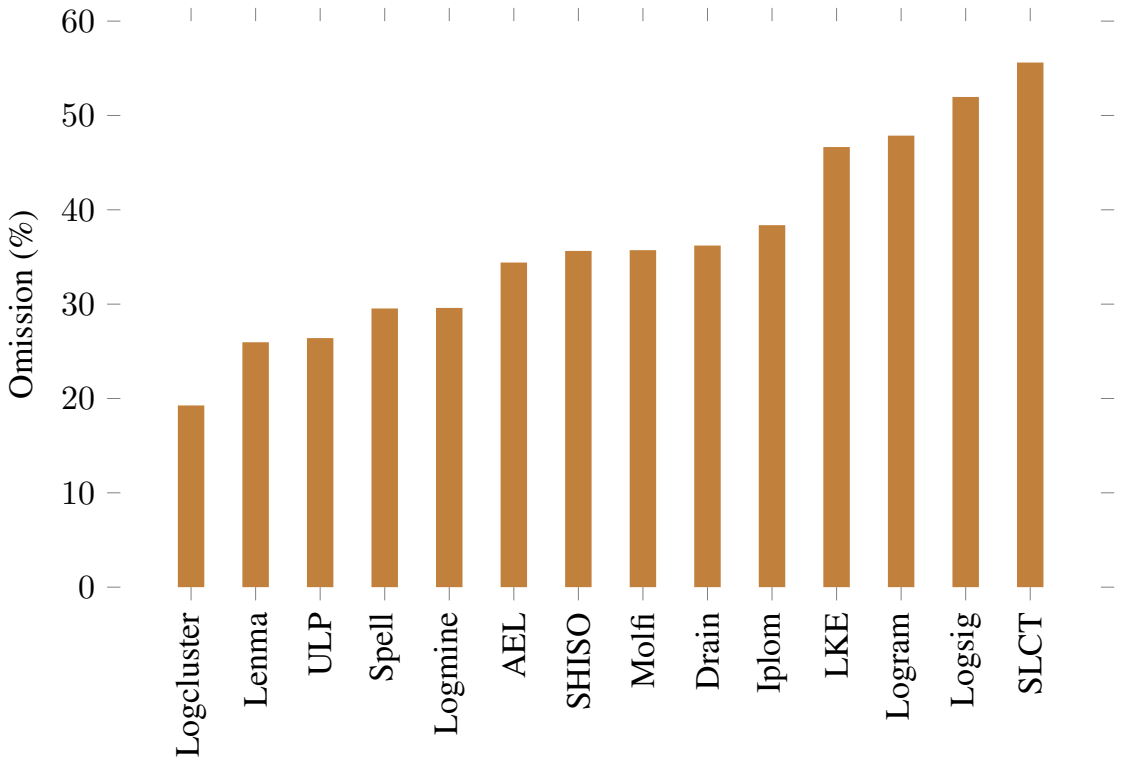

Figure 4.2: Ranking of log parsers by omission error ratio

4.5.2 RQ2. How do dynamic variables and log message density impact the performance of log parsing tools?

To understand the performance of log parsing tools using AML, we decided to investigate how dynamic variables and message density impact their performance. As a motivating example, after we analyzed manually more than 100 examples of log events that were misclassified by LogCluster, We found that the high commission error ratio is mainly due to the way the tool handles dynamic variables. We found that LogCluster does not always reproduce the exact position of the dynamic variables as that of the ground truth. For example, LogCluster parses the HDFS log event "`Deleting block blk_1781953582842324563 file /mnt/blk_1781953582842324563`" as `Deleting block file *{1,1}` as opposed to "`Deleting block * file *`, which is the correct template. The dynamic variable `blk_1781953582842324563` lost its position as the third item in the log structure. We also found a situation where the tool completely omits dynamic variables. For example, the log event "`BLOCK* NameSystem.delete:`

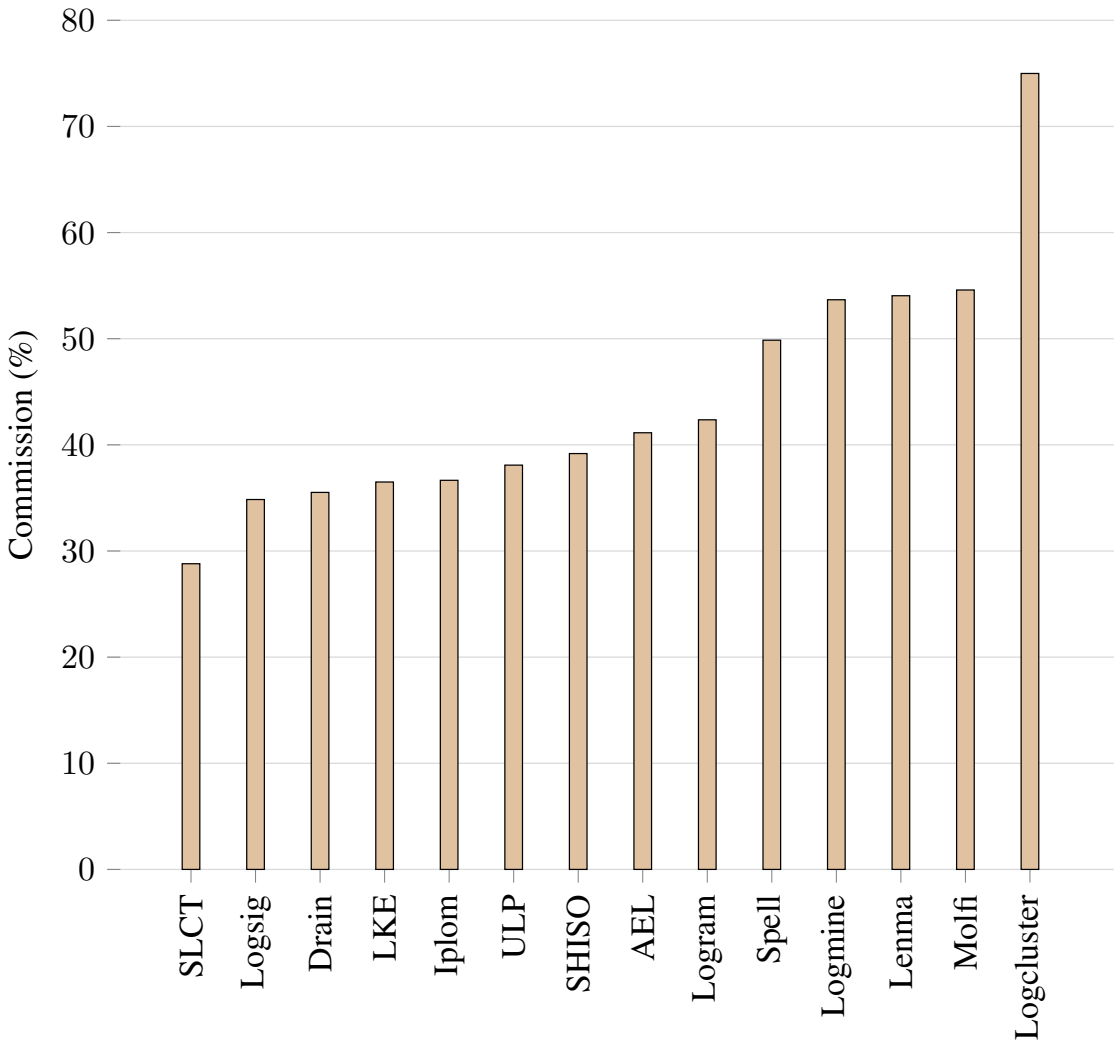

Figure 4.3: Ranking of the log parsers based on the average commission error ratio

Table 4.11: Log Parsing Techniques for Systems (HDFS to OpenSSH)

System	HDFS	Apache	Android	Windows	Mac	OpenStk	Linux	Openssh
ULP	0.5681	0.3333	0.4633	0.3866	0.4435	0.2800	-0.2802	0.0690
Drain	0.7055	0.3333	0.6605	0.3919	0.1709	0.0088	-0.2478	0.2439
SHISO	0.7500	0.3333	0.4313	0.2790	0.1455	0.1111	0.2274	0.1735
AEL	0.6663	0.3333	0.4976	0.3151	0.1356	0.1868	-0.1948	0.1731
Iplom	0.7100	0.3300	0.0000	0.2700	0.1400	0.2100	-0.1900	0.1600
Spell	0.7500	0.3333	0.7294	0.3312	0.1264	0.0123	-0.1964	0.2000
Lenma	0.7055	0.3333	0.6508	0.2976	0.1246	0.1155	-0.2550	0.1860
LogMine	0.6875	0.3333	0.3083	0.3247	0.1671	0.0097	0.2473	0.0107
LKE	0.7500	0.3333	0.7563	0.2877	0.0000	0.0242	-0.0767	0.0239
SLCT	0.2166	0.1429	0.7000	0.0506	0.0870	0.2679	0.0310	0.1260
Logsig	0.2760	0.1820	0.3450	0.1590	0.0720	0.2470	0.0400	0.1700
Molfi	0.0000	0.0000	0.1703	0.1770	0.0676	0.0018	0.2275	0.0612
Logram	0.0141	0.0009	0.3290	0.0806	0.1133	0.0000	0.0303	0.1172
Logcluster	0.0056	0.0000	0.1786	0.2414	0.0846	0.0023	-0.1906	0.0282
Average	0.4861	0.2373	0.4443	0.2566	0.1342	0.1055	-0.0591	0.1245

`blk_2568309208894455676 is added to invalidSet of 10.251.31.160:50010`" is parsed by LogCluster as " `BLOCK* NameSystem.delete : is added to invalidSet of`", ignoring the dynamic variables `blk_2568309208894455676` and `10.251.31.160:50010`. Dynamic variables can play an important role in debugging tasks, as shown by He et al. [79]. A log parser should be able to clearly recognize all the dynamic variables that are logged. Lenma, the tool with the third worst average commission error ratio, seems to suffer from the same design problems as LogCluster. We found many cases where Lenma, which also relies on clustering techniques, such as LogCluster, completely removes dynamic variables from the generated templates. For example, the Apache log event `jk2_init() Found child 6062 in scoreboard slot 9` is parsed as `jk2_init() Found child in scoreboard slot` by Lenma. Both the dynamic variables 6063 and 9 were ignored.

Table 4.11 and 4.12 show the AML scores from a system's standpoint. Systems like HDFS, Android, and Zookeeper generally enjoy higher average AML values, indicating often better AML performance with all the tested parsing techniques. While most techniques show variability in their efficiency across systems, some display remarkable consistency. For instance, Apache demonstrates a consistent value of 0.3333 across the majority of techniques (9 parsing techniques out of

Table 4.12: Log Parsing Techniques for Systems (Thunderbird to Proxifier)

System	Thunder.	Spark	Hadoop	Zookeeper	BGL	HealthApp	HPC	Proxifier
ULP	0.4103	0.3077	0.1684	0.6508	0.2542	0.7500	0.4774	0.4330
Drain	0.1871	0.2500	0.1361	0.3913	0.1443	0.0871	0.3433	0.3730
SHISO	0.1718	0.2241	0.1194	0.2333	0.0947	0.1681	0.0917	0.4612
AEL	0.1605	0.2100	0.1423	0.3100	0.1200	0.1500	0.2800	0.4400
Iplom	0.2200	0.2400	0.1300	0.4100	0.2000	0.1200	0.3700	0.4200
Spell	0.2350	0.2550	0.1150	0.3750	0.1550	0.1750	0.3600	0.4350
Lenma	0.1800	0.2300	0.1100	0.3200	0.1400	0.1300	0.3500	0.4150
LogMine	0.1950	0.2150	0.1000	0.3400	0.1300	0.1100	0.3400	0.4000
LKE	0.1700	0.2000	0.0900	0.3000	0.1200	0.1000	0.3300	0.3850
SLCT	0.1500	0.1800	0.0800	0.2800	0.1100	0.0900	0.3200	0.3700
Logsig	0.1650	0.1950	0.0700	0.2600	0.1000	0.0850	0.3100	0.3600
Molfi	0.1450	0.1750	0.0600	0.2400	0.0900	0.0800	0.3000	0.3500
Logram	0.1350	0.1650	0.0550	0.2300	0.0850	0.0750	0.2900	0.3400
LogCluster	0.1250	0.1550	0.0500	0.2200	0.0800	0.0700	0.2800	0.3300
Average	0.1780	0.2085	0.1006	0.3200	0.1270	0.1207	0.3400	0.4000

14). Linux exhibits high variability in results, suggesting its behavior is highly dependent on the technique applied. On the contrary, systems like Apache maintain consistent outcomes across methods, indicating lesser sensitivity to the applied technique. to investigate the inner characteristics of specific systems and how they may influence the parsing results. Specifically, we looked at how the density of dynamic variables within a log file can impact the AML score. Specifically, we define the density of dynamic variables as the ratio of dynamic variables to the total number of tokens in a log message. As illustrated in Table 4.13, systems manifest diverse densities of dynamic variables. Notably, high-density systems like HealthApp or Linux tend to exhibit lower AML scores in comparison to systems with lesser density such as Apache or Android. While density is a significant factor in influencing AML scores, it's paramount to consider it in conjunction with other variables such as the number of templates and their distribution or frequency. As detailed in Table 4.8, the template count and its distribution play a pivotal role in parsing complexity, indicating that density should not be assessed in isolation.

In evaluating the performance of log parsers, it is crucial to understand how varying densities of dynamic variables within log messages influence the accuracy of parsing, as measured by AML. The Pearson correlation coefficient is employed to quantify the linear relationship between the density of dynamic variables and the AML scores. This statistical method is chosen due to its effectiveness

Table 4.13: Density of dynamic variables in various systems

System	Dynamic variables density(%)
Thunderbird	12.47
Windows	15.75
Zookeeper	16.02
Android	16.70
Apache	16.95
HPC	17.76
SSH	18.60
Openstack	22.56
Hadoop	23.05
HDFS	23.18
Spark	25.32
Linux	26.93
Proxifier	29.74
Healthapp	44.25

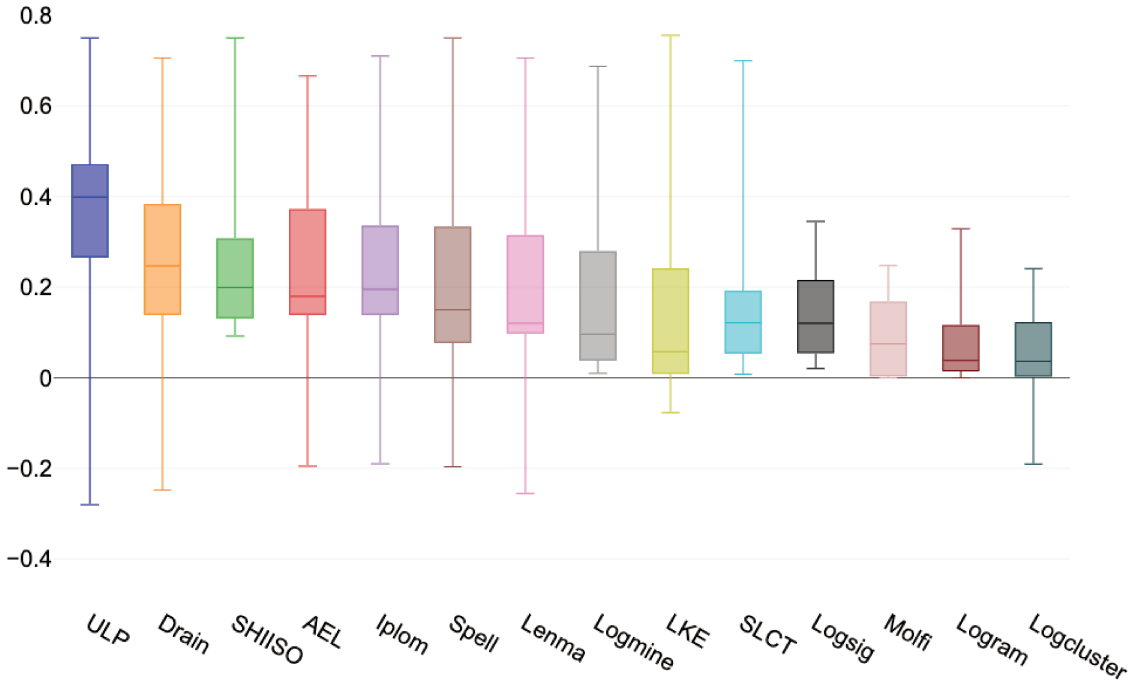

Figure 4.4: Accuracy distribution of log parsers across the datasets

in measuring the degree of a linear relationship between two continuous variables. This metric is particularly relevant in our context because it allows us to understand whether a higher density of

dynamic variables tends to complicate or facilitate the log parsing process, thereby influencing the AML scores. For instance, a significant negative correlation would suggest that as the complexity of logs increases (with more dynamic variables), the parsing accuracy decreases, posing challenges in parsing effectiveness. The correlation analysis highlights the varying impacts of dynamic variable density on the performance of different log parsers. It shows that the Average AML scores across parsers have a correlation of approximately 0.036 with density, indicating a very weak positive relationship. However, the complete analysis reflects a mix of weak positive and negative correlations between the dynamic variable density and the AML scores across different log parsers. ULP has a correlation of approximately -0.386 with dynamic variable density while Drain shows a correlation of approximately -0.120 with variables density. SHISO show a slight positive correlation with density indicating that as the density of dynamic variables increases, the AML scores might slightly increase as well. AEL, and Iplom show negative correlations. This suggests that for these parsers, higher density of dynamic variables could be associated with lower AML scores.
Log Parsers like ULP and SHISO that show a positive correlation likely have robust mechanisms for template matching that can effectively distinguish and correctly classify these variables. Parsers such as AEL might struggle with overfitting where the parser fits too closely to specific log patterns observed in training, failing to generalize well to new or slightly different log entries. Alternatively, these parsers might misclassify increased dynamic content as anomalies or errors.

In our analysis of log parsing inaccuracies impacting AML scores, certain dynamic variables were identified as major contributors to errors. The most frequent issues include effectively delineating log elements. Additionally, a significant number of errors are due to difficulties in parsing special characters. Misinterpretations of key-value pairs using colons as delimiters also pose significant challenges. Conversely, certain log attributes like MAC addresses, and URLs show less impact on AML score. This suggests that while certain formats consistently challenge parsing accuracy, others are managed more effectively, possibly due to parsers being optimized for these specific types.

4.5.3 RQ3.How does the performance of log parsing tools vary across different log datasets, and which datasets pose unique challenges?

