AML: An Accuracy Metric Model for Effective Evaluation of Log Parsing Techniques

Issam Sedki[a], Abdelwahab Hamou-Lhadj[a] and Otmane Ait-Mohamed[a]

[a]Concordia University, Montreal, Canada

ARTICLE INFO

Keywords:
Log Parsing
Log Analytics
Software Logging
AIOps
Software Maintenance and Evolution

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.

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 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 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 [5, 7, 8, 9], understanding system faults [5, 6, 7], performance and quality analysis [12, 13, 6], operational intelligence [8, 9, 12, 10, 14], failure prediction[9, 11], data leaks and security issues [60], and AI for IT Operations (AIOps) [15, 8]. Logs, however, are partially structured [5, 11], making the extraction of useful information from massive log files a challenging task [16, 17]. 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 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 have thousands of log templates [54, 55].

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

1.1. Problem Statement

Many log parsing techniques and tools have been proposed to automatically parse log events[1], such as LogSig [36], LogCluster [15], AEL [31], Drain [30], IpLom [28, 23], Lenma [34], LFA [25], LKE [35], LogMine [27], Molfi [29], Shiso [33], SLCT [24], Spell [37], Logram [26], and recently ULP [47]. 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 [21], Parsing Accuracy [26], and Template Accuracy [38]. These metrics have several drawbacks. For example, parsing accuracy is

*Corresponding author: Abdelwahab Hamou-Lhadj
i_sedki@live.concordia.ca (I. Sedki);
wahab.hamou-lhadj@concordia.ca (A. Hamou-Lhadj);
otmane.aitmohamed@concordia.ca (O. Ait-Mohamed)
ORCID(s):

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

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.

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 magnified by an entire order of magnitude [22].

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. [30] and Dai et al. [26] cite commendable effectiveness reflected by 99% and 93% respectively, yet the same assessments from Zhu et al. [21] and Sedki et al. [47] 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.

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 ([2], [19], [4]), 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 [41]. 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.

1.4. Paper Organization

This paper is structured as follows: In Section 2, we provide an overview of existing log parsing accuracy metrics and discuss their limitations. Section 3 delves into the details of the novel AML, explaining its methodology and capabilities. In Section 4, we apply AML to assess the accuracy of 14 log parsing tools, presenting our methodology. Section 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 6. Finally, we summarize our findings and contributions in Section 10, concluding the paper.

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. [53] 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. [21] 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. [38] 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

Table 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 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 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
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

identified templates such as over-generalized (OG), under-generalized (UG), and mixed (MX). Similar to [38], 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 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.

2.1. Grouping Accuracy

Grouping accuracy (GA) is used mainly by tools such as Drain [30] and AEL [31] 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 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 (1). 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:

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

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 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).

2.2. Parsing Accuracy

Several parsers such as Spell [37], Logpunk [39] 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 (2).

$$PA = \frac{K}{N} \tag{2}$$

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 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:

$$PA = \frac{\text{Number of Correctly Parsed Messages}}{\text{Total Number of Messages}} \tag{3}$$

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.

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 [61]. 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:

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

$$
\begin{aligned}
d(\text{"ERR: File no"}, \text{"ERR: File foun"}) = \\
1 + d(\text{"ERR: File n"}, \text{"ERR: File fou"}) \\
d(\text{"ERR: File n"}, \text{"ERR: File fou"}) = \\
1 + d(\text{"ERR: File "}, \text{"ERR: File fo"}) \\
d(\text{"ERR: File "}, \text{"ERR: File fo"}) = \\
2 + d(\text{"ERR: File "}, \text{"ERR: File "})
\end{aligned}
$$

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.

2.4. Template Accuracy

Recently, Khan et al. [38] 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:

$$PTA = \frac{O}{L} \tag{4}$$

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).

$$RTA = \frac{O}{N} \tag{5}$$

Consider the result of the parsing shown in Table 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).

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

$$RTA = 2/3 = 66\% \tag{7}$$

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.

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 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.

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 [2]. 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 [2]. 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 [2]. 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 [4]. 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 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

Table 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

Table 6
Computing errors of omission and commission for the example in Table 5

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

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.