Furthermore, we examined how the accuracy of the tools varies depending on each dataset. Figure 4.4 shows, using a boxplot, how the parsers compare to each other using AML as a measure of accuracy. The median of ULP is higher than that of all other tools, followed by Drain, SHISO, and AEL. LogCluster, Logram, Molfi, LogSig, and SLCT perform the worst across the datasets. In addition, we observe that the interquartile ranges (the width of the boxes) of box plots of all the tools show relatively high variability of the data, meaning that the accuracy of a log parser depends greatly on the structure of the log file itself. We attribute this to the lack of logging standards, which causes log files to vary in their content and structures, making it difficult to have a tool that can identify the structure of different log files in a consistent manner. Figure 4.5 provides a different

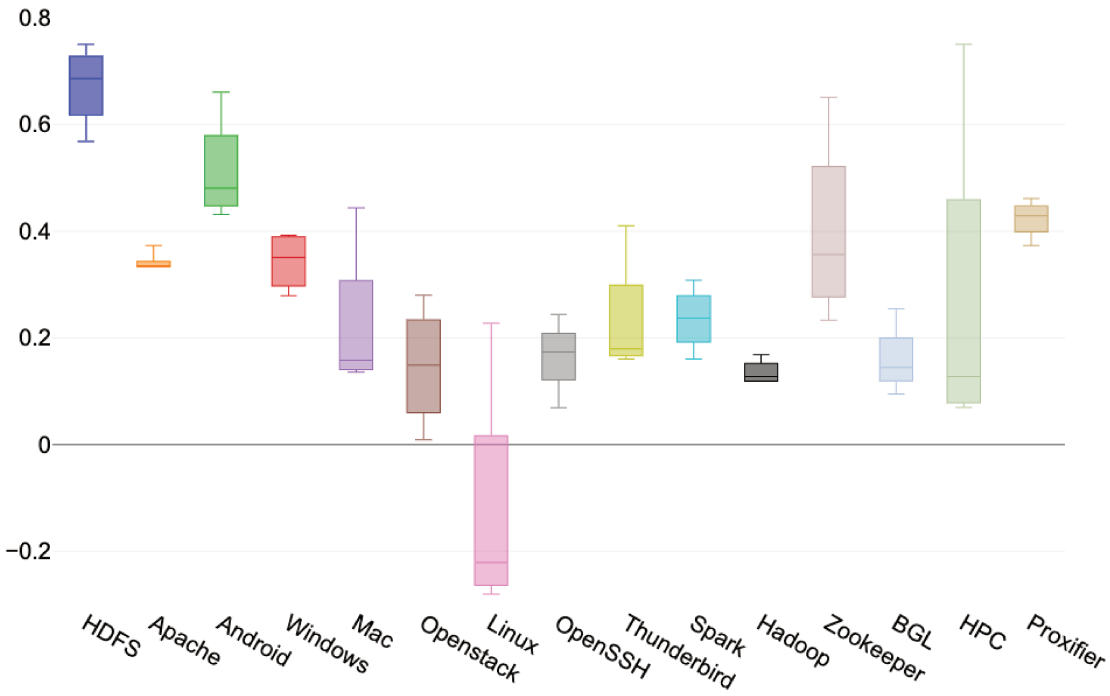

Figure 4.5: Variability analysis of ULP, Drain, SHISO, and AEL by dataset

perspective by examining the variability of the AML accuracy of the tools in parsing each dataset. In this figure, we focus only on the four most performing tools, namely ULP, Drain, SHISO, and

AEL. The table shows that the results of parsing HPC, Linux, and Zookeeper logs vary significantly compared to the other datasets. This may suggest that these log files are among the hardest to parse.

4.5.4 RQ4. How do these tools perform using AML compared to other accuracy metrics?

Table 4.14 shows the results of comparing the accuracy of the tools using AML, Grouping Accuracy (GA), Parsing Accuracy (PA), Precision Template Accuracy (PTA), and Recall Template Accuracy (RTA). As discussed in Section 4.2.5, we excluded the edit distance in this analysis. As we can see in the table, using GA, the accuracy of 7 out of the 14 tools (50%) is greater than 70% and 4 other tools have an accuracy of more than 60%. This is highly optimistic, considering the fact that the same tools perform poorly using the other metrics. For example, the accuracy of LogCluster using GA is 62.34%, while the accuracy of the same tool is 14.70%, 10.07%, 21.97%, and 5.68% using PA, PTA, RTA, and AML respectively. Similar results can be observed for Molfi and Logram. This is due to the fact that GA is primarily concerned with the way the log events are grouped together independently of the correctness of the log templates, which explains the major discrepancy between the measures. The difference between AML and PTA, as well as AML and RTA cannot be clearly deduced from the table. Statistical tests are needed to measure the magnitude of the difference between AML and these other metrics. We use Cliff's δ effect size [155] to assess the magnitude of the difference between the results obtained by AML and those of GA, PA, PTA, and RTA. Cliff's test is a non-parametric effect size measure that quantifies the magnitude of dominance as the difference between two groups X and Y [155]. Cliff's δ ranges from –1 to +1. A Cliff's δ that is equal to -1 means that all observations in Y are larger than all observations in X. It is equal to +1 if all observations in X are larger than the observations in Y. A δ value that converges to 0 indicates that the distribution of the two observations is identical.

Cliff's delta, denoted as δ, is given by the following equation:

$$\delta = \frac{\sum_{i,j}[x_i > x_j] - [x_i < x_j]}{m \times n} \tag{14}$$

where the two distributions are of size m and n with items x_i and x_j, respectively. Here, $[\cdot]$ is the

Iverson bracket, which is 1 when the contents are true and 0 when false [156].

The Cliff's δ effect size can be grouped into ranges. The effect is considered small for $0.147 \leq |\delta| < 0.33$, moderate for $0.33 \leq |\delta| < 0.474$, or large for $|\delta| \geq 0.474$ [155]. The analysis of the effect sizes using Cliff's δ between AML and GA, PA, PTA, and RTA has revealed varying degrees of differences. The effect size between AML and GA was found to be large (Cliff's δ = 1.00), indicating a significant difference between the two accuracy metrics. Similarly, for PA, the effect size was also large (Cliff's δ = 0.76). On the other hand, the effect size between AML and PTA was moderate (Cliff's δ = 0.43), suggesting a noticeable difference between the two metrics. The effect size between AML and RTA was large (Cliff's δ = 0.51), showing a significant difference between the two metrics.

Additional statistical tests are needed to measure the magnitude of the difference between AML and these other metrics. We use the Spearman Correlation Coefficient [155], which is a non-parametric correlation coefficient calculated using Equation 15 where d_i represents the difference between the two ranks of each observation and n represents the number of observations.

$$Rs = 1 - \frac{6 * \sum_1^n d_i^2}{n(n^2 - 1)} \tag{15}$$

Spearman's correlation coefficient ranges from -1 to +1. A positive correlation means that as one variable increases, the other variable also tends to increase. A negative correlation means that as one variable increases, the other tends to decrease. A strong correlation is reached when the value of the Spearman coefficient is close to -1 or +1. Values close to zero indicate weak to no correlation. The correlation analysis revealed that while AML has moderate to strong correlations with existing metrics, it also shows distinct behavior in certain scenarios. For instance, the strong correlation with PTA suggests that AML is robust in evaluating the precision of template-based parsing. However, AML's unique approach to evaluating log parsing performance also means it can capture aspects that other metrics do not, especially in complex parsing scenarios where traditional metrics might align but still miss critical errors.This is highlighted by its varying degrees of correlation across the board. The table below presents the correlation coefficients between various metrics and AML: The interpretation of correlation coefficients reveals nuanced insights into the relationship between

AML and other metrics. The moderate to strong correlation of 0.675 with GA suggests that while both GA and AML are related, AML extends beyond GA's scope by capturing additional aspects of log parser performance, particularly those not covered by simple grouping accuracy. Similarly, a strong correlation of 0.721 with PA indicates that AML not only aligns well with traditional parsing accuracy but also adds depth by addressing errors in template generation that PA might overlook. Moreover, the strongest correlation of 0.799 with PTA underscores that both AML and PTA assess similar characteristics of log parsers effectively. Although the substantial correlations are strong, they do not indicate perfect alignment, AML is particularly effective in complex parsing environments where both the presence and correctness of parsed templates are crucial. AML addresses the limitations of PA, which is notably sensitive to the frequency of templates. Additionally, unlike the PTA and RTA, which focus solely on the number of correctly identified templates, AML extends its assessment to ensure accurate parsing within these templates and across the entire log file.

4.5.5 RQ5. How can we use AML to analyze the sources of parsing errors?

One of the main advantages of AML over existing accuracy metrics is that it can be used to guide practitioners in understanding the root causes of parsing errors. This is made possible by the ability to see how a tool identifies each template by going into the level of log events associated with the template. This debugging mode is not possible using other metrics. To illustrate this feature, we take as an example the results obtained with AEL when applied to the Android log dataset. Table 4.16 shows sample templates from the Android dataset and the results of error ratios of omission and commission, as well as ICSI. A software developer may choose to dig deeper into one of these templates to understand the root causes of parsing errors. For example, for Template T22, of the three expected log events in the ground truth shown below with ids 39, 1267, 1377, AEL was able to correctly detect only two log events (an omission error ratio of 0.33). A tool that uses AML can generate a report that points out the parsing faults. In the case of template T22, AEL did not properly parse the log event 1267. The reason behind this is due to the presence of `-1` dynamic value, which confused the parser.

- 39: WindowManager: Application requested orientation 1, got rotation 0 which has compatible metrics

- 1267: WindowManager: Application requested orientation -1, got rotation 0 which has compatible metrics
- 1377: WindowManager: Application requested orientation 1, got rotation 0 which has compatible metrics

Using AML with its components, omission errors, commission errors, and ICSI, we analyzed hundreds of log events to understand the root causes of most parsing issues. We've identified some prevalent challenges that often culminate in parsing errors. These errors inevitably impact the overall accuracy of the log parsers as shown in Table 4.17.

One of the pivotal challenges is the inherent variability in the data types in logs. Tokens can exhibit diverse formats, making it a complex task for log parsers to accurately differentiate between dynamic and static parts. This variability is often manifested in elements like IPv4 and IPv6 tokens, domain names, and universally unique identifiers (UUIDs). Each presents a unique set of characteristics that requires tailored parsing strategies.

Adding to the complexity are the intricate structures within the log events. Nested or non-linear formats introduce an additional layer of complexity, making the extraction of relevant information a non-trivial task. Multi-level nested tokens illustrate such complex structures that demand advanced parsing techniques capable of unraveling the embedded information accurately.

Furthermore, the distinction and separation of dynamic and static tokens in log events are pivotal factors that significantly influence the accuracy of log parsing. Each system might adopt a diverse set of techniques for this separation, employing delimiters such as spaces, commas, or others to distinguish between different types of tokens. It's worth noting that many log parsing techniques, including those cataloged in Loghub benchmark [151] [5] often necessitate the specification of these delimiters during the pre-processing phase to streamline the parsing process.

Such diversity in separation methods underscores the necessity for log parsers that are not only robust but also versatile. The ability to adapt to and efficiently process a variety of separation techniques becomes instrumental in enhancing the precision and reliability of log parsing.

Log events are also characterized by their unusualness - the presence of unexpected tokens,

[5] https://zenodo.org/record/3227177#.YUqmXtNPFRE

formats, or patterns. Such elements can trigger parsing errors if not aptly managed. Instances like non-standard MAC address formats or URLs with query parameters are cases in point, requiring specific attention to ensure accurate parsing.

4.6 Discussion on improving log parsing

This study provided us with important insights on how to improve log parsing. We discuss key lessons learned in this section as well as opportunities on improvement of the AML metric. The ultimate objective is to encourage the adoption of AML for effectiveness evaluation of log parsing techniques.

4.6.1 Implications of AML in the development of future log parsers

Using AML, developers of log parsers can focus on reducing errors of ommision and commission both at the log message and template levels. This can be done in several ways. The first one is to consider similarity instead of exact match when assessing the degree by which two log messages below to the same template. Based on our experiments, we found that many parsing errors are caused by extensive use of special and alphanumerical characters. Exact match will almost always lead to errors. More on this in Section 6.3 where we discuss the concept of intelligent parsing. Additionally, log parsers can use AML to test their parsers and identify areas where they fail the most. For example, some parsers may be good at identifying log messages, but introduce many false positives and new templates. Others may be good at detecting just the right number of templates, but fail in associating log messages to these templates. The ommission, commission, and ICSI can be readily used to assess the strengths and weaknesses of a parser.

4.6.2 Problems with logging practices

Parsing errors are mainly caused by the inability to distinguish between static and dynamic tokens. We found that this issue can be reduced if better logging practices are adopted. For example, consider the following two log events of the HDFS dataset: `transmit 1 2 3` and `transmit blk123`. Parsing these events will lead to two templates, namely, `transmit <*> <*> <*>`

and `transmit <*>` despite the fact that both events log the same information, which consists of transmitting data blocks 1, 2, and 3. Another example would be in the HPC log events `Fan speeds ( 3552 3534 3375 **** 3515 3479)` and `Fan speeds (3552 3534 3375 11637 3515 3479`. The inconsistencies in the way the logging statements corresponding to these events are written tend to mislead most parsers we examined in this paper. For example, some parsers mistakenly generated two different templates, namely `Fan speeds (<*> <*> <*> **** <*> <*>)` and `Fan speeds (<*> <*> <*> <*> <*> <*>)`. The lack of guidelines for logging has been reported in many studies such as the work of Keyur et al. [157] on the practice of logging in the Linux system, and the work of Chen et al. [158] on the logging practices in Java applications. We believe that improving logging practices by following some standardized ways to write logs can lead to improved parsing.

4.6.3 Intelligent parsing

One possibility to prevent parsing errors would be to have a log parser that supports similarity instead of exact match when comparing log events. For example, the two HPC log events `not responding` and `not-responding` can be considered similar and therefore mapped to the same template. We should also design parsers that can predict the structure of a log event by learning a partial representation of the template. For example, for the log event `Received block blk_1687916 of size 910 from 10.240.15.214`, it may be sufficient to recognize part of the template (let us say `Received block * of`) in order to classify unseen log events. The remaining dynamic variables are still to be identified later. However, this is much simpler to achieve than having to identify the exact template when dealing with a diverse set of log templates. Partial matching should be exercised with care to avoid discarding important data that can affect log analytics tasks. In addition to this, we found many instances where parsing can be made accurate if the parser has the ability to recognize logs with similar semantics. For example, the two log events `packet sent to 16` and `block sent to 45` from the HDFS log dataset can be parsed correctly if we treat ”packet” and ”block” as synonymous since both logs refer to ”data sent to a port number”. Semantic analysis of logs can further reduce ambiguities due to

the inherent imprecision of natural language. Future research should focus on developing an intelligent parser that considers the semantics of logs, perhaps by integrating natural language processing techniques with log parsing. In addition, a good parser should be able to check for spelling errors, acronyms, and other imprecisions pertaining to the use of natural language that almost always lead to parsing errors.

4.6.4 Ground truth datasets to train the parsers

In this study, we used 2,000 log events from each dataset to test the parsers. These datasets contain errors that have misled many of the parsers we used. For example, in the Mac log file, the token `CrazyIvan46!` in the log event `CI46 - Perform CrazyIvan46!` is considered static, while the token `CrazyIvan46` refers to a username and, therefore, should be dynamic. Another example would be the case of the HDFS log dataset, where block ids such as `blk_23333989` are sometimes labeled * and other times as `blk_*`. The dynamic variables that appear frequently are mistakenly parsed in the ground truth as static tokens. For example, the variable `fecd:467f` appears 18 times in the Mac dataset without any change. It was parsed as a static token, while it should be dynamic. Another common error in the ground truth consists of considering static variables, which are syntactically similar to dynamic variables, as dynamic variables. For example, in the Hadoop log file, some parsers interpret the log event `jetty-6.1.26` as $\star$ despite the fact that `jetty-6.1.26` refers to static content. This type of static token is the hardest to detect because it bears most of the characteristics of dynamic variables. In this paper, although we have made every effort to fix the ground truth of the datasets used in this study, we believe that cleaner and larger ground truth datasets are desirable.

4.6.5 Weighted AML

The introduction of weights to the AML metric could enable a context-driven evaluation of log parsing errors.

The allocation of higher weights to certain templates would underscore scenarios where missing certain log events or templates is deemed more critical, possibly due to their role in security, system performance monitoring, or other tasks.

We can calculate the Weighted AML score using the formula:

$$WeightedAML = \frac{\sum_{1}^{\max(N,D)}(WeightedICSI_i)}{\max(N,D)} \tag{16}$$

Assuming we are evaluating a log parsing tool for network security with a ground truth dataset of five predefined log templates:

- Template A: Firewall Rule Updated - Firewall rule updated by user [username] for IP [IP]
- Template B: Suspicious Activity Detected - Suspicious activity detected from IP [IP]
- Template D: System Error - System error: [error message]

Template B, followed by D can be given a higher weight due to their sensitivity. Table 4.18 compares normal AML to weighted AML.

4.6.6 Diagnostic insights in template identification

The AML model in this paper is built on the principle that precise template identification is central to effective log parsing. We aim for a ”perfect match,” where every element of a parsed log event aligns flawlessly with its ground truth. However, in the practical world of log parsing, partial identifications are common and can be equally valuable. These are instances where not all dynamic tokens are identified but substantial portions are, offering useful insights. In these cases, static tokens within the log events maintain their categorization, illustrating a balance between accuracy and completeness. While our focus is on reducing these errors, we acknowledge that the path to perfection is paved with instances of partial identifications, each bringing us a step closer to optimal log parsing.

Even though our current AML model does not incorporate partial identifications, practitioners often find value in partial identifications as they can significantly reduce the manual workload, making the parsing process more manageable and efficient.

In the landscape of log parsing metrics, the introduction and systematic analysis of partial identifications remain largely unexplored. By including partial identifications in the evaluation process,

we believe future iterations of log parsing models will offer more nuanced, practical, and actionable insights.

Each partial identification, in its unique way, draws us closer to a world where log parsing is not just about perfection but about practicality and efficiency.

4.6.7 Threats to Validity

Internal validity: Threats to internal validity are associated with factors that may impact our results. Many of the tools we used in this study contained bugs and old libraries. We had to fix these bugs, and update the libraries. We tested the new versions thoroughly to ensure that the changes we made did not impact the functionality of the tools so as to minimize potential internal threats to validity. In addition, we corrected minor errors in the datasets as discussed in Section 4.4.2. We carefully checked every log template and made sure that our corrections did not alter the log event structure represented by these templates.

External validity: Threats to external validity are related to the ability to generalize our results. To support generalizability, we used 16 log datasets generated from a variety of software systems and experimented with 14 log parsing tools, which cover most of the tools that exist in the public domain. Although more studies should be conducted to fully generalize our results, we believe that this threat to validity is greatly minimized considering the number of datasets and tools we have used in this paper.

Reliability validity: Reliability validity concerns the ability to replicate this study. To mitigate this threat, we put all the data used in this paper online, including the detailed results of parsing 16 log datasets using 14 parsers. Data can be found in Zenodo :
https://zenodo.org/record/7872794#.ZEsHxezMJhE

4.7 Future Work

The soundness of AML is supported by its comprehensive evaluative scope, adaptability, quantitative rigor, and potential for empirical validation. AML distinctly evaluates errors of omission and

commission, which are critical for understanding the nuances of information loss and misinterpretation in log parsing. While this paper establishes a foundation for evaluating log parsing tools using the AML metric, there are additional validation strategies that we believe could further substantiate the effectiveness of AML. However, these strategies fall outside the current scope of our research and are suggested as directions for future work: Future research could include conducting empirical studies to examine the correlation between AML scores and the success of downstream tasks such as Anomaly Detection, System Monitoring or Predictive Maintenance. Additionally, incorporating human evaluation can provide valuable insights into the practical utility of AML compared to other metrics. Participants would perform typical log management tasks using outputs from parsers with different AML scores. After task completion, participants could provide qualitative and quantitative feedback on the ease of use, effectiveness, and overall satisfaction with the parsed data and benefits in terms of task performance and data usability.

4.8 Replication Package

The datasets, scripts and results are available on Zenodo:
https://zenodo.org/record/7872794#.ZEsHxezMJhE

4.9 Acknowledgment

This work is partly supported by the Natural Sciences and Engineering Council of Canada (NSERC).

4.10 Conclusion

In this study, we introduced AML, an innovative accuracy metric tailored for log file parsing. AML uniquely assesses the accuracy of log parsers by quantifying both omission and commission errors at both the template and log file levels, all encapsulated within a single metric. Through a comprehensive evaluation, we have demonstrated that AML surpasses existing accuracy metrics in terms of reliability and ease of use for log parsing tasks.

Our extensive experimentation involved 14 log parsers applied to 16 log file datasets from the LogHub benchmark, providing robust evidence of AML's effectiveness as a superior accuracy measure. Furthermore, we conducted a comparative analysis, pitting AML against other established accuracy metrics, highlighting its superiority in providing a more nuanced and dependable assessment of log parsing performance.

Beyond its role as a reliable evaluation metric, AML offers an additional dimension of utility. It empowers practitioners with the ability to dissect and comprehend the root causes of parsing errors, opening up new avenues for troubleshooting and refinement in log parsing processes.

As we look toward the future, there exist promising directions for advancing AML and enhancing log parsing tools. Future research endeavors could explore the introduction of weighted considerations for omission and commission errors or fine-tuning AML for specific log templates, thereby tailoring the metric to specific application domains. Additionally, the evolution of log parsing tools should incorporate intelligent parsing approaches that leverage semantics alongside syntax, addressing the complex and evolving nature of log data. Integrating natural language processing capabilities could prove pivotal, especially when dealing with the inherent ambiguities and imprecisions found in natural language logs, even in cases lacking standardized logging practices.

Table 4.14: Comparison of the performance of tools using AML and other accuracy metrics.(%)

Parser	AML	GA	PA	PTA	RTA
ULP	35.72	73.34	54.62	29.69	33.08
Drain	26.12	86.54	55.44	28.65	30.55
Shiso	25.10	64.82	24.53	16.75	17.08
AEL	24.51	75.94	52.56	25.99	28.13
Iplom	23.52	75.89	39.34	15.83	15.41
Spell	22.46	79.26	54.22	16.46	18.56
Lenma	19.97	73.33	29.18	16.47	23.38
LKE	16.85	60.18	9.97	11.23	12.72
Logmine	16.76	70.39	29.69	15.53	20.17
SLCT	15.57	59.28	45.71	21.84	16.04
LogSig	14.52	50.39	16.58	14.76	10.22
Molfi	9.70	60.10	10.85	9.07	11.20
Logram	8.57	55.47	25.74	13.04	14.46
LogCluster	5.68	62.34	14.70	10.07	21.97

Table 4.15: Correlation of different metrics with AML

Metric	**Correlation with AML**
GA	0.675
PA	0.721
PTA	0.799
RTA	0.646

Table 4.16: An example of parsing results of AEL when applied to Android logs. ELE stands for Expected Log Events and DLE stands for Detected Log Events

Template ID	Identified Template	ELE	DLE	Omission	Commission	ICSI
T22	Application requested orientation *, got rotation * which has compatible metrics	3	2	0.33	0.00	0.67
T23	applyOptionsLocked: Unknown animationType=*	2	2	0.00	0.00	1.00
T25	Bad activity token: *@*	1	1	0.00	0.00	1.00
T26	battery changed pluggedType: *	1	1	0.00	0.00	1.00
T27	cancelAutohide	15	15	0.00	0.00	1.00
T28	cancelNotification, cancelNotificationLocked, callingUid = *,callingPid = *	2	2	0.00	0.00	1.00
T29	cancelNotification,index:*	23	3	0.87	0.00	0.13

Table 4.17: Challenges in Log Parsing with Examples

Challenge	**Examples**
Unseparated Token Sequence	`2022-03-15T13:45:32+00:00[ERROR] 500InternalServerErrorID:123456`
UUIDs or IDs	`ID:abcd1234efgh5678, ID:5678abcd-1234-efgh`
Datetime Tokens	`Timestamp:[2022/03/15::13:45:32+00]`
MAC Addresses	`Connected Device MAC=00-11-22-33-44-55, MAC=00:11:22:33:44:55`
URLs with Query Parameters	`GET https://example.com/api?user=123&status=active& role=admin`
Multi-level Nested Tokens	`Event [Timestamp:(2023-09-20 14:23:00) Details:(Error:Failed to connect)]`
Alphanumeric & Special Characters	`EventID:#A1b2_c3! - User 'john.doe' authentication success`
Delimiter Variations	`INFO [2023-09-20] - User:john.doe | IP:192.168.1.1, Status=Active`
IPv4 and IPv6 Addresses	`IPv4: 192.168.1.1, IPv6: 2001:0db8:85a3:0000:0000:8a2e:0370:7334`

Table 4.18: Comparison between Normal AML and Weighted AML

Aspect	**Normal AML**	**Weighted AML**
Metric Calculation	$\frac{\sum_1^{\max(N,D)} ICSI_i}{\max(N,D)}$	$\frac{\sum_1^{\max(N,D)} (WeightedICSI_i)}{\max(N,D)}$
Weighting Criteria	Equal weight for all templates	Custom weights based on log template importance/criticality
Interpretation	General performance assessment	Context-driven analysis
Additional Information	Limited insight into error impact	Highlights specific template-related errors
Use Cases	Standard evaluation	Security or domain-specific evaluation

Chapter 5

Taxonomoy of LECs

"Order and simplification lay the foundation for mastery. Through structure, we transform complexity into understanding."

— Adapted from Thomas Mann

From AML to Log Taxonomy: With AML providing a fine-grained lens for evaluating log parsers, we gain new insights into the specific kinds of errors and inaccuracies that impede robust parsing. While AML effectively captures both omission and commission errors, the root causes behind these mistakes often trace back to the structural and contextual complexities of the logs themselves, properties that vary significantly across different applications and platforms. This realization naturally leads us to explore the deeper characteristics of log data that can confound even well-validated parsers. Accordingly, the next chapter introduces a taxonomy of log characteristics, outlining how

diverse structural elements, from nested tokens to inconsistent delimiters, can systematically challenge parsing methods. This taxonomy not only illuminates why certain log formats prove particularly difficult, but also provides a framework for creating parsing strategies better aligned with real-world logging variability.

I. Sedki, A. Hamou-Lhadj, O. Ait-Mohamed and N. Ezzati-Jivan, "Developing a Taxonomy for Advanced Parsing Techniques," In Proc. of the 33rd ACM/IEEE International Conference of Program Comprehension (ICPC), Ottawa, ON, Canada, April 2025

Abstract :Logs are indispensable for debugging, anomaly detection, and failure prediction in software engineering. However, the complexity and diversity of log data across different systems pose substantial challenges to existing parsing methods. Despite efforts using heuristic-based, machine learning, and neural network approaches, none have consistently achieved high accuracy across diverse systems. This highlights a critical gap in understanding how log data characteristics and patterns affect parsing effectiveness.

To address this challenge, our study conducts a comprehensive qualitative and quantitative analysis of 16 heterogeneous log datasets, documenting the structure, components, and sequence patterns within these logs. From this analysis, we derive a detailed taxonomy that categorizes the observed patterns, providing a new perspective on log data.

The resulting taxonomy offers valuable insights for developing next-generation log parsing tools that are intelligent and adaptable, capable of handling the unique characteristics of diverse log formats with improved accuracy.

5.1 Introduction

Logs are generated during software execution by embedded logging statements, capturing a wide range of events such as system errors, user activities, and operational status updates [115]. These logs play a crucial role in numerous software engineering tasks, including debugging [3, 43, 4], debugging and comprehension of system failures [44][45][46][47][8], testing and performance analysis [48][37] [49], operational intelligence [8] [37][50][5], data leakage detection [51], failure detection [46], failure prediction [8][**?**],anomaly detection [3, 4, 2**?**], and AI-driven IT operations [1, 2]. Their versatility makes logs an indispensable resource for maintaining the reliability, performance, and security of software systems [**?**].

Log parsing is a technique that is used to automatically extract structures (also known as templates) from heterogeneous and unstructured log events. Consider the log event illustrated in Figure 5.1; the corresponding log template would be `Received Block ⋆ of Size ⋆ from` . Here, the ⋆ symbol represents dynamic tokens. One may think that extracting such structures can be achieved using traditional regular expressions [117][61]. The problem is that typical log files contains thousands of templates [50] [118] [47], which makes it almost impossible to use regular expressions. In addition, as the system evolves, new structures emerge requiring constant and costly domain-driven updates of the Log parsing algorithms [80] [132, 133].

The increasing importance of automated log parsing has driven the development of numerous log parsers, utilizing diverse methodologies such as machine learning, pattern mining, natural language processing (NLP) [12, 13, 14], and Large Language Models (LLMs) [15, 16, 17, 18], reflecting the growing interest in LLM-driven solutions. A comprehensive survey of these tools is provided by El-Masri et al. [19]. Examples of notable log parsers include Drain [20], Spell [21], SHISO [22], AEL [23], Lenma [24], LogSig [25], SLCT [26], IPLoM [27], LKE [28], LogMine

Logging Statement: `LOG.info("Received Block "+ block_id + " of size " + block_size + " from " + ip)`
Log Event: `270423 283349 9876 INFO dfs.DataNodeResponder: Received block blk_-1680 of size 4536 from 10.163.23.167`
Log Template: `Received Block <⋆> of size <⋆> from <⋆>`

Figure 5.1: An example of a logging statement, a log event, and the corresponding log template

[29], Logram [13], and ULP [30].

The lack of standardized guidelines and best practices for logging [31, 32] has led to significant variability in log structures across different systems [33]. This diversity in log formats, coupled with the ever-growing volume of log data [34, 35, 36], presents considerable challenges for log parsing. Consequently, effective analysis and troubleshooting of systems become more difficult, often hindering the speed and reliability of system diagnostics and maintenance [37, 38].

Achieving high accuracy across diverse log formats remains a persistent challenge for most existing parsers[19, 77, 20, 13]. This difficulty is partly due to the predominant focus in research on developing novel parsing algorithms, while insufficient attention has been given to understanding the root causes of log parsing errors.

Existing research on software logging and log parsing can be broadly categorized into four main areas: characterization studies (e.g., [61]), automation techniques [68], machine learning models [9], and evaluation studies [74, 70]. Existing Characterization studies (e.g., [32]) primarily explore logging practices from a coding perspective, examining how logging statements are created and incorporated into software and their possible impacts. However, these studies do not sufficiently focus on the analysis of generated log events or the specific characteristics that often lead to parsing errors.

While these contributions have advanced log parsing capabilities, there remains a significant gap in the understanding why log parsing yield many errors and a low accuracy overall. Few studies have thoroughly investigated the root causes of these errors, despite their critical impact on downstream log analysis tasks, such as anomaly detection and failure diagnosis. Even minor parsing inaccuracies, as low as 4%, can significantly impact performance, potentially resulting in effects amplified by an entire order of magnitude [79].

In this study, we conduct a characterization study using 16 log datasets and eight log parsers to reveal the LECs that lead to parsing errors. Using the open coding research methodology [159], we group these LECs into categories to form a taxonomy of LECs that describes the components, patterns, variables, and structures within log events that commonly cause parsing errors. We also study the effect of LECs on parsing tools, and the difference between various datasets. The taxonomy provides a deep understanding of the complexity of log data, which can guide the development

of advanced log parsing tools and better logging practices.

The remainder of this paper is structured as follows: In the next section, we present our methodology. Section III delves into the results where we describe the taxonomy of LECs, their impact on log parsing tools, and the relationship between LECs and parsing errors across different datasets, along with key insights that inform the future of logging. Section IV discusses the broader implications of our findings, followed by conclusions and directions for future research.

5.2 Methodology

In this study, our main objective is to create a comprehensive taxonomy of LECs that commonly lead to parsing errors. Unlike previous research that evaluates specific log parsers, we focus on identifying LECs that are consistently problematic across multiple parsers and datasets.

By analyzing errors generated by eight well-known log parsers on 16 diverse log datasets, we uncover common LECs that challenge parsers, regardless of the algorithms used. The outcome of this analysis is a taxonomy of LECs that lead to parsing errors. We also apply the taxonomy to understand the impact of LECs on various parsing tools. To achieve this goal, we formulated the following research questions:

RQ1: What are the types of LECs that lead to parsing errors? This research question seeks to systematically identify and categorize LECs frequently associated with parsing errors. By understanding these characteristics, we can pinpoint specific log data features that present challenges for existing parsing tools, leading to inaccuracies or failures. Categorizing these LECs allows researchers and developers to refine parsers to better address these critical aspects.

RQ2: How do different log parsing techniques correlate with their effectiveness in handling various LECs? This question explores how different log parsing techniques perform in relation to specific LECs. Log parsers often make assumptions about log structure, such as consistent token placement, delimiter use, or expected patterns. These assumptions can make parsing effective for some logs but can lead to errors for logs that deviate from these expectations. By analyzing how different parsing techniques—such as regular expressions, pattern matching, token counting, and positional assumptions—handle various LECs, we aim to understand which methods are most

effective and why.

RQ3: How do LECs shape log patterns and influence parsing difficulty across different datasets? This question explores the role of LECs in shaping log patterns and clarifies why certain datasets are more challenging to parse. By defining essential log components such as dynamic values, nested tokens, punctuation, LECs fundamentally influence log structures and the resulting formatting patterns. The complexity, variability and frequency of these LECs are key factors in determining the difficulty of parsing a dataset. Datasets with consistent and simple LECs often result in predictable log patterns that are easier for parsers to handle. In contrast, datasets containing diverse, frequent, or deeply nested LECs introduce structural variability that complicates parsing. Understanding how LECs form log patterns helps in developing tools that can recognize and adapt to these patterns, ultimately improving parsing accuracy and efficiency across diverse datasets.

5.2.1 Log Datasets

We used datasets from the Loghub repository[151][1], covering systems like HDFS, Hadoop, Spark, OpenStack, BGL, HPC, Windows, Android, Apache, and more. These diverse datasets provide a comprehensive corpus for analyzing LECs.

Each dataset in the LogHub benchmark includes 2,000 manually parsed log events, serving as a ground truth for our analysis. These events, along with their corresponding log templates, are crucial for validating the accuracy of log parsers. For instance, the Apache dataset includes templates such as "`jk2_init() Found child * in scoreboard slot *`" and "`mod_jk child workerEnv in error state *`".

In our examination, we processed a total of 32,000 log events across the 16 datasets using eight log parsing tools.

[1] https://zenodo.org/record/3227177#.YUqmXtNPFRE

5.2.2 Log Parsing Tools:

Our selection of log parsers is grounded in the resources provided by LogPAI—which offers a wealth of log parsing tools and is well-recognized within the academic community—and two comprehensive surveys on log parsing by Zhu et al.[146] and El-Masri et al.[33]. From the parsers included in the LogPAI benchmark, we chose eight prominent tools: Drain[20], IPLoM[27], AEL[23], Spell[160], LenMa[24], LogMine[29], SHISO[22], and ULP[30]. These parsers are among the best-performing tools identified in prior surveys [146, 30, 33], representing a diverse array of parsing methodologies and algorithms. We configured each tool according to the guidelines provided in their respective publications, ensuring that they operate as intended for our analysis.

In this study, we selected representative log parsers from three distinct categories—clustering-based approaches (LogMine, Lenma), heuristic-based methods (Drain, Shiso, Spell, AEL, and IPLoM), and frequent pattern mining (ULP)—to examine their performance against various Log Event Characteristics (LECs). **Clustering-based parsers** organize log messages into hierarchical, adaptable clusters, offering users granular control over clustering parameters to enhance analytical depth. **Frequent pattern mining** methods focus on identifying recurring sequences in logs by applying predefined thresholds, which helps in effectively extracting frequent patterns from log data. **Heuristic-based** techniques leverage domain-specific rules and specialized data structures to generate log templates and improve parsing efficiency [161]. These three categories have been widely studied in the literature [77, 19] and are recognized for their potential to achieve robust results. Some of these tools, such as Drain, have even gained traction in the industry [2].

5.2.3 Identifying Log Event Characteristics (LECs)

We used open coding, a qualitative research technique, to identify LECs leading to parsing errors [159]. Open coding allowed us to explore complex, unstructured log data without predefined categories.

We used eight renowned log parsers to process 16 diverse log datasets, generating both successful parses and parsing errors. Parsing errors were identified by comparing parser outputs to the

[2]https://pypi.org/project/drain3/0.6/

ground truth templates provided in the datasets.

To extend our analysis across all 2,000 log events per dataset, we automated the identification of LECs using regular expressions and named entity recognition (NER). For instance, IPv4 addresses were detected using a regex pattern, while file paths were identified based on Unix and Windows formats. Automation enabled efficient processing of large datasets, supplemented with manual reviews for accuracy.

Through an iterative coding process, we categorized LECs into broader groups, such as "Structural Characteristics" for nested tokens and "Data Types" for timestamps, IP addresses, and file paths. This categorization formed the basis for a taxonomy that provides insights into common patterns and parsing challenges.

5.3 Results and Discussion

5.3.1 RQ1: Types of LECs and their categories

To identify the types of LECs, as discussed earlier, we applied the selected eight log parsers to 16 log datasets. All log parsers successfully processed the datasets, except IPLoM, which timed out on the Android dataset despite multiple attempts. The data and results used in this paper are available on Zenodo [3]

After analyzing the results, we categorized LECs based on their impact on log data parsing. Recognizing that overlaps between categories are inevitable due to the multifaceted nature of log events, we crafted each category to emphasize distinct aspects that contribute to parsing challenges. We identified **30 distinct log characteristics** that we grouped into **3 primary categories**, namely Log Event Presentation, Data Types, and Structural Arrangement of Tokens. Each category addresses unique aspects of logs, from the variability of content, to the relationships between elements, to the formatting of events. By combining these perspectives, the taxonomy captures the breadth of challenges that parsers must address. Each category targets distinct yet interconnected aspects of log data that collectively determine parsing complexity.