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 8 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.

$$ErrO_{Ti} = \begin{cases} \frac{FN}{TP+FN} & TP + FN \neq 0 \\ 0 & TP + FN = 0 \end{cases} \quad (8)$$

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 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 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 9. 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 $ErrC_{Ti} = 0$.

$$ErrC_{Ti} = \begin{cases} \frac{FP}{TP+FP} & TP + FP \neq 0 \\ 0 & TP + FP = 0 \end{cases} \quad (9)$$

For the example in Table 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.

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

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

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.

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. [19] 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 [1] 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 10. AML aggregates the ICSIs of all templates to measure the overall performance of a parser as we will show in the next subsection.

$$ICSI_{Ti} = 1 - \left(ErrO_{Ti} + ErrC_{Ti}\right) \tag{10}$$

$ICSI_{Ti}$ 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.

3.3. Calculating AML

The AML metric is calculated using Equation 11, 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).

$$AML = \frac{\sum_{1}^{\max(N,D)} ICSI_i}{\max(N, D)} \tag{11}$$

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.

$$AML = \frac{1 + 0.6 - 0.16 + 0 + 0}{5} = 0.35 \tag{12}$$

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. Study Setup

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.[21] and EL-Masri et al. [53]. In this paper, we evaluate the accuracy of 14 log parsing tools, which include the 12 best-performing tools that were surveyed by Zhu et al.[21] and EL-Masri et al. [53]. We add to this list Logram [26] and ULP [47], which were recently released. We used the same parameter settings described by Zhu et al. [21] and the authors of Logram [26] and ULP [47]. 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 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.

4.2. Datasets

We evaluate the selected log parsing tools using 16 log datasets from the LogHub benchmark [41], which is available online[3]. The datasets consist of a collection of log files,

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

Table 7
The log parser used in this study

Parser	Key characteristic	Reference
Lenma	Computes similarity to templates of existing clusters.	[34]
Shiso	Measure resemblance of new templates to clusters.	[33]
LKE	Distance-based clustering technique	[35]
LogSig	Identifies log events using a set of signatures	[36]
Molfi	Parsing as a multiple objective optimization problem	[29]
ULP	Pattern recognition technique based on text similarity	[47]
SLCT	Mining line patterns and outlier events from textual event logs	[24]
Logmine	Uses MapReduce to abstract heterogeneous logs	[27]
LogCluster	Extracts terms from the logs into tuples	[15]
Spell	Relies on the longest common sequence	[37]
Drain	Abstracts logs into event using a tree	[30]
AEL	Relies on textual similarity to group logs together	[31]
IpLom	Employs a heuristic-based hierarchical clustering	[28, 23]
Logram	Leverages n-gram dictionaries	[26]

generated from various systems, including Apache, HPC, and HDFS as shown in Table 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 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 [30], Logram [26], ULP [47], Khan et al.[38]). Table 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.

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 `blk_` string from the ground truth file. We made similar corrections to other files when the errors were straightforward. Khan et al. in [38] 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.

4.3. 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?

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.

Table 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

Table 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 * *`

5. Results

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

Table 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 (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%).

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

Table 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 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:

Table 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

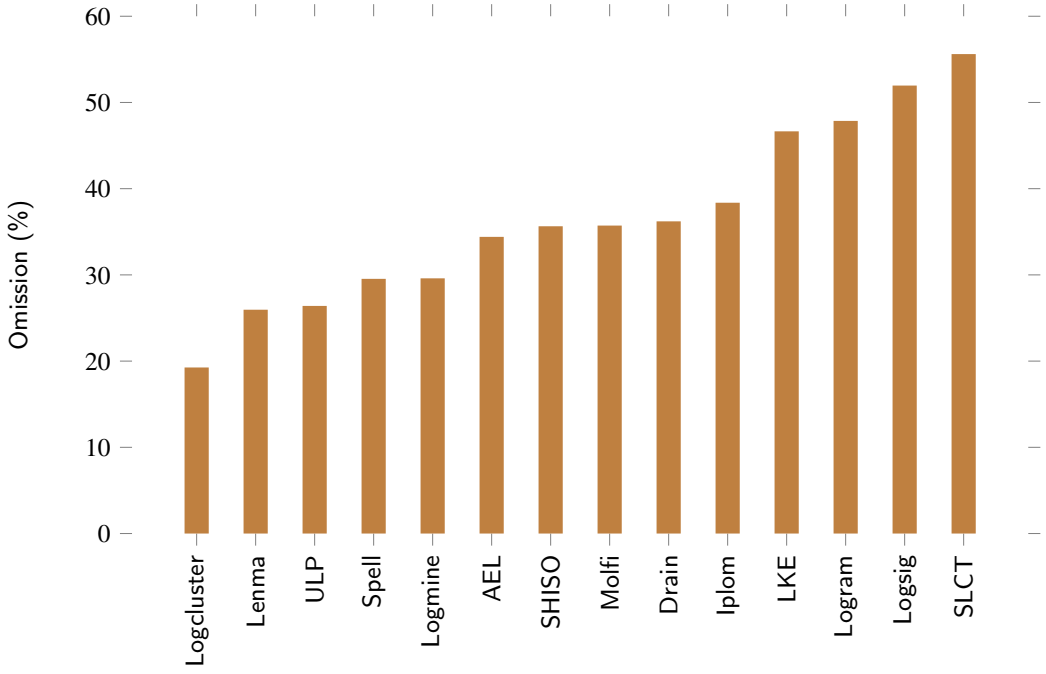

Figure 2: Ranking of log parsers by omission error ratio

`* * 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 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%).

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*`

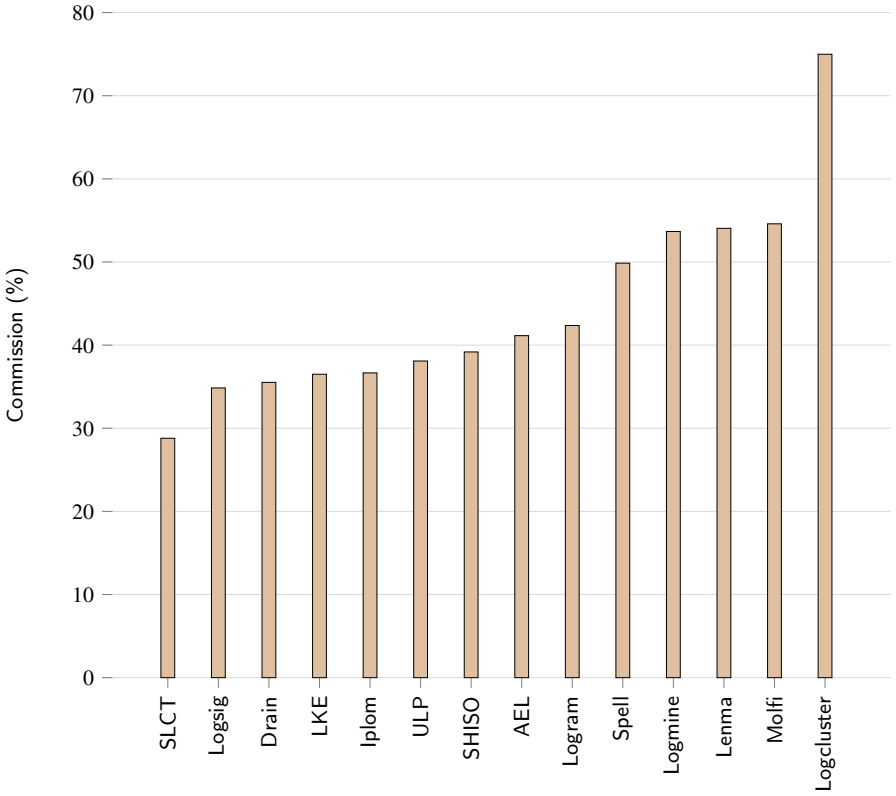

Figure 3: Ranking of the log parsers based on the average commission error ratio

`NameSystem.delete: 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. [22]. 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 11 and 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 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 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 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 in measuring the degree of a linear relationship between two continuous variables. This metric is particularly relevant in