Table 5.1 shows the distribution of LECs in correctly parsed (C.P) and incorrectly parsed (I.P)

[3]The results of the study are available at: https://doi.org/10.5281/zenodo.7868027

Table 5.1: Characteristics Occurrence in Correctly and Incorrectly Parsed Log Events Across Log Parsers

Legend: C.P = nb of occurrences of the characteristic in the correctly parsed logs, I.P = nb of occurrences of the characteristic in the incorrectly parsed logs

Characteristic	AEL		Drain		Iplom		Lenma		Logmine		SHISO		Spell		ULP	
	C.P	I.P	C.P	I.P	C.P	I.P	C.P	I.P	C.P	I.P	C.P	I.P	C.P	I.P	C.P	I.P
Alphanumeric and special characters	3570	9731	3421	9880	3475	9455	3682	9619	3370	9931	3581	9720	3363	9938	3282	10 019
Log event with only static tokens	1548	180	1546	182	1525	203	1552	176	1447	281	989	739	1463	265	1526	202
UUID	228	562	16	774	251	539	357	433	79	711	268	522	101	689	301	489
Single-level nested tokens	1301	2460	1243	2518	923	2571	1143	2618	453	3309	1220	2542	1230	2531	1201	2561
Multi-level nested tokens	29	64	24	69	0	29	24	69	2	91	29	64	2	91	29	64
Enclosed quotations	92	1311	161	1242	91	1194	154	1249	97	1306	124	1279	157	1246	193	1210
URL with query parameters	0	11	0	11	0	11	0	11	0	11	0	11	0	11	0	11
Equals-separated key-value Pairs	2395	2917	2685	2627	2205	1994	1904	3408	2350	2962	1895	3417	2576	2736	2926	2386
Key-value pairs formed with a colon	5354	8934	5619	8669	4755	8753	5922	8366	4541	9747	5637	8651	4967	9321	5590	8698
Word-number pair	5933	4628	6505	4056	5131	5257	5682	4879	4260	6301	5431	5130	5911	4650	6411	4150
Folder structure	2268	2677	2311	2634	1821	3021	2356	2589	1765	3180	2314	2631	1811	3134	1886	3059
Datetime tokens	2697	4088	3090	3695	2463	4216	2361	4424	1706	5079	2628	4157	2555	4230	2863	3922
Time duration	90	788	47	831	90	788	116	762	63	815	79	799	64	814	68	810
URL	0	27	0	27	0	27	0	27	0	27	0	27	0	27	0	27
Decimal	3919	6388	4530	5777	3486	6735	4491	5816	2834	7473	3971	6336	4133	6174	3575	6732
Data volume and unit	372	1244	316	1300	373	1243	404	1212	11	1605	393	1223	340	1276	360	1256
Nouns	573	1404	578	1399	199	1651	608	1369	537	1440	581	1396	212	1765	355	1622
Unseparated token sequence	11 969	15 711	12 927	14 753	10 289	15 571	12 205	15 475	10 040	17 640	11 393	16 287	11 740	15 940	12 660	15 020
Protocol name	982	531	995	518	988	525	985	528	990	523	990	523	986	527	983	530
Mac address	4	67	20	51	4	67	20	51	20	51	16	55	0	71	63	8
Non-standard mac address format (EUI-64)	1884	1884	1881	1887	2441	1326	1881	1887	1596	2172	1886	1882	2441	1327	1884	1884
ID Token	1039	72	1039	72	997	69	749	362	304	807	743	368	1017	94	1044	67
Boolean token	455	281	593	143	18	21	591	145	420	316	458	278	219	517	176	560
IPv4 token	3107	4759	3741	4125	2687	5179	3637	4229	2675	5191	3112	4754	3386	4480	2744	5122
IPv6 token	34	82	57	59	32	84	50	66	57	59	23	93	0	116	91	25
Domain name	1272	2198	1312	2158	1195	2066	1451	2019	1210	2260	1305	2165	1244	2226	1342	2128
Hexadecimal	917	698	891	724	626	903	584	1031	837	778	933	682	692	923	1132	483
Log highlighters	205	18	128	95	143	77	178	45	126	97	168	55	67	156	131	92
Token with punctuation	1186	2156	968	2374	745	2587	1140	2202	913	2429	1198	2144	582	2760	872	2470
Boolean in a format different from true/false	1241	3362	1351	3252	735	3506	1345	3258	1090	3513	1176	3427	982	3621	1506	3097

log events for the eight log parsers. The Log parsers exhibited varying numbers of parsing errors: Drain (851), ULP (923), IPLoM (882), AEL (914), LenMa (883), Spell (959), SHISO (963), and LogMine (965).

The data reveals that issues with delimiters and token boundaries are prevalent across all parsers. Characteristics like "Unseparated Token Sequence" and "Key-Value Pairs formed with a colon" frequently result in parsing inaccuracies due to inconsistent token delimitations. For example, "Unseparated Token Sequence" leads to over 15,000 incorrect parses across most parsers, with LogMine showing 17,640 incorrect versus just 10,040 correct parses.

Structural complexity is also a major challenge, with LECs like "Multi-Level Nested Tokens," "Tokens with Punctuation," and "Enclosed Quotations" often leading to inaccuracies. "Multi-Level Nested Tokens" show nearly zero correct parses for parsers such as IPLOM and LogMine. Similarly, tokens with punctuation tend to confuse parsers that rely on consistent patterns.

Some parsers, like LenMa and ULP, tend to perform slightly better with specific LECs, such as "UUIDs" or "Multi-level nested tokens", compared to others like Drain and LogMine. For example, LenMa achieves 357 correct parses for "UUIDs" compared to IPLOM's 79. The analysis highlights that no single parser consistently outperforms others across all LECs, indicating the potential benefits of hybrid parsing approaches that combine the strengths of multiple techniques.

Further analysis of the impact of LECs [4] on parsing errors shows that four characteristics—"Unseparated Token Sequence" (19%), "Alphanumeric and Special Characters" (12%), "Key-Value Pair with a Colon" (11%), and "Decimal" (8%)—contribute to over 50% of the parsing errors. Meanwhile, characteristics like "MAC Address," "Multi-Level Nested Tokens," "URLs," and "Log Highlighter" have a consistently low impact on errors.

We discuss in more detail each category and its LECs alongside the impact on the parsers in the following subsections.

[4]The impact is calculated using the formula:.

$$RIM_p = \frac{INC_p}{T_p} \tag{17}$$

Where RIM_p represents the relative impact of a specific characteristic for a given log parser p. INC_p denotes the *number of occurrences* of the characteristic in the incorrectly parsed log events for parser p, and T_p represents the *total number of occurrences* of all characteristics in the incorrectly parsed log events for parser p within all the datasets.

Table 5.2: Characteristics under Log Event Presentation

Characteristic	Example	Challenge
Log events with only static tokens	`The server is now running.`	Identifying the lack of dynamic content
Alphanumeric and special characters	`[APP-1234] ... username = johndoe_123!`	Parsing mixed alphanumeric values with special characters, which can lead to misinterpretation of tokens.
Tokens with Punctuation	`can't`	Correctly handling words with embedded punctuation, avoiding split or misinterpretation
Log highlighter	`##### Transaction Canceled! #####`	Recognizing repeated special characters and distinguishing them from the core content

Log Event Presentation:

This category focuses on characteristics that affect the visual presentation of log events, such as multiline logs or special highlighters like '#### ERROR ####'. These features can obstruct tokenization or make it difficult to distinguish key parts of a log, affecting the parser's ability to interpret it correctly. While conventional log formats are well-understood and parsers can reliably handle them, unconventional formats—featuring unusual spacing, special characters, or non-standard highlighting—introduce variations that parsers may not be trained to handle, leading to misinterpretation or parsing failures. Table 5.2 summarizes the LECs in this category. "Log events with only static tokens", like "The server is now running," challenge parsers by lacking dynamic elements, which can lead to their significance being overlooked.

Tokens with "alphanumeric and special characters", such as "[APP-1234] ... username = johndoe_123!", present challenges in distinguishing control characters and interpreting token boundaries. Additionally, common punctuation in words like "can't" requires careful handling to avoid misclassification.

"Log highlighters", like "##### Transaction Canceled! #####", add complexity by requiring parsers to differentiate emphasis markers from core log content, ensuring proper extraction and analysis.

Data Type:

This category focuses on characteristics representing specific data types in logs, such as timestamps, IP addresses, numerical values, or Booleans. These data types often vary in format, such as timestamps differing due to regional settings or Boolean values being represented as ”true/false,” ”1/0,” or ”yes/no.” Unlike *Log Event Presentation*, which deals with visual aspects, Data Type Representation addresses the correct identification and interpretation of underlying data types.

Variations and inconsistencies in data types can create parsing challenges. For example, a parser might misread an unconventional timestamp format or fail to recognize Boolean values like ”Y/N.” Log parsers that use regular expressions face particular difficulties here—matching all possible formats for data types like timestamps or IP addresses is often impractical, given the many variations. Regular expressions also lack context awareness, making it challenging to distinguish similar elements, such as a colon in a timestamp versus one used as a delimiter. These challenges are especially pronounced in logs with inconsistent or unconventional formats.

As shown in Table 5.3, this category has the most LECs (19 out of 30). ”UUIDs” and ”IDs” can vary significantly in length, format, and delimiters, making their recognition and extraction challenging across different contexts. Datetime tokens differ based on regional settings or logging configurations, involving multiple formats, inconsistent use of separators, and handling time zones. Representations of time durations involve various units, requiring special handling of singular vs. plural forms and abbreviations.

”Decimal” numbers introduce complexity due to differences in representation, such as ”3.14” vs. ”3,14,” depending on locale. Properly distinguishing numeric values from other tokens or separators is critical for accurate parsing. Similarly, ”data volumes” can be represented using different units or abbreviations, such as ”KB” versus ”Kilobytes,” requiring normalization for consistency.

”Protocol names” like ”HTTP” or ”FTP” may appear in different contexts within logs or combined with other tokens, necessitating disambiguation. ”MAC addresses”, including non-standard formats, add complexity due to the use of varying delimiters, such as colons or hyphens, requiring parsers to handle these variations consistently. ”Boolean values” can be represented in multiple forms, including ”True/False,” ”1/0,” and ”Yes/No,” which introduces ambiguity during parsing.

Table 5.3: Characteristics under Data Type

Characteristic	Example	Challenge
UUIDs	`session ID 123456789`	Variability in length, format, and delimiters, leading to inconsistencies in identification and extraction
Datetime tokens	`2022-03-15 13:45:32+00:00`	Handling multiple formats, inconsistent use of separators, and correctly accounting for time zones.
Time duration	`2 hours 30 mins`	Parsing different time units, distinguishing between singular/plural, and inconsistencies in abbreviations or units.
Decimal	`3.14`	Differentiating between numeric values and potential delimiters, managing locale-specific formats (e.g., commas vs. periods)
Data volumes and unit	`10KB, 10 Kilobytes`	Handling unit variations and abbreviations.
Protocol name	`HTTP, FTP`	Disambiguating protocol names that may be used in different contexts or combined with other tokens.
MAC addresses	`00-11-22-33-44-55`	Handling various delimiters (e.g., colon, hyphen)
Non-standard MAC address format (EUI-64)	`3B-A7-94-FF-FE-07-CB-D0, or 3BA7:94FF:FE07:CBD0`	Various delimiters (hyphen, colon)
Boolean token	`True, False, T, F`	Multiple representations
Boolean in a format different from true/-false	`True,true,0, 1, Yes`	Parsing multiple representations (e.g., True/False, 0/1, Yes/No)
IPv4 Addresses	`192.168.0.1:8080`	Handling addresses with port numbers, distinguishing different segments, and avoiding confusion with unrelated numbers.
IPv6 Addresses	`2001:0db8:85a3:0000:0000:8a2e:0370:1234`	Expanded and contracted forms
Domain names	`example.com, www.example.com`	Parsing different structures, including subdomains, and managing variations in domain components.
Hexadecimal	`0x3F8A`	Identifying hexadecimal values with or without a prefix, and distinguishing them from other alphanumeric tokens.
URLs	`https://www.hdfs.com`	Parsing base URL as one unit, confusion due to the separator in the URL string
URLs with query parameters	`https://www.hdfs.com/search ?id=1&type=a`	Parsing base URL as one unit, confusion due to the separator in the URL string
Folder structure	`/home/user/docs`	treating "/" as a non separator
Nouns	`User Cyprus`	Differentiating static tokens from dynamic ones, particularly when common nouns or place names
ID	`Block blk_+1234, blk_ 1234, blk_-1234`	Managing inconsistent delimiters and variations in prefixes

”IPv4” and ”IPv6” addresses each present distinct challenges. IPv4 addresses may include port numbers, and both address types have variable formats, increasing the difficulty of accurately identifying and parsing them. ”Domain names” can also vary, often including subdomains or different structures that need to be managed effectively.

”Hexadecimal” values may include or omit the ”0x” prefix, making identification within a log context difficult. ”URLs” and ”URLs with query parameters” involve multiple components, such as base URLs, paths, and query parameters, which must be parsed correctly to prevent errors due to misuse of separators. ”Folder structures” represented with slashes require careful parsing to ensure that ”/” is interpreted correctly within the context, maintaining the intended hierarchy.

Identifying whether a ”Noun” is a static token or dynamic content presents particular challenges, especially when dealing with common nouns, place names, or ambiguous words. Finally, ”IDs” with inconsistent formats, prefixes, or delimiters, such as ”blk_+1234” or ”blk_-1234,” present difficulties in maintaining consistent identification and parsing, which may lead to inaccuracies.

Structural Arrangement of Tokens:

This category examines how the arrangement of token sequences within log events affects parsing. Unlike focusing on individual tokens, this category considers how multiple tokens are combined using punctuation marks and delimiters to form complex structures like key-value pairs, units, or nested messages. This involves understanding token relationships rather than treating them in isolation.

Unlike *Log Event Presentation*, which deals with visual formatting, this category focuses on the syntactic organization and relationships between tokens, such as delimiters (commas, colons, equals signs), nesting (brackets, parentheses), and token sequences that define the log event. For example, colons may separate keys, or information might be nested in brackets (”Error in [ModuleName] at (Timestamp)”). Table 5.4 summarizes different structural arrangements of log tokens and their associated parsing challenges. ”Single-level” and ”multi-level nested tokens” present difficulties due to unclear boundaries and nested levels that complicate extraction. ”Key-value pairs” using inconsistent separators (’=’ or ’:’) increase parsing complexity, as these separators may vary or be inconsistently used.

Table 5.4: Characteristics under Structural Arrangement of Tokens

Characteristic	Example	Challenge for Log Parsing
Single-level nested tokens	`Error: (404) Not Found`	Difficulty in extracting values from a single-level nested structure, where the boundaries may be ambiguous.
Multi-level nested tokens	`Request received: [2012-04-23T18:25:43.511Z (sasl_plain)]`	Complex token extraction due to multiple levels of nesting.
Key-value pairs (colon or equal)	`User name=JohnDoe, action:login`	Difficulty in recognizing key-value pairs due to inconsistent use of separators (colon or equal) and inconsistent spacing.
Word-number pairs	`User John Doe ... session ID 12345`	Identifying relationships between words and numbers when no clear separator is present, making it challenging to distinguish static words from dynamic numerical values.
Enclosed quotations	`User 'JohnDoe' ... session ID '123456789'`	Extracting values enclosed within quotations while differentiating quoted content from other tokens in the log.
Unseparated Token Sequence	`Error code:500Error message: ... ID123456`	Identifying boundaries between tokens in sequences without clear separators, leading to difficulties in tokenizing and understanding log content.

"Word-number pairs" without explicit delimiters challenge parsers to correctly associate related tokens, while "Enclosed quotations" require accurately extracting values and distinguishing them from other tokens. "Unseparated token sequences" create significant complexity by lacking clear boundaries, making it hard for log parsers to tokenize correctly. These challenges emphasize the need for sophisticated strategies to parse diverse and structurally complex logs effectively.

Statistical Association between Log Characteristics and Parsing Errors

To explore the relationship between log characteristics and parsing errors, we conducted a Chi-square Test of Independence. This test is particularly well-suited for analyzing associations between categorical variables, allowing us to determine whether specific LECs are significantly associated with parsing errors. In this context, the null hypothesis asserts that there is no association between the presence of a log characteristic and parsing errors. A p-value less than the significance level (commonly set at $\alpha = 0.05$) indicates strong evidence against the null hypothesis, suggesting that the association is statistically significant. A majority of the log characteristics exhibit extremely low p-values (in many cases equal to zero or very close to zero), which are significantly below the

threshold of $\alpha = 0.05$.

The Chi-square analysis highlights that Characteristics such as "Log Events with Only Static Tokens," "Alphanumeric and Special Characters," and "Word-Number Pairs" exhibit a substantial impact on parsing errors.

In contrast, characteristics like "IPv4 Tokens" and "IPv6 Tokens" demonstrate relatively minor effects, with effect sizes under 10. Specifically, "IPv6 Tokens" show an effect size of just over 2, with a p-value of 0.15, indicating minimal influence on parsing errors.

Overall, the results provide statistical evidence that the LECs affect Log Parsing performance.

Discussion and Implications: The results hints to key implications for improving log parsing techniques. First, "Unseparated Token Sequence" and "Alphanumeric and Special Characters" are consistently challenging across all parsers, suggesting that new approaches should prioritize methods to effectively handle token boundary detection and mixed character types.

The difficulties with "Key-value pairs" formed using varied separators further emphasize the need for parsing methods that adapt to different conventions. Parsers should be designed to interpret separator patterns in real-time, rather than relying on predefined rules. Incorporating dynamic, probabilistic models or token prediction capabilities would allow parsers to flexibly adjust to diverse key-value structures.

From a log writing perspective, the findings underscore the importance of consistent and predictable log formatting. Standardizing delimiters, timestamp formats, and minimizing ambiguous characters can significantly enhance the effectiveness of log parsers. Developing best practices for log generation—focused on uniformity and simplicity—would reduce parsing complexities and improve downstream analysis accuracy.

5.3.2 RQ2: Evaluating Algorithmic Adaptability to Diverse LECs

Analyzing different parsing outcomes revealed that many Log parsers struggle with diverse LECs, leading to frequent parsing errors. These errors typically stem from rigid assumptions about log structures and token positions, which fail to accommodate the variability inherent in different LECs.

Parsers based on fixed token positions tend to misinterpret logs when LECs deviate from anticipated structures, whether through missing elements or altered token order. For example, a parser trained on the format "[Timestamp] User [Username] performed [Action]" may misidentify "performed" as a username if it encounters "[2023-09-17 10:16:00] User performed logout." This issue is especially pronounced in parsers like Drain, Spell, and LogSig, each of which assumes a static log structure. Specifically, Drain uses fixed token positions, Spell depends on token sequencing, and LogSig interprets rare tokens as dynamic, resulting in parsing errors when these assumptions are unmet.

Many log parsers also use regular expressions to enhance tokenization often fails with LECs involving inconsistent formats. Parsers utilizing domain-specific knowledge can accurately interpret LECs for specific systems but struggle in unfamiliar contexts where LECs vary widely. Token counting and delimiter-based tokenization, while useful for uniform logs, often cannot handle variations in token counts or unexpected delimiters. Machine learning algorithms offer flexibility for dynamic formats but often sacrifice precision for simpler, structured logs.

The subsequent sections explore the strengths and shortcomings of log parsers when handling diverse LECs, with a focus on how the design choices behind these tools impact the tool adaptability to different log formats.

Clustering-Based Log Parsers

Clustering-based log parsers group log events with similar characteristics using methods like hierarchical and density-based clustering. These techniques help identify patterns and simplify parsing, but the choice of clustering method affects parsing accuracy, especially for complex LECs.

LogMine [29] uses hierarchical clustering and performs well for structured logs but struggles with complex LECs like "Multi-level Nested Tokens", resulting in significant parsing errors (3,309 incorrect parses vs. 453 correct ones). The hierarchical method relies on predefined structures, which can be disrupted by complex nested tokens, leading to misclassification.

LenMa [24] uses density-based clustering and generally outperforms LogMine but still faces challenges with LECs like Equals-Separated Key-Value Pairs and Hexadecimal values. Density-based clustering, such as DBSCAN, is effective for handling variability and noise without needing

Table 5.5: Comparison of Clustering-Based, Heuristic-Based, and Frequent Pattern Mining Log Parsers

Parser Type	**Strengths**	**Weaknesses**	**Parsers**
Clustering-Based	• Handles variability relatively well. • Effective for grouping logs with similar characteristics.	• Struggles with complex nested structures. • Lack inherent mechanisms for token boundary detection.	LogMine, LenMa
Heuristic-Based	• Hierarchical parsing tree for identifying token boundaries (e.g., Drain). • Better with specific LECs through tailored heuristics.	• Struggles with deeply nested tokens. • Limited adaptability to dynamic log variations.	Drain, Spell, Shiso, AEL, IPLOM
Frequent Pattern Mining	• Effective for structured logs with consistent patterns. • Automates parsing through frequency analysis of static vs. dynamic tokens.	• Struggles with infrequent or rare tokens. • Poor handling of hierarchical or complex nested tokens.	ULP

a predefined number of clusters. However, it struggles with structured LECs because tokens like ”key-value pairs” or ”hexadecimals” are often sparse or inconsistently distributed, making cohesive clustering difficult. Additionally, density-based clustering lacks an inherent mechanism for recognizing token boundaries, which contributes to parsing errors in structured log elements.