Table 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

Table 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

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

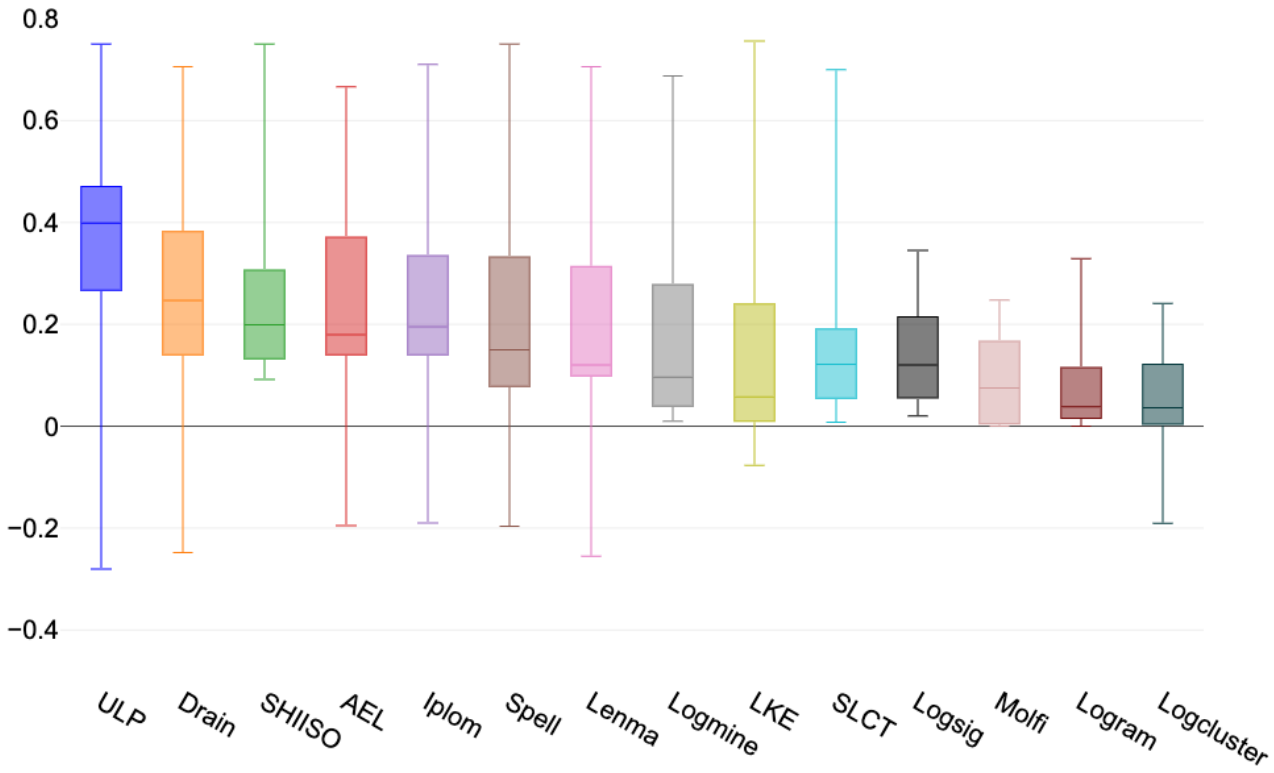

Figure 4: Accuracy distribution of log parsers across the datasets

Table 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

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.

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 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 5 provides a different 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.

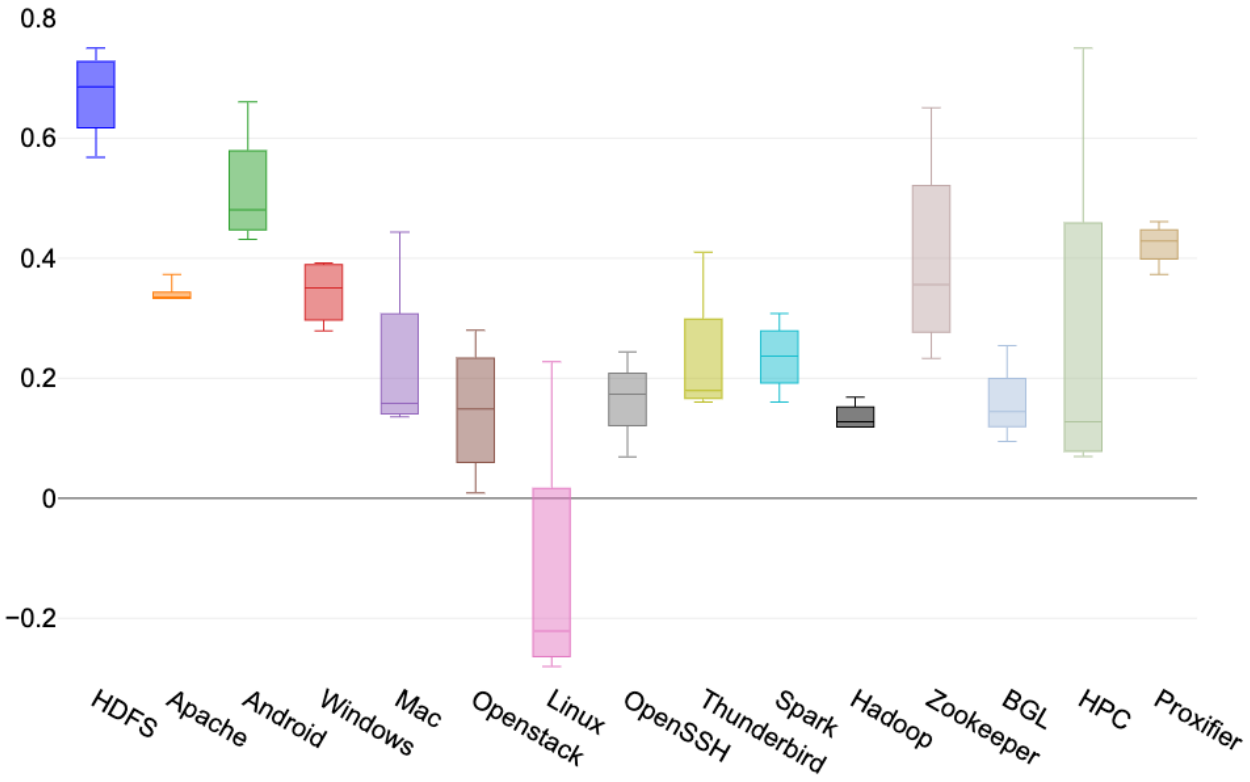

Figure 5: Variability analysis of ULP, Drain, SHISO, and AEL by dataset

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

Table 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 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 [57] 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 [57]. 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

Table 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

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:

$$\delta = \frac{\sum_{i,j}[x_i > x_j] - [x_i < x_j]}{m \times n} \tag{13}$$

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 [3].

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 $|\delta| \geq 0.474$ [57]. 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 $\delta = 1.00$), indicating a significant difference between the two accuracy metrics. Similarly, for PA, the effect size was also large (Cliff's $\delta = 0.76$). On the other hand, the effect size between AML and PTA was moderate (Cliff's $\delta = 0.43$), suggesting a noticeable difference between the two metrics. The effect size between AML and RTA was large (Cliff's $\delta = 0.51$), 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 [57], which is a non-parametric correlation coefficient calculated using Equation 14 where d_i represents the difference between the two ranks of each observation and n represents the number of observations.

$$Rs = 1 - \frac{6 * \sum_1^n d_i^2}{n(n^2 - 1)} \tag{14}$$

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.

Table 15
Correlation of different metrics with AML

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

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 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

Table 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 plugged-Type: *	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

- 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 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 [41] [5] often necessitate the specification of these delimiters during the pre-processing phase to streamline the parsing process.

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

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, 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.

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.

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

Table 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`

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.

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. [59] on the practice of logging in the Linux system, and the work of Chen et al. [58] 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.

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.

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

Table 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

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 `*` 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.

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:

$$WeightedAML = \frac{\sum_1^{\max(N,D)} (WeightedICSI_i)}{\max(N, D)} \quad (15)$$

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 18 compares normal AML to weighted AML.

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.

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 4.2. 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

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.

8. Replication Package

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

9. Acknowledgment

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

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.