Heuristic approaches

Heuristic log parsers, including Drain, Shiso, Spell, AEL, and IPLOM, use distinct methods to identify patterns in log data, each with strengths and limitations in handling various LECs.

We focus on comparing other heuristic log parsers to Drain, as it consistently shows the highest log parsing performance to Log parsing surveys [146].

Drain uses a parsing tree to organize log tokens hierarchically, performing better at parsing ”Unseparated token sequences” by effectively inferring boundaries. However, it struggles with ”Single-level nested tokens” (where it has the worst performance comparatively) due to the complexity that disrupts its hierarchical model, making it less suitable for deeply nested structures.

Spell relies on the Longest Common Subsequence (LCS) method, which helps it handle longer log entries like ”date volumes”, ”time durations” and ”Single-level nested tokens” relatively better than Drain. The LCS algorithm works well for consistently structured log entries by matching recurring sequences of tokens. This allows Spell to recognize and parse recurring patterns effectively, even in slightly variable contexts.

Shiso also uses a parsing tree like Drain but adds token-wise Euclidean distance to classify log events. This distance-based classification gives Shiso an edge over Drain in dealing with subtle variations between tokens. As a result, Shiso performs better with LECs such as ”Log Highlighters,” ”Tokens with Punctuation,” and ”Hexadecimals.”

AEL uses heuristics to replace dynamic tokens with placeholders, grouping similar log events based on token consistency. It Performs better than Drain at parsing ”Single-level” and ”Multiple-level nested tokens”.

IPLOM partitions log entries iteratively based on structural attributes like token count and position. It struggles with highly variable token structures, particularly ”Key-value pairs formed with a

colon", "Decimals" or logs with inconsistent formats. The reliance on token frequency and fixed positions limits IPLOM's adaptability to dynamic, unpredictable log entries. it is good at IDs because of the repetition

Frequent Pattern Mining Log Parsers

Frequent pattern mining techniques, like ULP, identify common sequences in log data by analyzing the frequency of patterns. ULP automates log parsing by grouping similar events and using frequency analysis to differentiate between static and dynamic tokens. This approach works well for structured logs, particularly those with consistent formats such as "Key-Value Pairs" and "Unseparated Token Sequences", where it can accurately distinguish between static and dynamic components.

However, ULP's reliance on local frequency can lead to challenges. Infrequent tokens, such as those in "Boolean Tokens with Non-Standard Formats" or "Alphanumeric Sequences with Special Characters", may be misclassified due to their lower repetition.

Discussion and Implications: The analysis demonstrates that parsers relying on rigid token positions, delimiters, regular expressions (regex), frequency-based assumptions, domain-specific knowledge, or predefined structures are particularly vulnerable to errors when Log Event Characteristics (LECs) vary. A core implication is the need for flexible parsing mechanisms that can accommodate the diversity of LECs without rigid assumptions. Future parsers must dynamically adapt to variations in log structures. Table 5.5 highlights the potential of hybrid approaches that leverage the strengths of each log parsing method—combining clustering-based, heuristic-based, and frequent pattern mining techniques. A hybrid approach could effectively address the limitations of individual parsers by dynamically selecting the best parsing strategy based on the LECs present in a dataset.

5.3.3 RQ3: Analysis of LECs across Systems

Table 5.6 shows the percentage of log parsing errors for all log parsers combined for a given dataset. The analysis of mismatch percentages across different log event characteristics (LECs) for various datasets reveals key insights into parsing difficulties. The mismatch percentages reflect the

Table 5.6: Percentage of Log parsing Errors for a specific LEC to the total of the same LEC found for a given dataset)(NA MEANS THE LEC IS NOT PRESENT)

LEC	BGL	HealthApp	Windows	Android	Hadoop	Zookeeper	OpenSSH	Spark
Alphanumeric and special characters	94.63%	65.61%	73.88%	66.23%	76.29%	86.62%	87.64%	99.98%
Log event with only static	21.73%	0.69%	NA	0.00%	0.00%	1.03%	0.00%	51.56%
UUID	NA	NA	20.59%	NA	NA	NA	100.00%	100.00%
Single-level nested tokens	76.74%	87.50%	79.46%	47.36%	100.00%	77.43%	71.12%	15.18%
Multi-level nested tokens	100.00%	NA	NA	70.82%	85.71%	NA	NA	100.00%
Enclosed quotations	NA	NA	39.58%	68.08%	28.57%	NA	100.00%	99.57%
URL with Query parameters	NA	NA	NA	NA	NA	NA	NA	NA
Equals-separated key-value Pairs	96.62%	55.09%	13.73%	43.30%	79.33%	44.14%	90.41%	19.06%
Key-value pairs formed	68.01%	83.70%	70.31%	49.24%	64.93%	28.27%	81.70%	39.70%
Word-number pair	50.38%	18.53%	86.41%	47.77%	86.58%	43.96%	63.80%	34.13%
Folder structure	49.29%	NA	50.42%	81.98%	25.55%	10.72%	100.00%	99.82%
Datetime tokens	51.79%	38.19%	84.21%	48.33%	88.32%	41.09%	49.42%	29.58%
Time duration	86.11%	NA	87.50%	100.00%	100.00%	NA	0.00%	0.38%
URL	NA	NA	NA	100.00%	100.00%	NA	NA	NA
Decimal	65.57%	100.00%	75.41%	90.09%	51.18%	31.39%	72.78%	48.62%
Data volume and unit	43.27%	NA	75.00%	NA	NA	NA	33.33%	43.61%
Nouns	87.50%	62.50%	100.00%	47.31%	32.99%	93.55%	99.04%	99.15%
Unseparated token sequence	81.19%	66.76%	56.45%	42.83%	66.68%	34.15%	66.78%	50.67%
Protocol name	100.00%	NA	NA	0.00%	14.29%	100.00%	99.75%	66.67%
Mac Address	96.25%	NA	NA	NA	NA	NA	NA	NA
Non-standard mac address	100.00%	NA	74.98%	28.57%	NA	NA	NA	100.00%
ID Token	100.00%	NA	NA	35.76%	2.30%	25.64%	0.17%	14.43%
Boolean token	NA	51.39%	33.33%	42.18%	57.14%	NA	NA	NA
IPv4 token	66.32%	NA	75.27%	100.00%	26.79%	31.24%	76.05%	99.66%
IPv6 token	96.25%	NA	NA	NA	NA	NA	NA	NA
Domain name	62.50%	82.16%	NA	63.15%	42.53%	34.11%	10.39%	100.00%
Hexadecimal	43.75%	87.50%	13.31%	57.19%	73.61%	54.63%	33.33%	100.00%
Log highlighter	NA	NA	NA	57.14%	0.00%	0.00%	NA	100.00%
Token with punctuation	36.42%	NA	46.05%	91.88%	44.53%	12.56%	8.04%	100.00%
Boolean in a format different from true/false	67.91%	53.57%	34.62%	33.51%	92.23%	41.84%	49.81%	92.66%
Total	73.33%	62.44%	59.59%	50.27%	64.54%	31.84%	72.65%	46.49%

LEC	Linux	Mac	Openstack	HDFS	Apache	Proxifier	HPC	Thunderbird
Alphanumeric and special characters	96.68%	80.67%	89.30%	37.29%	41.71%	49.93%	95.30%	77.22%
Log event with only static	50.67%	54.72%	NA	NA	NA	0.00%	14.47%	15.44%
UUID	100.00%	71.13%	76.51%	NA	NA	NA	NA	68.23%
Single-level nested tokens	72.17%	77.67%	69.30%	14.29%	100.00%	100.00%	69.73%	87.55%
Multi-level nested tokens	100.00%	96.00%	NA	NA	NA	NA	NA	100.00%
Enclosed quotations	69.06%	74.32%	99.90%	NA	NA	NA	40.00%	91.36%
URL with Query parameters	100.00%	100.00%	NA	NA	NA	NA	NA	100.00%
Equals-separated key-value Pairs	98.38%	82.20%	100.00%	NA	NA	3.88%	17.68%	50.21%
Key-value pairs formed	96.33%	74.00%	80.68%	4.39%	49.65%	50.21%	91.95%	61.44%
Word-number pair	98.52%	81.00%	36.68%	1.75%	10.09%	97.76%	75.99%	42.54%
Folder structure	88.04%	89.21%	89.55%	20.19%	100.00%	0.00%	64.00%	63.24%
Datetime tokens	98.81%	83.30%	85.56%	7.51%	25.29%	98.05%	58.85%	64.94%
Time duration	97.04%	92.80%	68.77%	NA	100.00%	100.00%	100.00%	89.83%
URL	100.00%	100.00%	NA	NA	NA	NA	NA	100.00%
Decimal	97.39%	79.72%	82.88%	3.74%	98.32%	97.80%	100.00%	52.33%
Data volume and unit	98.69%	55.96%	69.00%	NA	100.00%	100.00%	100.00%	77.88%
Nouns	96.57%	78.16%	52.28%	4.76%	98.49%	97.19%	68.17%	85.64%
Unseparated token sequence	90.71%	68.51%	79.33%	11.64%	33.03%	49.68%	60.35%	60.31%
Protocol name	62.16%	49.62%	0.00%	NA	0.00%	0.00%	22.22%	36.36%
Mac Address	80.95%	31.97%	NA	NA	NA	NA	100.00%	87.46%
Non-standard mac address	90.14%	99.86%	81.23%	8.00%	NA	NA	100.00%	36.15%
ID Token	90.74%	85.45%	NA	2%	NA	100.00%	52.94%	8.80%
Boolean token	85.71%	86.00%	100.00%	NA	NA	NA	NA	53.85%
IPv4 token	98.40%	94.26%	84.67%	3.74%	94.38%	64.18%	100.00%	49.34%
IPv6 token	51.52%	41.77%	NA	NA	NA	NA	100.00%	81.82%
Domain name	99.02%	90.67%	89.95%	NA	49.24%	49.16%	33.33%	63.22%
Hexadecimal	57.19%	46.94%	87.03%	NA	NA	100.00%	97.96%	62.34%
Log highlighter	30.77%	32.76%	NA	NA	NA	0.00%	61.53%	25.26%
Token with punctuation	96.38%	83.34%	82.97%	98.53%	33.03%	59.53%	98.41%	76.62%
Boolean in a format different from true/false	82.23%	77.83%	89.44%	14.29%	64.52%	53.34%	60.51%	86.17%
Total	93.81%	76.80%	83.13%	9.85%	38.80%	59.53%	61.50%	62.28%

inherent complexity and variability of logs across different datasets. High mismatch rates are observed across many datasets for LECs like ”Alphanumeric and special characters” (e.g., 96.68% for Linux and 99.98% for Spark) and ”Multi-level nested tokens”, which often reach 100%, indicating significant parsing challenges. In contrast, HDFS, Zookeeper and Apache exhibit notably lower total mismatch rates (9.85%, 31.84% and 38.80%, respectively), suggesting more structured and consistent log formats that are easier to parse. The total mismatch rate highlights Linux (93.81%), BGL (73.33%), and OpenStack (83.13%) as among the most challenging datasets. Overall, LECs such as ”Alphanumeric and special characters”, ”nested tokens”, ”UUIDs”, and ”Key-value pairs” significantly impact all log parsers effectiveness.

Logs that rely heavily on descriptive ”Nouns” to convey operations or system states, as seen in OpenStack, tend to introduce ambiguity. This makes it difficult for parsers to accurately group or categorize log entries, especially when these nouns are context-dependent and vary widely in usage. Linux (93.81%) Openstack, and Mac (76.80%) logs show high mismatch rates, likely due to the dynamic nature of these systems, which often include variable-length data fields, dynamically generated tokens, or more unstructured content. These logs include a broader range of services and processes, contributing to the complexity. In contrast, HDFS (9.85%) demonstrates a much lower mismatch rate, indicating that its logs are more standardized and structured, likely due to the uniformity of distributed file system operations. OpenSSH (72.65%) also exhibits high mismatch rates, suggesting significant dynamic content or variability, due to complex cloud infrastructure and security authentication mechanisms.

The analysis comparing the LEC presence in HDFS, Zookeeper and Apache logs provides insights into why these datasets achieved higher match scores and experienced fewer parsing challenges compared to others like Linux or Openstack. These logs share similarities in the predictability and consistency of their log elements, contributing to their higher parsing success rates.

First, many complex LECs are absent (those shown as ”NA” in Table 5.6), such as ”Multi-level nested tokens”, ”Enclosed quotations”, and ”URLs with query parameters”. The absence of such complex elements reduces structural variability, making it easier for parsers to identify consistent patterns. For example, HDFS logs do not include ”Domain names”, ”Protocol names”, or ”Data volume and units”, whereas these are present in Apache.

Linux logs exhibit high complexity due to the presence of nearly all LECs, which makes them more challenging to parse consistently. LECs like ”Multi-level nested tokens”, varied ”UUIDs”, and diverse usage of ”IP addresses” create significant parsing challenges, leading to high mismatch percentages.

The difference in mismatch rates for UUIDs between HDFS and Linux highlights how the consistency and context of log usage significantly influence parsing success. In HDFS, IDs follow a simple and well-known pattern (blk_38865049064139660), making them predictable and easier for parsers to handle often managed with simple regular expressions. The consistent format of these IDs reduces complexity, allowing parsers to accurately identify and process them with minimal difficulty. Conversely, in Linux, UUIDs and similar unique identifiers frequently appear in varied and complex contexts, often embedded within more intricate structures. The diverse usage of these identifiers across different contexts—such as process IDs, session identifiers —complicates the parsing process. This variability contributes to the high mismatch rate, demonstrating that standardized tokens can still present challenges when their usage is not uniform. Additionally, Linux log entries often contain IP addresses in both IPv4 and more complex formats (e.g., `rhost=220-135-151-1.hinet-ip.hinet.net`), further adding to parsing difficulties. The dynamic nature of these addresses, especially when combined with other LECs, contributes to high mismatch rates for both IPv4 and domain name tokens.

Discussion and Implications: The analysis of LECs across different systems highlights the direct impact that the presence, absence, and complexity of specific log event characteristics have on log parsing performance. Systems with more standardized log structures, such as HDFS, which lack many complex LECs, demonstrate significantly fewer parsing errors. The presence or absence of specific Log Event Characteristics (LECs) allows us to classify datasets by their parsing complexity. Datasets like HDFS, Zookeeper and Apache lack many of the more complex LECs, such as ”nested tokens”, ”UUIDs”, and ”Log highlighter”s. This absence of complexity enables parsers to form simpler log patterns, leading to more efficient parsing with fewer mismatches. In contrast, datasets like Linux feature all LECs, including ”Multi-level nesting”, UUIDs, and punctuation tokens.

The notion of log patterns emerges as a central aspect of this analysis. Log patterns are essentially formed from the recurring combinations and sequences of LECs within log entries. Systems

like HDFS and Apache, with simpler combinations of LECs, and consistency for UUIDs inherently form simpler and more predictable log patterns, which makes parsing easier. Given the significantly lower parsing mismatch rates in HDFS and Apache, these datasets provide a model for creating logging standards that could help reduce parsing difficulties across other systems. The structured nature of their logs, combined with a limited number of complex LECs, serves as a potential blueprint for designing log formats that balance informative detail with parsing ease.

5.3.4 Threats to Validity

The selection of log datasets and parsing tools presents a potential threat to internal validity. To mitigate this, we used 16 log datasets from the widely recognized LogPai repository, which are commonly employed in similar research studies. As for the parsing tools, we selected eight high performing log parsing tools, ensuring that they cover a range of methods. We believe that these tools represent a comprehensive set of log parsing tools. Another threat to validity is the identification of LECs. Due to the complexity of some log datasets, we may have missed certain LECs. To address this, we applied the open coding research method and we carefully analyzed and reviewed log errors to ensure thorough identification of LECs. A threat to conclusion validity exists regarding the correctness of the results obtained. We classified the identified LECs following a detailed and systematic process to ensure accurate classification. Moreover, there is a threat to reliability validity concerning the replicability of this study. To mitigate this threat, we have made all the results available online to facilitate the reproduction of our study's findings.

Finally, the generalizability of our results may also pose a threat to validity. Our study was conducted on 16 log datasets using eight parsing tools, and we do not claim that our findings can be generalized to all types of logs, particularly to industrial or proprietary log datasets, to which we did not have access.

5.4 Conclusion

In this study, we identified 30 distinct LECs that we grouped into three categories using the open coding method, and by analyzing over 32,000 log events from 16 different systems. LECs represent

the diverse structural elements present in log entries, ranging from token sequences and data types to structural arrangements. Understanding LECs is crucial because they directly influence the success of log parsing techniques, especially as modern systems generate increasingly heterogeneous log data. Our findings show that the assumptions made by parsing algorithms—such as fixed token positions- often struggle to accommodate the diversity of LECs, leading to parsing inaccuracies. The analysis of mismatch rates across datasets reveals notable disparities in parsing difficulty. Datasets like Linux and OpenStack demonstrate high mismatch rates due to their structural complexity and variability, while HDFS and Apache logs exhibit lower mismatch rates, suggesting more consistent, structured formats.

Future work should focus on advancing log parsing tools to intelligently recognize and adapt to complex log patterns. Hybrid approaches that combine heuristic and machine learning techniques could leverage domain-specific knowledge alongside data-driven adaptability. Insights from systems with lower parsing mismatch rates, like HDFS and Apache, could also inform the creation of standardized logging practices, ultimately contributing to the development of more resilient and effective log parsing tools.

Chapter 6

Application: Log privacy

From Log Taxonomy to Privacy Considerations: A clear understanding of the structural and contextual properties of logs, gained through the log taxonomy, offers not only a roadmap for more adaptable parsers but also a vantage point for identifying where sensitive data may arise. In many real-world systems, a single log entry can inadvertently contain personally identifiable information (PII), such as usernames or IP addresses, particularly when the log format is inconsistent or ambiguously defined. By highlighting these structural "trouble spots," the taxonomy reveals how privacy risks can emerge from seemingly benign logging practices. Consequently, the final chapter turns to privacy considerations, presenting guidelines and methods to ensure that sensitive data is handled responsibly. In doing so, it closes the loop between understanding log variability and applying that knowledge to meet both operational and regulatory demands in AIOps.

I. Sedki. "A Preliminary Study on the Privacy Concerns of Using IP Addresses in Log Data". In Proc. of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE 2024 Ideas, Visions and Reflections), New York, NY, USA, pp. 527–531. doi: https://doi.org/10.1145/3663529.3663791

Abstract: Log data, crucial for system monitoring and debugging, inherently contains information that may conflict with privacy safeguards. This study addresses the delicate interplay between log utility and the protection of sensitive data, with a focus on how IP addresses are recorded. We scrutinize the logging practices against the privacy policies of Linux, OpenSSH, and MacOS, uncovering discrepancies that hint at broader privacy concerns. Our methodology, anchored in privacy benchmarks like GDPR, evaluates both open-source and commercial systems, revealing that the former may lack rigorous privacy controls. The research finds that the actual logging of IP addresses often deviates from policy statements, especially in open-source systems. By systematically contrasting stated policies with practical application, our study identifies privacy risks and advocates for policy reform. We call for improved privacy governance in open-source software and a reformation of privacy policies to ensure they reflect actual practices, enhancing transparency and data protection within log management.