References

[1] S.N, C. Composite Index: Methods and Properties. *Journal Of Applied Quantitative Methods*. **12** (2017)

[2] R.G, C. K, G. Assessing the accuracy of remotely sensed data: principles and practices. *CRC Press; 3rd Edition*. (2020)

[3] Graham, R.L., Knuth, D.E., Patashnik, O. and Liu, S., 1989. Concrete mathematics: a foundation for computer science. Computers in Physics, 3(5), pp.106-107.

[4] Liu, C., Frazier, P. Kumar, L. Comparative assessment of the measures of thematic classification accuracy. *Remote Sensing Of Environment*. **107**, 606-616 (2007)

[5] Xu, W., Huang, L., Fox, A., Patterson, D. & Jordan, M. Detecting Large-Scale System Problems by Mining Console Logs. *Proceedings Of The ACM SIGOPS 22nd Symposium On Operating Systems Principles*. pp. 117-132 (2009)

[6] He, S., Zhu, J., He, P. & Lyu, M. Experience report: System log analysis for anomaly detection. *2016 IEEE 27th International Symposium On Software Reliability Engineering (ISSRE)*. pp. 207-218 (2016)

[7] Lin, Q., Zhang, H., Lou, J., Zhang, Y. & Chen, X. Log clustering based problem identification for online service systems. *Proceedings Of The 38th Infternational Conference On Software Engineering Companion*. pp. 102-111 (2016)

[8] Oliner, A. & Stearley, J. What supercomputers say: A study of five system logs. *Proceedings Of The 37th Annual IEEE/IFIP International Conference On Dependable Systems And Networks (DSN 2007)*. pp. 575–584 (2007)

[9] Lin, Q., Hsieh, K., Dang, Y., Zhang, K., Xu, Y., Lou, J., Li, C., Wu, Y., Yao, R., Chintalapati, R. & Zhang, D. Predicting node failure in cloud service systems. *Proceedings Of The 2018 ACM Joint Meeting On European Software Engineering Conference And Symposium On The Foundations Of Software Engineering (ESEC/SIGSOFT FSE 2018)*. pp. 480–490 (2018)

[10] Lou, J., Fu, Q., Yang, S., Xu, Y. & Li, J. Mining Invariants from Console Logs for System Problem Detection.. *USENIX Annual Technical*

Conference. pp. 23-25 (2010)

[11] Xu, W., Huang, L., Fox, A., Patterson, D. & Jordan, M. Online system problem detection by mining patterns of console logs. *Proceedings Of The 9th IEEE International Conference On Data Mining (ICDM 2009)*. pp. 588–597 (2009)

[12] He, S., Lin, Q., Lou, J., Zhang, H., Lyu, M. & Zhang, D. Identifying impactful service system problems via log analysis. *Proceedings Of The 2018 ACM Joint Meeting On European Software Engineering Conference And Symposium On The Foundations Of Software Engineering (ESEC/SIGSOFT FSE 2018)*. pp. 60–70 (2018)

[13] Nagaraj, K., Killian, C. & Neville, J. Structured comparative analysis of systems logs to diagnose performance problems. *Proceedings Of The 9th USENIX Conference On Networked Systems Design And Implementation*. pp. 26-26 (2012)

[14] Huang, P., Guo, C., Lorch, J., Zhou, L. & Dang, Y. Capturing and Enhancing in Situ System Observability for Failure Detection. *Proceedings Of The 13th USENIX Conference On Operating Systems Design And Implementation*. pp. 1-16 (2018)

[15] Vaarandi, R. & Pihelgas, M. LogCluster-A data clustering and pattern mining algorithm for event logs. *Network And Service Management (CNSM), 2015 11th International Conference On*. pp. 1-7 (2015)

[16] Shang, W., Jiang, Z., Adams, B., Hassan, A., Godfrey, M., Nasser, M. & Flora, P. An Exploratory Study of the Evolution of Communicated Information about the Execution of Large Software Systems. *2011 18th Working Conference On Reverse Engineering*. pp. 335-344 (2011)

[17] Shang, W., Jiang, Z., Hemmati, H., Adams, B., Hassan, A. & Martin, P. Assisting developers of Big Data Analytics Applications when deploying on Hadoop clouds. *2013 35th International Conference On Software Engineering (ICSE)*. pp. 402-411 (2013)