6.1 Introduction

Logs are essential in software systems, serving as detailed records of events or processes that provide critical insights for a range of tasks, from debugging [3, 43, 4] and failure prediction [46] to operational intelligence [127, 1, 2]. At the heart of these logs are structures known as log templates. A log template is a predefined format for log messages, outlining a consistent structure while allowing for variable data specific to each event. For example, a log template designed to record authentication failures might look like this: `Authentication failed from [IP Address] at [Timestamp].` In this template, `IP Address` and `Timestamp` are placeholders. These placeholders are filled with actual data during runtime — the IP address from where the authentication attempt was made and the specific time of the attempt. While the overall format of the log message

remains the same across different instances, the dynamic content (IP addresses and timestamps) changes, reflecting the specifics of each individual event.

Despite their utility, logs frequently capture personal data, including user IDs, email addresses, IP and MAC addresses. This inadvertent capture during routine operations raises privacy concerns, especially when logs are shared externally [82].

The concept of personal data in privacy law is broadly defined to include any information that can identify an individual, either directly or indirectly [83, 84].The General Data Protection Regulation (GDPR), defines personal data as `any information relating to an identified or identifiable natural person` (Article 4, GDPR). This definition supports the scope of our analysis, ensuring consistency and compliance with widely recognized standards [85]. This definition was highlighted in the landmark Breyer case, where IP addresses were recognized as personal data when linked with other identifiable information [93]. This ruling clarifies the privacy considerations surrounding IP addresses in logs, affirming their status as personal data. User perceptions of IP addresses as highly sensitive are documented in studies [94], with concerns exacerbated by prevalent IP tracking practices [95, 96]. The treatment of IP addresses in logs, thus, becomes a critical aspect of maintaining user privacy.

Several studies have explored the security aspects of system logging and proposed guidelines for secure logging practices. Zeng et al. [97] discuss a range of security vulnerabilities that arise from system logging and emphasize the necessity for robust logging mechanisms to prevent data breaches and maintain system integrity.

In a more focused study, Patel et al. [98] delve into the specifics of logging within the Linux kernel suggesting improvements for effective logging that enhances both security and performance monitoring [98]. Lyons et al. [99] examine the extent of sensitive information logging within the Android ecosystem. Their study measures the prevalence of sensitive data in logs and discusses the implications for privacy and security, stressing the need for stringent controls on what is logged and how it is managed.

This study delves into the collection and amalgamation of IP addresses with other personal data in logs, juxtaposed against stated privacy policies. We focus on IP addresses due to their widespread use in digital communications and their debugging value, coupled with privacy implications when

associated with other data. In this study, we focus on dissecting the nuanced balance between the utility of log data and the imperative of safeguarding user privacy. This research is pivotal in highlighting the often-overlooked privacy implications in routine logging activities, especially in the context of evolving privacy regulations and increasing user sensitivity towards data privacy. Through our findings, we aspire to contribute to a more informed and responsible approach to logging practices, ensuring that they are not only effective but also respectful of user privacy rights.

6.2 Study Setup

The objective of this study is to assess the frequency and context in which IP addresses and other personal data are captured within logs across diverse systems and to evaluate the alignment of these practices with respective privacy policies. We examine logs from three widely-used systems—Linux, OpenSSH, and MacOS—chosen for their representation of different environments and their potential to capture a breadth of user activities.

6.2.1 Research Questions

We specifically aim to answer the following Research Questions.

RQ1: How do the privacy policies of Linux, OpenSSH, and MacOS compare in terms of their approaches to collecting and using IP addresses and other personal data?: This question will investigate the differences and similarities in how each system's privacy policy addresses data collection, with a focus on the types of data collected, the clarity of purpose for data collection.

RQ2: How do the privacy policies of these systems align with their actual logging practices, particularly in the context of IP address handling? : This question investigates the consistency between stated privacy policies and the actual data handling practices observed in the logs.

6.2.2 Dataset

The datasets used in this study consist of a subset of log files from the LogHub benchmark[77][1]. The benchmark contains log datasets that are generated from different types of systems. This benchmark is used extensively in log analysis research [136, 145, 30, 147, 77, 19]. Each log dataset of the LogHub benchmark comes with a subset of 2,000 log events that have been parsed manually. We selected three log datasets from this benchmark namely : Linux, OpenSSH and MacOS.

We included Linux Logs in our analysis due to their wide-ranging use in different contexts, including servers, desktops, and embedded systems. These logs are particularly rich in data, capturing a variety of system events, application behaviors, and user interactions.

OpenSSH Logs were selected for their relevance in both server management and personal computing. These logs provide detailed insights into secure operations, including authentication processes and connection details. The choice of OpenSSH logs is significant for exploring privacy concerns in environments where logs could be accessed by multiple applications within an organization.

Lastly, MacOS Logs were chosen to represent the personal computing environment. These logs provide a window into user interactions, system processes, and the behavior of various applications. The inclusion of MacOS logs is vital to understand how external breaches might lead to unauthorized access to personal data.

6.2.3 Methodology

We started by examining the most current privacy policies of Linux, OpenSSH, and MacOS to understand the official stance on data handling, with a focus on the handling of IP addresses, their combination with other personal identifiers, and the declared purposes for data collection. This initial step laid the groundwork for a comparative analysis against actual data logging practices.

We employed a comprehensive approach to identify personal data within log files, drawing inspiration from the General Data Protection Regulation (GDPR), one of the most stringent privacy laws globally. The GDPR's definitions of personal data are broadly applicable and align

[1] https://zenodo.org/record/3227177#.YUqmXtNPFRE

with the principles of various privacy regulations, making it an ideal benchmark for our analysis [85, 86, 162]. Our methodology involved initially focusing on the types of personal data explicitly mentioned in the privacy policies of Linux, OpenSSH, and MacOS, such as browser information, operating system details, device identifiers, domain names, and location data. Recognizing the GDPR's extensive scope, we further included a diverse array of data types like usernames, domain names, and unique device identifiers. The outcome of this step was a curated list of log events, each containing at least one IP address. We also noted the presence of combinations with other personal data.

Then, we parsed the log events containing IP addresses, extracting their underlying log templates. Once we isolated the log templates, we determined the intent behind the data collection for each template. This interpretive phase involved a close examination of the semantics embedded in the log messages. Our goal was to identify the specific purposes of logging these data points, whether it related to user authentication processes, system error tracking, or other operational activities as shown in Table 6.1.

Finally, we compared the extracted data from logs and the stated privacy policies. The primary focus of this comparison was twofold: first, to compare the data combinations involving IP addresses found within individual log events with the stated policy, and second, to scrutinize the alignment between the derived purposes from these log events and the purposes declared in the privacy policies. For instance, if a system's policy did not clearly articulate the collection of `IP addresses` and `domain name` for `user authentication` purposes, yet such a practice was evident in the logs, this was flagged as a discrepancy.

6.3 Results

Upon analysis, we found 1337 log events containing IP addresses for Linux, 1634 for OpenSSH, and 50 for MacOS. We identified 6 distinct log templates containing IP addresses within the Linux logs, 11 within the macOS logs, and 17 within the OpenSSH logs. Each template captures a unique log event type.(Table 6.3). Table 6.1 shows an exert of the log templates containing IP addresses and their inferred purposes.

Table 6.1: Purpose Mapping Examples

log template	Purpose
ANONYMOUS FTP LOGIN FROM <IP address>	Anonymous login
Authentication failed from <IP address>...: Permission denied in replay cache code	Authentication failure
Authentication failed from <IP address>...: Software caused connection abort	Authentication failure
connection from <IP address>() at <Time>	Connection origin and time

Table 6.2: Privacy Policy Comparison: Data and Purposes For Collection Across the 3 Systems

	Type of Data Collected	**Location Data**	**Purpose and Legal Basis**
Linux	IP Address, Domain Name, Browser	Country	Service enhancement, Debugging
OpenSSH	IP Address, Browser, OS, Device details	Approximate	Multiple purposes (User consent, Legitimate interests)
Mac Os	IP Address, Browser, OS, Device details	Precise	Power services, transactions, communication, security, law compliance (Consent, Contract, Legal compliance, Vital interests, Legitimate interests)

Table 6.3: Analysis of IP Address Occurrences and combinations in log events (LEs)

Dataset	**LEs with IP**	**LEs with IP and other Personal data**	**nb. log templates with IP address**
Linux	1337	633	6
OpenSSH	1734	1724	17
MacOS	50	15	11

Table 6.4: Analysis of IP Address and Associated Data Occurrences in Selected Log Datasets by Purpose of Collection.

Log dataset	Purpose for collection	IP address occurrences	Occurrences with Other data	What other data
Linux	Anonymous login	2	0	-
	Authentication failure	356	272	username
	Connection origin and time	979	361	username
MAC	Network client connection	10	10	username
	Client action	5	5	username
	Code signing requirement	3	0	-
	Network-related action	18	0	-
	Network status	14	0	-
OpenSSH	Failed authentication attempt	1491	1491	username, uid
	Unauthorized Access Detection	198	198	domain name, username
	Authentication ended	34	34	username
	Authentication success	1	1	username
	Authentication issue	10	0	-

6.3.1 Comparative Analysis of Privacy Policies

Table 6.2 delineates the diverse approaches to IP address collection by the three applications. [2] In our comparative analysis of privacy policies, we examined the approaches to IP address collection and the stated purposes for data collection across Linux, OpenSSH, and MacOS.

The privacy policy of Linux is relatively vague regarding IP address collection. It mentions gathering IP addresses, domain names, and browser information but does not provide explicit details on the purpose of this data collection beyond general service enhancement and debugging.

OpenSSH's policy acknowledges the collection of a broader range of data, including IP addresses, browser, operating system, and device details. However, it remains ambiguous about the specific purposes of data collection. The policy cites "multiple purposes" such as user consent and legitimate interests, which implies flexibility in data usage but lacks detailed information about how the data is used and protected.

In contrast, MacOS demonstrates a more comprehensive and detailed approach. Its policy acknowledges the collection of IP addresses, browser information, operating system details, and specifies the precision of the location data collected. MacOS stands out for its nuanced approach based on regional legislation, classifying data, including IP addresses, as personal where legally required. It also commits to treating amalgamated non-personal and personal data as personal data, reflecting a proactive stance towards privacy compliance. MacOS outlines an exhaustive list of purposes for data collection, ranging from operational necessities to legal compliance, offering a clearer picture of data usage.

6.3.2 Analysis of Personal Data in Logs

Table 6.4 shows the results of the analysis of IP addresses with other personal data, and the stated purposes.[3] In our analysis of log data from Linux, MacOS, and OpenSSH, we observed distinct patterns in the use of IP addresses, their combination with other personal data, and the stated purposes for data collection, which we then compared with their respective privacy policies.

[2] *as described in their privacy policies, which we last reviewed in October 2023.*

[3] *Our replication package :* https://doi.org/10.5281/zenodo.7988584

OpenSSH Logs Analysis: OpenSSH exhibited an extensive use of IP address logging, often combined with other personal identifiers. A striking majority of the log events (1,724 out of 1,734) involved IP addresses combined with usernames and UIDs, especially in 'Failed authentication attempt' and 'Unauthorized Access Detection' contexts. This pattern is aligned with OpenSSH's role in secure communications and suggests a potential overreach in data collection, raising significant privacy concerns. Systems like OpenSSH are designed for secure communications and are likely to log every connection attempt, which includes IP addresses for security purposes. This results in a higher number of log events associated with connection and authentication templates.

Linux Logs Analysis: Linux logs indicated a substantial focus on security and operational monitoring. Out of 1,337 log events containing IP addresses, 633 also included usernames or domain names. The high frequency of occurrences (356 in 'Authentication failure' and 979 in 'Connection origin and time') underscores a significant emphasis on user activities and system interactions. However, this amalgamation of IP addresses with other identifiers like usernames aligns with operational needs but introduces potential privacy challenges. Linux, being a versatile operating system, may have a broader range of templates due to the diversity of its applications, but not all events will necessarily involve network interactions that include IP addresses.

MacOS Logs Analysis: MacOS logs showed a more conservative approach. Among the 50 log events with IP addresses, only 15 involved a combination with usernames, reflecting a more restrained strategy in personal data handling. This lower frequency, especially in contexts like 'Network client connection' (10 occurrences) and 'Client action' (5 occurrences), indicates a more privacy-conscious approach, balancing operational requirements with user privacy. The significantly lower count in MacOS logs can be attributed to its primary use as a personal operating system. Unlike Linux or OpenSSH, which are heavily network-oriented and often deployed in server environments, MacOS is generally used in personal computing where network interactions are less frequent and diverse. As a result, the occurrence of IP addresses in MacOS logs is less common, reflected in the much lower count of such log events.

6.3.3 Comparative Analysis with Privacy Policies

In our analysis, we observed notable differences between the actual personal data captured in log files and the stated privacy policies of Linux, MacOS, and OpenSSH.

For Linux, the logs reveal a significant number of instances where IP addresses are combined with usernames, particularly in contexts like 'Authentication failure' and 'Connection origin and time.' This practice suggests a focus on monitoring user activities and system interactions. However, such detailed amalgamation of data, especially the combination with usernames, is not explicitly reflected in the Linux privacy policy, which primarily mentions the collection of IP addresses, domain names, and browser information for service enhancement and debugging. This discrepancy indicates a gap between the policy's broad statements and the specific practices observed in the logs.

In the case of MacOS, the logs indicate a collection of IP addresses in combination with usernames in a limited number of instances. However, this specific combination is not directly addressed in the MacOS privacy policy, which, while comprehensive in detailing the types of data collected (such as IP addresses, browser, OS, and device details), does not explicitly mention the collection of usernames. This omission represents a significant gap between the logging practices and the privacy policy, indicating a potential breach in privacy practices, regardless of the limited number of occurrences.

OpenSSH stands out for its extensive logging of IP addresses, often in combination with usernames and UIDs, as seen in a significant number of log events categorized under 'Failed authentication attempt' and 'Unauthorized Access Detection.' This extensive use of IP addresses for security-related purposes is broadly acknowledged in the OpenSSH privacy policy, which states the collection of a wide range of data under multiple purposes, including user consent and legitimate interests. However, the policy's general scope does not fully capture the specific, intensive use of IP addresses observed in the logs, suggesting a need for more precise and detailed policy articulations.

6.3.4 Discussion

Our research findings brings to the forefront the urgent need for enhanced personal data management strategies within logs. The following are key areas where immediate attention and action

are imperative:

Open-Source Systems - A Call for Enhanced Governance: In open-source environments like Linux and OpenSSH, the diverse and decentralized nature of development contributes to a varied approach to privacy. This diversity, while fostering innovation, potentially leads to inconsistencies in privacy practices. Our observations suggest a heightened privacy risk in these systems, underscoring the need for more structured governance around log data management. Implementing robust privacy-focused guidelines and enhancing oversight could significantly mitigate these risks.

Commercial Systems: Continuous Vigilance and Refinement: For commercial systems, as exemplified by MacOS, our study reveals a more cautious approach to logging practices. However, instances of data amalgamation, such as the combination of IP addresses with usernames, highlight that privacy concerns are not exclusive to open-source systems. Continuous refinement of privacy practices and regular audits are essential, even for commercial entities, to ensure alignment with evolving privacy standards and user expectations.

Advocacy for User-Centric Privacy Terms: Recognizing the prevalent issue of *forced consent,* where users must agree to privacy policies to access services, our study also critiques the transparency and user empowerment aspects of such policies. We advocate for clear, comprehensible, and accessible privacy terms that empower users with genuine choice and control over their personal data. To this end, we propose developing mechanisms that enable users to give explicit consent for each category of data being collected. For instance, instead of a single, blanket acceptance of all terms, privacy policies could present modular consent forms. These forms would allow users to opt-in or opt-out of specific data collection practices, particularly for data types like IP addresses, device details, or location data.

Engagement with Industry Stakeholders: To further these objectives, engaging with companies and industry stakeholders is crucial. Informing them of our findings is a step towards fostering a collaborative environment where privacy concerns are openly addressed. Extending this research to other systems would broaden our understanding of industry-wide practices, contributing to more comprehensive and effective privacy solutions.

6.4 Conclusion

In conclusion, this study breaks new ground by analyzing log files for personal data handling practices across diverse systems, a subject that has been given insufficient attention to date. Our methodological innovation lies in the granularity of our analysis, particularly in how IP addresses intertwine with other personal identifiers, revealing a landscape ripe for privacy enhancement.

In our investigation, we observed notable discrepancies in the logging practices and privacy policy transparency of Linux and OpenSSH when compared to MacOS. The open-source platforms demonstrated a concerning degree of privacy risks, compounded by the ambiguity in their privacy policies. This lack of clarity poses significant challenges in ensuring user data protection. Even within the comparatively stringent privacy framework of MacOS, we identified instances where the amalgamation of data contravened their own privacy policy stipulations.

Looking ahead, we aim to extend the conversation on privacy protections by sharing our findings with the companies managing these systems. Engaging with these entities is a critical step toward advocating for improved practices and ensuring that the implications of our research are translated into actionable change. Furthermore, the analysis of a broader range of logs will enrich our understanding of the privacy landscape, while collaboration with legal experts can deepen the study's impact by exploring the legal ramifications of our findings.

The path forward calls for a concerted effort involving industry stakeholders, legal authorities, and the research community to forge a consensus on privacy-respecting logging standards. By continuing this research and advocating for its application, we contribute to the vital endeavor of safeguarding personal data integrity and fostering trust in digital ecosystems.

Chapter 7

Conclusion and Future Work

"What we know is a drop; what we don't know is an ocean."

— Isaac Newton

The findings and contributions of this thesis represent a step forward in the field of log management, with a focus on improving log parsing accuracy, efficiency, privacy and adaptability. The key contributions of this work include:

- **Development of the Universal Log Parser (ULP)**: ULP leverages frequent token analysis to enhance log template extraction, effectively addressing the heterogeneity of log formats across different systems. This approach has been demonstrated to significantly improve parsing performance by adapting to diverse log structures.
- **Introduction of the Accuracy Metric for Log Parsing (AML)**: The AML framework provides a comprehensive means of evaluating the accuracy of log parsers, addressing the current lack of standardized and comprehensive metric. This metric enables a more objective and consistent assessment of the efficacy of log parsing techniques across a variety of log datasets.
- **Creation of a Comprehensive Taxonomy for Log Characteristics**: The taxonomy developed in this thesis categorizes the key attributes of log data, aiding in the characterization of logs and supporting more adaptive parsing techniques. This taxonomy forms the basis for evaluating log parsers and serves as a reference for understanding the complexity of log events.

These contributions collectively advance the field of log analysis, providing essential tools and frameworks that improve the way we process and understand system logs, especially in complex, large-scale environments. However, given the rapid evolution of modern computing systems and the increasing complexity of log data, there remains considerable scope for further research in this area.

Building upon the foundations laid in this thesis is critical for addressing the ongoing challenges posed by heterogeneous log formats, dynamic system behaviors, and large-scale data environments. The taxonomy of log characteristics developed here provides a strong framework for understanding the structural complexities of log data, but continued research is necessary to refine and adapt these insights to emerging technologies such as microservices and cloud-native systems. Leveraging

the taxonomy of Log Event Characteristics (LECs) to develop more robust log parsing techniques presents an exciting opportunity to address the persistent challenges in log parsing, particularly the heterogeneity of log formats and the dynamic nature of log tokens.

Log patterns represent a significant area for future research. By analyzing recurring templates and structural attributes within log events, we can better understand typical system behaviors and ”hot spots” that generate significant logging activity. The extraction of consistent log patterns could help classify logs components into meaningful categories such as normal operational behaviors, anomalies, or potential indicators of performance bottlenecks. These insights could ultimately lead to predictive maintenance models, enabling intelligent systems that provide real-time operational insights and enhancing both system reliability and performance. By studying the prevalence and frequency of these patterns, researchers can determine which system events are most critical or frequent. This allows for the identification of ”hot spots” in system behavior, where particular activities might be generating significant logging information. Moreover, understanding these patterns could lead to the classification of logs into meaningful categories, such as normal operational behaviors, anomalies, or potential indicators of performance bottlenecks.

The benefits of having a well-defined set of log patterns are substantial. For instance, identifying log patterns allows developers and operators to create targeted rules for alerting, which can significantly enhance Anomaly Detection. Additionally, log patterns can facilitate more efficient log storage by using the same template for repeated events, thereby compressing redundant information. They also help improve log parsing accuracy since consistent patterns allow parsers to more effectively differentiate between static and dynamic content within log messages.

Furthermore, the identification and categorization of these patterns make it easier to develop predictive maintenance models by determining which sequences of events may indicate an impending issue. In this way, log patterns can serve as the foundation for intelligent logging systems that are capable of providing operational insights in real-time, helping both developers and operators maintain system reliability and performance.

Another crucial area for further research is understanding the cost of logging. Real-time and high-traffic systems, like high-frequency trading platforms, demand extremely efficient logging to avoid performance bottlenecks. A refined taxonomy can guide selective logging of high-impact

events, reducing unnecessary logs while preserving essential system information. Establishing best practices for logging efficiency, such as batching or serialization techniques, will also be key in minimizing logging overhead.

The application of the LEC taxonomy for benchmarking and testing log parsers is another potential avenue for advancement. Developing a comprehensive benchmark will allow for a more structured evaluation of log parsing tools, focusing on parser accuracy, variability, hierarchical token complexity, and handling delimiters. This benchmark should include both unit and integration tests, ensuring that parsers can handle real-world complexities effectively. Such a testing process, combining synthetic and real log data, could provide a rigorous evaluation framework for parser performance and accuracy.

Enhancing the AML (Accuracy Metric for Log Parsing) framework introduced in this thesis also holds promise. Future work could refine AML to evaluate the impact of parsing errors on downstream analysis tools and combine accuracy metrics with computational efficiency. Another promising direction would be to integrate AML with performance metrics to balance both parsing accuracy and computational efficiency, ensuring that log parsing solutions are scalable for high-throughput, real-time environments.

Another important aspect to explore further is privacy and security in log parsing. With logs often containing sensitive data like IP addresses or PII, privacy-preserving parsing techniques that mask or obfuscate sensitive information during analysis are necessary. Future research should focus on developing privacy-aware parsing frameworks that align with data protection regulations, such as GDPR, ensuring compliance without sacrificing analytical power.

As we look to the future of log parsing, we stand at a moment where large language models (LLMs) and advances in AI are reshaping the way we handle unstructured data. LLMs, known for their remarkable capacity to process huge corpora and detect patterns in text, promise a powerful extension to existing log analytics. They can discover latent relationships within logs and offer intuitive explanations of system behavior that might otherwise remain hidden to traditional parsers. By applying advanced natural language processing, LLMs can identify anomalies, highlight causal links, or even propose remediation steps in near-real time.

However, while LLMs add a new level of adaptive intelligence, they do not entirely replace

structured log parsing methods. Most LLM-based approaches still benefit from having clean, well-defined inputs—especially for mission-critical applications where precision is paramount. Structured parsing provides the foundation that shapes raw logs into consistent templates, ensuring that key tokens, timestamps, and severity levels are accurately extracted. This structured data can then be ingested by LLMs to further explore nuanced patterns, facilitating more sophisticated insights and recommendations.

Moreover, although LLMs excel at understanding unstructured text, they can be prone to hallucinations or inaccuracies when the context is ambiguous or when the data does not align well with the model's training distribution. The careful evaluation and robust metrics introduced in this thesis—particularly the AML (Accuracy Metric for Log Parsing)—remain highly relevant in such scenarios. They can help gauge how accurately LLM-driven methods are interpreting logs, ensuring that faulty predictions do not propagate to critical decision-making processes. Similarly, the taxonomy of log characteristics developed here can guide the integration of LLMs by identifying which log types or patterns are best suited for advanced NLP techniques and where additional domain-specific models or rules might be needed.

In this evolving landscape, human engineers also maintain a vital role. While LLMs can automate the detection of anomalies and generate sophisticated analyses, subject-matter expertise is essential for interpreting results, validating recommendations, and ensuring ethical use of potentially sensitive log data. Humans bring domain knowledge, contextual judgment, and regulatory awareness that LLMs currently lack, especially for privacy-sensitive logs. Consequently, a collaborative model emerges: structured parsing and metrics-driven validation lay a stable groundwork; LLMs augment this structure with deeper pattern recognition; and human engineers oversee and fine-tune both the data pipelines and the interpretative processes to uphold reliability, compliance, and trust.

Ultimately, the synergy of structured methods (as proposed in this thesis) and LLM-based analytics allows organizations to leverage the best of both worlds—transforming raw log data into dependable, machine-friendly forms, then using advanced AI to extract higher-level insights. This fusion delivers robust, explainable, and trustworthy AIOps solutions that neither technology alone could fully achieve.

The evolution of logs themselves is another aspect to consider. Current logs reflect today's architecture, with discrete events captured in textual form, often requiring significant post-processing. As systems grow increasingly distributed and intelligent, logs may transform into more adaptive, real-time telemetry streams that integrate directly with machine learning models for on-the-fly system adaptation. This shift could reduce the need for traditional log analysis while enabling proactive system management.

However, as these systems become more sophisticated, ensuring transparency, accountability, and trust becomes crucial. If LLMs or future AI systems take over diagnosing, predicting, and optimizing system behavior, there is a risk that the systems become opaque, complicating accountability in failure moments. It will be essential to ensure that humans still understand how these systems work, even as they become more autonomous.

In the end, the future of log parsing will depend not only on technological advancements but also on our choices regarding the balance between automation and interpretability. As we march towards a future where systems can understand and adapt themselves, we must ask whether we will continue to analyze system behavior as we do today, or whether we are witnessing the beginning of a paradigm shift that will redefine logs and system intelligence. This thesis serves as a foundation for these future endeavors, offering the tools and insights necessary to push the boundaries of log management, AIOps, and system intelligence.

Bibliography

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

[2] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *Proc. of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2007)*, 2007, pp. 575—584.

[3] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proc. of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, ser. SOSP '09, New York, NY, USA, 2009, p. 117–132.

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

[5] Y. Dang, Q. Lin, and P. Huang, "Aiops: Real-world challenges and research innovations," in *Proc. of the 41st International Conference on Software Engineering (ICSE 2019), Companion Volume*, 2019, pp. 4—-5.

[6] G. Kim, P. Debois, J. Willis, and J. Humble, *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution Press, 2016.

[7] Z. Li, "Studying and suggesting logging locations in code blocks," in *in Proc. of the ACM/IEEE 42nd International Conference on Software Engineering: Companion*

Proceedings, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 125–127. [Online]. Available: https://doi.org/10.1145/3377812.3382168

[8] Q. Lin, K. Hsieh, Y. Dang, K. Zhang, H.and Sui, Y. Xu, J. Lou, C. Li, Y. Wu, R. Yao, R. Chintalapati, and D. Zhang, "Predicting node failure in cloud service systems," in *Proc. of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/SIGSOFT FSE 2018)*, 2018, pp. 480––490.

[9] Z. Li, T.-H. P. Chen, and W. Shang, "Where shall we log? studying and suggesting logging locations in code blocks," in *Proc. 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2020, pp. 361–372.

[10] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *Proc. of the 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2019, Montreal, QC, Canada, May 25-31, 2019*, H. Sharp and M. Whalen, Eds. IEEE / ACM, 2019, pp. 121–130. [Online]. Available: https://dl.acm.org/citation.cfm?id=3339932

[11] Y. Lyu, G. K. Rajbahadur, D. Lin, B. Chen, and Z. M. J. Jiang, "Towards a consistent interpretation of aiops models," *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 1, Nov. 2021. [Online]. Available: https://doi.org/10.1145/3488269

[12] S. Kobayashi, K. Fukuda, and H. Esaki, "Towards an nlp-based log template generation algorithm for system log analysis," in *Proc. of The Ninth International Conference on Future Internet Technologies*. ACM, 2014, p. 11.

[13] H. Dai, H. Li, C.-S. Chen, W. Shang, and T.-H. Chen, "Logram: Efficient log parsing using nn-gram dictionaries," *IEEE Transactions on Software Engineering*, vol. 48, no. 3, pp. 879–892, 2022.

[14] S. Thaler, V. Menkonvski, and M. Petkovic, "Towards a neural language model for signature extraction from forensic logs," in *Proc of the 5th International Symposium on Digital Forensic and Security (ISDFS)*. IEEE, 2017, pp. 1–6.

[15] Z. Jiang, J. Liu, Z. Chen, Y. Li, J. Huang, Y. Huo, P. He, J. Gu, and M. R. Lyu, "Llmparser: A llm-based log parsing framework," *ArXiv*, vol. abs/2310.01796, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:269123283

[16] V.-H. Le and H. Zhang, "Log parsing: How far can chatgpt go?" p. 1699–1704, 2024. [Online]. Available: https://doi.org/10.1109/ASE56229.2023.00206

[17] Z. Jiang, J. Liu, Z. Chen, Y. Li, J. Huang, Y. Huo, P. He, J. Gu, and M. R. Lyu, "Lilac: Log parsing using llms with adaptive parsing cache," *ACM Softw. Eng.*, vol. 1, no. FSE, Jul. 2024. [Online]. Available: https://doi.org/10.1145/3643733

[18] V.-H. Le and H. Zhang, "An evaluation of log parsing with chatgpt," *ArXiv*, vol. abs/2306.01590, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:263887216

[19] D. El-Masri, P. Fabio, Y.-G. Guéhéneuc, A. Hamou-Lhadj, and A. Bouziane, "A systematic literature review on automated log abstraction techniques," *Information and Software Technology*, vol. 122, pp. 106–276, 02 2020.

[20] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *Proc. of the 17th IEEE International Conference on Web Services (ICWS)*. IEEE, 2017, pp. 33–40.

[21] M. Du and F. Li, "Spell: Online streaming parsing of system event logs," in *Proc. of the 16th International Conference on Data Mining (ICDM 2016)*, 2016, p. 859–864.

[22] M. Mizutani, "Incremental mining of system log format," in *Proc. of the 2013 IEEE International Conference on Services Computing (SCC)*, 2013, pp. 595–602.

[23] Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora, "An automated approach for abstracting execution logs to execution events," *Journal of Software Maintenance*, vol. 20, no. 4, p. 249–267, 2008.

[24] K. Shima, "Length matters: Clustering system log messages using length of words," *ArXiv*,

vol. abs/1611.03213, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:16326353

[25] L. Tang, T. Li, and C.-S. Perng, "Logsig: Generating system events from raw textual logs," in *Proc. of the 20th ACM international conference on Information and knowledge management*. ACM, 2011, pp. 785–794.

[26] R. Vaarandi, "Mining event logs with slct and loghound," in *Proc. of the 2008 of the IEEE Network Operations and Management Symposium (NOMS)*. IEEE, 2008, pp. 1071–1074.

[27] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "A lightweight algorithm for message type extraction in system application logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 11, pp. 1921–1936, 2012.

[28] Q. Fu, J.-G. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *Proc. of the Ninth IEEE International Conference on Data Mining, ICDM'09*. IEEE, 2009, pp. 149–158.

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

[30] I. Sedki, A. Hamou-Lhadj, O. Ait-Mohamed, and M. A. Shehab, "An effective approach for parsing large log files," in *Proc. Of the 38th IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2022, pp. 1–12.

[31] R. Zhou, M. Hamdaqa, H. Cai, and A. Hamou-Lhadj, "Mobilogleak: A preliminary study on data leakage caused by poor logging practices," in *Proc. of the 2020 International Conference on Software Analysis, Evolution, and Reengineering (SANER 2020), ERA Track*, 2020, pp. 577–581.

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

[33] D. El-Masri, P. Fabio, Y.-G. Guéhéneuc, A. Hamou-Lhadj, and A. Bouziane, "A systematic literature review on automated log abstraction techniques," *Information and Software Technology*, vol. 122, pp. 106–276, 02 2020.

[34] W. Shang, Z. M. Jiang, H. Hemmati, B. Adams, A. E. Hassan, and P. Martin, "Assisting developers of big data analytics applications when deploying on hadoop clouds," in *Proc. of the 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 402–411.

[35] A. Miransky, A. Hamou-Lhadj, E. Cialini, and A. Larsson, "Operational-log analysis for big data systems: Challenges and solutions," *IEEE Software*, vol. 33, no. 2, pp. 55–59, 2016.

[36] W. Shang, Z. M. Jiang, B. Adams, A. E. Hassan, M. W. Godfrey, M. Nasser, and P. Flora, "An exploratory study of the evolution of communicated information about the execution of large software systems," in *Proc. of the 18th Working Conference on Reverse Engineering*, 2011, pp. 335–344.

[37] S. He, Q. Lin, J. Lou, H. Zhang, M. R. Lyu, and D. Zhang, "Identifying impactful service system problems via log analysis," in *Proc. of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/SIGSOFT FSE 2018)*, 2018, pp. 60—-70.

[38] D. Yuan, S. Park, P. Huang, Y. Liu, M. M. Lee, X. Tang, Y. Zhou, and S. Savage, "Be conservative: enhancing failure diagnosis with proactive logging," in *Proc. of the 10th USENIX conference on Operating Systems Design and Implementation*. USENIX Association, 2012, pp. 293–306.

[39] W. Shang, Z. M. Jiang, H. Hemmati, B. Adams, A. E. Hassan, and P. Martin, "Assisting developers of big data analytics applications when deploying on hadoop clouds," in *Proc. of the 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 402–411.

[40] S. Chittala, "Aiops and devops : Catalysts of digital transformation in the age of automated operations," *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 2024.

[41] N. Bosch and J. Bosch, "Software logging for machine learning," *ArXiv*, vol. abs/2001.10794, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:210942829

[42] P. Gupta, H. Kumar, D. Kar, K. Bhukar, P. Aggarwal, and P. Mohapatra, "Learning representations on logs for aiops," *Proc. of the 16th IEEE International Conference on Cloud Computing (CLOUD)*, pp. 155–166, 2023.

[43] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: System log analysis for anomaly detection," in *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2016, pp. 207–218.

[44] T. Barik, R. DeLine, S. M. Drucker, and D. Fisher, "The bones of the system: a case study of logging and telemetry at microsoft," in *Proc. of the 38th International Conference on Software Engineering (ICSE 2016), Companion Volume*, 2016, pp. 92–101.

[45] J. Cito, P. Leitner, T. Fritz, and H. C. Gall, "The making of cloud applications: an empirical study on software development for the cloud," in *Proc. of the 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015)*, 2015, pp. 393–403.

[46] P. Huang, C. Guo, J. R. Lorch, L. Zhou, and Y. Dang, "Capturing and enhancing in situ system observability for failure detection," in *Proc. of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2018)*, 2018, pp. 1—-16.

[47] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *Proc. of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2007)*, 2007, pp. 575—584.

[48] Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora, "Automatic identification of load testing problems," in *Proc. of the 24th IEEE International Conference on Software Maintenance (ICSM 2008)*, 2008, pp. 307—-316.

[49] K. Nagaraj, C. Killian, and J. Neville, "Structured comparative analysis of systems logs to diagnose performance problems," in *Proc. of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 26–26.

[50] M. Chow, D. Meisner, J. Flinn, D. Peek, and T. F. Wenisch, "The mystery machine: End-to-end performance analysis of large-scale internet services," in *Proc. of the 11th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'14. USA: USENIX Association, 2014, p. 217–231.

[51] R. Zhou, M. Hamdaqa, H. Cai, and A. Hamou-Lhadj, "Mobilogleak: A preliminary study on data leakage caused by poor logging practices," in *Proc. of the 2020 International Conference on Software Analysis, Evolution, and Reengineering (SANER 2020), ERA Track*, K. Kontogiannis, F. Khomh, A. Chatzigeorgiou, M.-E. Fokaefs, and M. Zhou, Eds. IEEE, 2020, pp. 577–581. [Online]. Available: https://doi.org/10.1109/SANER48275.2020.9054831

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

[53] B. Chen and Z. M. (Jack) Jiang, "Characterizing logging practices in java-based open source software projects — a replication study in apache software foundation," *Empirical Softw. Engg.*, vol. 22, no. 1, pp. 330–374, Feb. 2017. [Online]. Available: https://doi.org/10.1007/s10664-016-9429-5

[54] Y. Zeng, J. Chen, W. Shang, and T.-H. Chen, "Studying the characteristics of logging practices in mobile apps: a case study on f-droid," *Empirical Softw. Engg.*, vol. 24, no. 1, pp. 3394–3434, 2019. [Online]. Available: https://doi.org/10.1007/s10664-019-09687-9

[55] B. Chen and Z. M. J. Jiang, "Characterizing and detecting anti-patterns in the logging code," in *Proc. of the 39th International Conference on Software Engineering*, ser. ICSE '17. IEEE Press, 2017, pp. 71–81. [Online]. Available: https://doi.org/10.1109/ICSE.2017.15

[56] Z. Liu, X. Xia, D. Lo, Z. Xing, A. E. Hassan, and S. Li, "Which variables should i log?" *IEEE Transactions on Software Engineering*, 2019, early access.

[57] J. Zhu, P. He, Q. Fu, H. Zhang, M. R. Lyu, and D. Zhang, "Learning to log: Helping developers make informed logging decisions," in *Proc. of the 37th International Conference*

on Software Engineering - Volume 1, ser. ICSE '15. IEEE Press, 2015, pp. 415–425. [Online]. Available: http://jiemingzhu.github.io/pub/jmzhu_icse2015.pdf

[58] H. Li, W. Shang, and A. E. Hassan, "Which log level should developers choose for a new logging statement?" *Empirical Softw. Engg.*, vol. 22, no. 4, pp. 1684–1716, Aug. 2017. [Online]. Available: https://doi.org/10.1007/s10664-016-9456-2

[59] D. Yuan, H. Mai, W. Xiong, L. Tan, Y. Zhou, and S. Pasupathy, "Sherlog: Error diagnosis by connecting clues from run-time logs," in *Proc. of the Fifteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XV. New York, NY, USA: Association for Computing Machinery, 2010, pp. 143–154. [Online]. Available: https://doi.org/10.1145/1736020.1736038

[60] A. Oliner, A. Ganapathi, and W. Xu, "Advances and challenges in log analysis," *Commun. ACM*, vol. 55, no. 2, pp. 55–61, Feb. 2012. [Online]. Available: https://doi.org/10.1145/2076450.2076466

[61] Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora, "An automated approach for abstracting execution logs to execution events," *Journal of Software Maintenance*, vol. 20, no. 4, p. 249–267, 2008.

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

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

[64] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *Proc. of the 17th IEEE International Conference on Web Services (ICWS)*. IEEE, 2017, pp. 33–40.

[65] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "A lightweight algorithm for message type extraction in system application logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 11, pp. 1921–1936, 2012.

[66] Q. Fu, J.-G. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *Proc of the Ninth IEEE International Conference on Data Mining ICDM'09*. IEEE, 2009, pp. 149–158.

[67] S. Gholamian and P. A. S. Ward, "On the naturalness and localness of software logs," in *Proc of the IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, 2021, pp. 155–166.

[68] B. Chen and Z. M. J. Jiang, "Studying the use of java logging utilities in the wild," in *Proc. of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 397–408. [Online]. Available: https://doi.org/10.1145/3377811.3380408

[69] C. Zhi, J. Yin, S. Deng, M. Ye, M. Fu, and T. Xie, "An exploratory study of logging configuration practice in java," in *Proc of the 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2019, pp. 459–469.

[70] Y. Zeng, J. Chen, W. Shang, and T.-H. Chen, "Studying the characteristics of logging practices in mobile apps: a case study on f-droid," *Empirical Softw. Engg.*, vol. 24, no. 1, pp. 3394–3434, 2019. [Online]. Available: https://doi.org/10.1007/s10664-019-09687-9

[71] H. Li, T.-H. P. Chen, W. Shang, and A. E. Hassan, "Studying software logging using topic models," *Empirical Softw. Engg.*, vol. 23, no. 5, pp. 2655–2694, Oct. 2018. [Online]. Available: https://doi.org/10.1007/s10664-018-9595-8

[72] D. Yuan, S. Park, P. Huang, Y. Liu, M. M. Lee, X. Tang, Y. Zhou, and S. Savage, "Be conservative: Enhancing failure diagnosis with proactive logging," in *Proc. of the 10th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'12. USA: USENIX Association, 2012, pp. 293–306.

[73] Z. Li, H. Li, T.-H. Chen, and W. Shang, "Deeplv: Suggesting log levels using ordinal based neural networks," in *Proc. of the 43rd International Conference on Software Engineering (ICSE)*, 2021, pp. 1461–1472.

[74] R. Ding, H. Zhou, J.-G. Lou, H. Zhang, Q. Lin, Q. Fu, D. Zhang, and T. Xie, "Log2: A cost-aware logging mechanism for performance diagnosis," in *Proc. of the 2015 USENIX Conference on Usenix Annual Technical Conference*, ser. USENIX ATC '15. USA: USENIX Association, 2015, pp. 139–150.

[75] M. Du and F. Li, "Spell: Streaming parsing of system event logs," in *Proc. of the 16th IEEE International Conference on Data Mining (ICDM)*. IEEE, 2016, pp. 859–864.

[76] R. Vaarandi, "Mining event logs with slct and loghound," in *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*. IEEE, 2008, pp. 1071–1074.

[77] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *Proc. of the 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2019, Montreal, QC, Canada, May 25-31, 2019*, H. Sharp and M. Whalen, Eds. IEEE / ACM, 2019, pp. 121–130. [Online]. Available: https://dl.acm.org/citation.cfm?id=3339932

[78] L. Wu, J. Tordsson, E. Elmroth, and O. Kao, "Microrca: Root cause localization of performance issues in microservices," in *Proc. of the 2020 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2020, pp. 50–60.

[79] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "An evaluation study on log parsing and its use in log mining," in *Proc. of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2016, pp. 654–661.

[80] V.-H. Le and H. Zhang, "Log-based anomaly detection without log parsing," in *Proc. of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2021, pp. 492–504.

[81] Y. Fu, M. Yan, Z. Xu, X. Xia, X. Zhang, and D. Yang, "An empirical study of the impact of log parsers on the performance of log-based anomaly detection," *Empirical Software Engineering*, vol. 28, 2022.

[82] A. O. Portillo Dominguez and V. Ayala-Rivera, "Towards an efficient log data protection in software systems through data minimization and anonymization," in *Proc. of the 7th International Conference in Software Engineering Research and Innovation (CONISOFT)*. IEEE, 2019.

[83] M. Finck and F. Pallas, "They who must not be identified—distinguishing personal from non-personal data under the gdpr," *International Data Privacy Law*, vol. 10, no. 1, pp. 11–36, 2020.

[84] S. Conger, J. H. Pratt, and K. D. Loch, "Personal information privacy and emerging technologies," *Information Systems Journal*, vol. 23, pp. 401–417, 2013. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1111/j.1365-2575.2012.00402.x

[85] "General data protection regulation (gdpr)," Sep 2022. [Online]. Available: https://gdpr-info.eu/

[86] J. F. A. Murphy, "The general data protection regulation (gdpr)," *Irish medical journal*, vol. 111 5, p. 747, 2018.

[87] H. Li, L. Yu, and W. He, "The impact of gdpr on global technology development," *Journal of Global Information Technology Management*, vol. 22, no. 1, pp. 1–6, 2019. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/1097198X.2019.1569186

[88] D. A. Tamburri, "Design principles for the general data protection regulation (gdpr): a formal concept analysis and its evaluation," *Information Systems*, vol. 91, p. 101469, 2020.

[89] H. Gjermundrød, I. Dionysiou, and K. Costa, "Privacytracker: A privacy-by-design gdpr-compliant framework with verifiable data traceability controls," in *Proc. of the 2016 International Conference on Web Engineering*. Springer, 2016, pp. 3–15.

[90] B.-J. Koops and R. Leenes, "Privacy regulation cannot be hardcoded. a critical comment on the 'privacy by design' provision in data-protection law," *International Review of Law, Computers & Technology*, vol. 28, no. 2, pp. 159–171, 2014.

[91] A. Hamou-Lhadj, "Regulatory compliance and its impact on software development," in *Proc. of the 1st Workshop on Law Compliancy Issues in Organisational Systems and Strategies (iComply'10)*, 2010.

[92] "Global survey from dell technologies," http://www.dell.com/learn/us/en/uscorp1/press-releases/2016-10-11-dell-survey-shows-organizations-lack-awareness, last accessed: 2018-06-28.

[93] A. Reid, "The european court of justice case of breyer," *Journal of Information Rights, Policy and Practice*, vol. 1, no. 2, 2017.

[94] E. Markos, G. R. Milne, and J. W. Peltier, "Information sensitivity and willingness to provide continua: a comparative privacy study of the united states and brazil," *Journal of Public Policy & Marketing*, vol. 36, no. 1, pp. 79–96, 2017.

[95] I. Fouad, C. Santos, A. Legout, and N. Bielova, "Did i delete my cookies? cookies respawning with browser fingerprinting," *arXiv*, vol. abs/2105.04381, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:234336170

[96] V. N. Padmanabhan and L. Subramanian, "An investigation of geographic mapping techniques for internet hosts," in *Proc. of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, 2001, pp. 173–185.

[97] L. Zeng, Y. Xiao, H. Chen, B. Sun, and W. Han, "Computer operating system logging and security issues: a survey," *Security Communications Networks*, vol. 9, pp. 4804–4821, 2016.

[98] K. Patel, J. Faccin, A. Hamou-Lhadj *et al.*, "The sense of logging in the linux kernel," *Empirical Software Engineering*, vol. 27, p. 153, 2022.

[99] A. Lyons, J. Gamba, A. Shawaga, J. Reardon, J. Tapiador, S. Egelman, and N. Vallina-Rodriguez, "Log: It's big, It's heavy, It's filled with personal data! measuring the

logging of sensitive information in the android ecosystem," in *Proc. of the 32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, 2023, pp. 2115–2132. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/lyons

[100] X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang, "Pre-trained models for natural language processing: A survey," *Science China Technological Sciences*, vol. 63, no. 10, pp. 1872–1897, 2020.

[101] Y. Li, Y. Huo, Z. Jiang, R. Zhong, P. He, Y. Su, and M. Lyu, "Exploring the effectiveness of llms in automated logging generation: An empirical study," 07 2023.

[102] V.-H. Le and H. Zhang, "Log parsing: How far can chatgpt go?" in *Proc. of the 38th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '23. IEEE Press, 2024, p. 1699–1704. [Online]. Available: https://doi.org/10.1109/ASE56229.2023.00206

[103] Z. Ma, A. R. Chen, D. Kim, T.-H. Chen, and S. Wang, "Llmparser: An exploratory study on using large language models for log parsing," 04 2024, pp. 1–13.

[104] T. Ahmed, S. Ghosh, C. Bansal, T. Zimmermann, X. Zhang, and S. Rajmohan, "Recommending root-cause and mitigation steps for cloud incidents using large language models," in *Proc. of the 2023 IEEE International Conference on Software Engineering (ICSE)*, ser. ICSE '23. IEEE Press, 2023, p. 1737–1749. [Online]. Available: https://doi.org/10.1109/ICSE48619.2023.00149

[105] Y. Cao, S. Li, Y. Liu, Z. Yan, Y. Dai, P. S. Yu, and L. Sun, "A comprehensive survey of ai-generated content (aigc): A history of generative ai from gan to chatgpt," *arXiv preprint arXiv:2303.04226*, 2023.

[106] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[107] J. A. Baktash and M. Dawodi, "Gpt-4: A review on advancements and opportunities in natural language processing," *arXiv preprint arXiv:2305.03195*, 2023.

[108] S. Samsi, D. Zhao, J. McDonald, B. Li, A. Michaleas, M. Jones, W. Bergeron, J. Kepner, D. Tiwari, and V. Gadepally, "From words to watts: Benchmarking the energy costs of large language model inference," in *Proc. of the 2023 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2023, pp. 1–9.

[109] Y. Li, Y. Huo, Z. Jiang, R. Zhong, P. He, Y. Su, and M. R. Lyu, "Exploring the effectiveness of llms in automated logging generation: An empirical study," *arXiv preprint arXiv:2307.05950*, 2023.

[110] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proc. of the 22nd ACM SIGOPS Symposium on Operating Systems Principles*, ser. SOSP '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 117–132. [Online]. Available: https://doi.org/10.1145/1629575.1629587

[111] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: system log analysis for anomaly detection," in *Proc. of the 27th IEEE International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2016, pp. 207–218.

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

[113] M. Islam, K. Wael, and A. Hamou-Lhadj, "Anomaly detection techniques based on kappa-pruned ensembles," *IEEE Transactions on Reliability*, vol. 67, no. 1, pp. 212–229, 2018.

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

[115] M. Lemoudden and B. E. Ouahidi, "Managing cloud-generated logs using big data technologies," in *Proc. of the 2015 International Conference on Wireless Networks and Mobile Communications (WINCOM)*, 2015, pp. 1—7.

[116] A. Miransky, A. Hamou-Lhadj, E. Cialini, and A. Larsson, "Operational-log analysis for big data systems: Challenges and solutions," *IEEE Software*, vol. 33, no. 2, pp. 55–59, 2016.

[117] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *Proc. of the 41st International Conference on Software Engineering: Software Engineering in Practice*. IEEE Press, 2019, pp. 121–130.

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

[119] Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora, "An automated approach for abstracting execution logs to execution events," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 20, no. 4, pp. 249–267, 2008.

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

[121] Q. Fu, J.-G. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *Proc. of the Ninth IEEE International Conference on Data Mining (ICDM)*. IEEE, 2009, pp. 149–158.

[122] J. Lafferty, A. McCallum, and F. C. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Proc. of the 2001 International Conference on Machine Learning*, 2001, pp. 282–289.

[123] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "An evaluation study on log parsing and its use in log mining," in *Proc. of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2016, pp. 654–661.

[124] K. Shima, "Length matters: Clustering system log messages using length of words," *ArXiv*, vol. abs/1611.03213, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:16326353

[125] G. Macbeth, E. Razumiejczyk, and R. D. Ledesma, "Cliff´s delta calculator: A non-parametric effect size program for two groups of observations," *Universitas Psychologica*, vol. 10, no. 2, pp. 545–555, 2010. [Online]. Available: https://app.dimensions.ai/details/publication/pub.1113188555

[126] J. Romano, J. D. Kromrey, J. Coraggio, J. Skowronek, and L. Devine, "Exploring methods for evaluating group differences on the nsse and other surveys: Are the t-test and cohen'sd indices the most appropriate choices," in *Proc of the 2006 annual meeting of the Southern Association for Institutional Research*. Citeseer, 2006, pp. 1–51.

[127] S. He, Q. Lin, J. Lou, H. Zhang, M. R. Lyu, and D. Zhang, "Identifying impactful service system problems via log analysis," in *Proc. of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/SIGSOFT FSE 2018)*, 2018, pp. 60—-70.

[128] K. Nagaraj, C. Killian, and J. Neville, "Structured comparative analysis of systems logs to diagnose performance problems," in *Proc. of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 26–26.

[129] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection." in *Proc. of the 2010 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIXATC'10, p. 24.

[130] P. Huang, C. Guo, J. R. Lorch, L. Zhou, and Y. Dang, "Capturing and enhancing in situ system observability for failure detection," in *Proc. of the 13th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'18. USA: USENIX Association, 2018, p. 1–16.

[131] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Online system problem detection by

mining patterns of console logs," in *Proc. of the 9th IEEE International Conference on Data Mining (ICDM 2009)*, 2009, pp. 588—-597.

[132] W. Shang, Z. M. Jiang, B. Adams, A. E. Hassan, M. W. Godfrey, M. Nasser, and P. Flora, "An exploratory study of the evolution of communicated information about the execution of large software systems," in *Proc. of the 18th Working Conference on Reverse Engineering*, 2011, pp. 335–344.

[133] W. Shang, Z. M. Jiang, H. Hemmati, B. Adams, A. E. Hassan, and P. Martin, "Assisting developers of big data analytics applications when deploying on hadoop clouds," in *Proc. of the 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 402–411.

[134] L. Tang, T. Li, and C.-S. Perng, "Logsig: Generating system events from raw textual logs," in *Proc. of the 20th ACM international conference on Information and knowledge management*. ACM, 2011, pp. 785–794.

[135] Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora, "An automated approach for abstracting execution logs to execution events," *Journal of Software Maintenance*, vol. 20, no. 4, p. 249–267, 2008.

[136] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *Proc. of the 17th IEEE International Conference on Web Services (ICWS)*. IEEE, 2017, pp. 33–40.

[137] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "A lightweight algorithm for message type extraction in system application logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 11, pp. 1921–1936, 2012.

[138] A. A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "Clustering event logs using iterative partitioning," in *Proc. of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*, J. F. E. IV, F. Fogelman-Soulié, P. Flach, and M. Zaki, Eds. ACM, 2009, pp. 1255–1264. [Online]. Available: http://doi.acm.org/10.1145/1557019.1557154

[139] K. Shima, "Length matters: Clustering system log messages using length of words," *ArXiv*, vol. abs/1611.03213, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:16326353

[140] M. Nagappan and M. A. Vouk, "Abstracting log lines to log event types for mining software system logs," in *Proc. of the 7th International Working Conference on Mining Software Repositories, MSR 2010 (Co-located with ICSE), Cape Town, South Africa, May 2-3, 2010, Proceedings*, J. Whitehead and T. Zimmermann, Eds. IEEE, 2010, pp. 114–117. [Online]. Available: http://dx.doi.org/10.1109/MSR.2010.5463281

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

[142] S. Messaoudi, A. Panichella, D. Bianculli, L. Briand, and R. Sasnauskas, "A search-based approach for accurate identification of log message formats," in *Proc. of the 26th IEEE/ACM International Conference on Program Comprehension (ICPC'18)*. ACM, 2018, pp. 167–177.

[143] M. Mizutani, "Incremental mining of system log format," in *Proc. of the 2013 IEEE International Conference on Services Computing (SCC)*. IEEE, 2013, pp. 595–602.

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

[145] H. Dai, H. Li, C.-S. Chen, W. Shang, and T.-H. Chen, "Logram: Efficient log parsing using nn-gram dictionaries," *IEEE Transactions on Software Engineering*, vol. 48, no. 3, pp. 879–892, 2022.

[146] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *Proc. of the 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2019, Montreal, QC, Canada,*

May 25-31, 2019, H. Sharp and M. Whalen, Eds. IEEE / ACM, 2019, pp. 121–130. [Online]. Available: https://dl.acm.org/citation.cfm?id=3339932

[147] Z. A. Khan, D. Shin, D. Bianculli, and L. Briand, "Guidelines for assessing the accuracy of log message template identification techniques," in *Proc. of the 44th International Conference on Software Engineering*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1095–1106. [Online]. Available: https://doi.org/10.1145/3510003.3510101

[148] R. Congalton and K. Green, *Assessing the Accuracy of Remotely Sensed Data: Principles and Practices, Third Edition*, 08 2019.

[149] S. Koukoulas and G. A. Blackburn, "Introducing new indices for accuracy evaluation of classified images representing semi-natural woodland environments." *Photogrammetric Engineering and Remote Sensing*, vol. 67, pp. 499–510, 2001.

[150] C. Liu, P. Frazier, and L. Kumar, "Comparative assessment of the measures of thematic classification accuracy," *Remote Sensing of Environment*, vol. 107, pp. 606–616, 04 2007.

[151] S. He, J. Zhu, P. He, and M. R. Lyu, "Loghub: A large collection of system log datasets towards automated log analytics," *arXiv preprint arXiv:2008.06448*, 2020. [Online]. Available: https://arxiv.org/abs/2008.06448

[152] S. Zhang and G. Wu, "Efficient online log parsing with log punctuations signature," *Applied Sciences*, 2021.

[153] S. Nedelkoski, "Machine learning and knowledge discovery in databases: Applied data science track," in *Proc. of the 1993 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, 2021, pp. 122–138. [Online]. Available: https://doi.org/10.1007252F978-3-030-67667

[154] S. Chakrabartty, "Composite index: Methods and properties," *Journal of Applied Quantitative Methods, June, 2017, Volume 12, Issue 2*, vol. 12, 06 2017.

[155] J. Frost, *Introduction to Statistics: An Intuitive Guide for Analyzing Data and Unlocking Discoveries*, 2019.

[156] R. Graham, D. Knuth, and O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science*, 01 1994.

[157] K. Patel, J. G. Faccin, A. Hamou-Lhadj, and I. Nunes, "The sense of logging in the linux kernel," *Empirical Software Engineering*, vol. 27, no. 6, p. 153, 2022. [Online]. Available: https://doi.org/10.1007/s10664-022-10136-3

[158] B. Chen and Z. M. J. Jiang, "Characterizing logging practices in java-based open source software projects - a replication study in apache software foundation," *Empirical Software Engineering*, vol. 22, no. 1, pp. 330–374, 2017. [Online]. Available: http://dx.doi.org/10.1007/s10664-016-9429-5

[159] G. R. Gibbs, *Analyzing Qualitative Data*, 2007.

[160] M. Du and F. Li, "Spell: Streaming parsing of system event logs," in *Proc. of the 16th International Conference on Data Mining (ICDM 2016)*. IEEE, 2016, pp. 859–864.

[161] T. Zhang, H. Qiu, G. Castellano, M. Rifai, C. S. Chen, and F. Pianese, "System log parsing: A survey," *IEEE Transactions on Knowledge and Data Engineering*, 2023.

[162] "Gdpr fines after one year: Key takeaways for businesses," GDPR.eu. [Online]. Available: https://gdpr.eu/gdpr-fines-after-one-year-key-takeaways-for-businesses/