[18] Campbell, J. Introduction to remote sensing. *Geocarto International*. **2** pp. 64-64 (1987)

[19] Koukoulas, S. Blackburn, G. Introducing New Indices for Accuracy Evaluation of Classified Images Representing Semi-Natural Woodland Environments.. Photogrammetric Engineering And Remote Sensing. **67** pp. 499-510 (2001)

[20] Fu, Q., Zhu, J., Hu, W., Lou, J., Ding, R., Lin, Q., Zhang, D. & Xie, T. Where do developers log? an empirical study on logging practices in industry. *36th International Conference On Software Engineering, ICSE '14, Companion Proceedings, Hyderabad, India, May 31 - June 07, 2014*. pp. 24-33 (2014), http://doi.acm.org/10.1145/2591062.2591175

[21] Zhu, J., He, S., Liu, J., He, P., Xie, Q., Zheng, Z. & Lyu, M. Tools and benchmarks for automated log parsing. *Proceedings Of The 41st International Conference On Software Engineering: Software Engineering In Practice, ICSE (SEIP) 2019, Montreal, QC, Canada, May 25-31, 2019*. pp. 121-130 (2019), https://dl.acm.org/citation.cfm?id=3339932

[22] He, P., Zhu, J., He, S., Li, J. & Lyu, M. An evaluation study on log parsing and its use in log mining. *2016 46th Annual IEEE/IFIP International Conference On Dependable Systems And Networks (DSN)*. pp. 654-661 (2016)

[23] Makanju, A., Zincir-Heywood, A. & Milios, E. Clustering event logs using iterative partitioning. *Proceedings Of The 15th ACM SIGKDD International Conference On Knowledge Discovery And Data Mining, Paris, France, June 28 - July 1, 2009*. pp. 1255-1264 (2009), http://doi.acm.org/10.1145/1557019.1557154

[24] Vaarandi, R. A data clustering algorithm for mining patterns from event logs. *IP Operations Management, 2003.(IPOM 2003). 3rd IEEE Workshop On*. pp. 119-126 (2003)

[25] Nagappan, M. & Vouk, M. Abstracting log lines to log event types for mining software system logs. *Proceedings 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*. pp. 114-117 (2010), http://dx.doi.org/10.1109/MSR.2010.5463281

[26] Dai, H., 0007, H., Chen, C., Shang, W. & Chen, T. Logram: Efficient Log Parsing Using n-Gram Dictionaries. *IEEE Trans. Software Eng.*. **48**, 879-892 (2022), https://doi.org/10.1109/TSE.2020.3007554

[27] Hamooni, H., Debnath, B., Xu, J., Zhang, H., Jiang, G. & Mueen, A. LogMine: fast pattern recognition for log analytics. *Proceedings Of The 25th ACM International On Conference On Information And Knowledge Management*. pp. 1573-1582 (2016)

[28] Makanju, A., Zincir-Heywood, A. & Milios, E. A lightweight algorithm for message type extraction in system application logs. *IEEE Transactions On Knowledge And Data Engineering*. **24**, 1921-1936 (2012)

[29] Messaoudi, S., Panichella, A., Bianculli, D., Briand, L. & Sasnauskas, R. A Search-based Approach for Accurate Identification of Log Message Formats. *Proceedings Of The 26th IEEE/ACM International Conference On Program Comprehension (ICPC'18)*. pp. 167-177 (2018)

[30] He, P., Zhu, J., Zheng, Z. & Lyu, M. Drain: An online log parsing approach with fixed depth tree. *Web Services (ICWS), 2017 IEEE International Conference On*. pp. 33-40 (2017)

[31] Jiang, Z., Hassan, A., Hamann, G. & Flora, P. An Automated Approach for Abstracting Execution Logs to Execution Events. *Journal Of Software Maintenance*. **20**, 249-267 (2008)

[32] Mizutani, M. Incremental mining of system log format. *Services Computing (SCC), 2013 IEEE International Conference On*. pp. 595-602 (2013)

[33] Mizutani, M. Incremental mining of system log format. *Services Computing (SCC), 2013 IEEE International Conference On*. pp. 595-602 (2013)

[34] Shima, K. Length Matters: Clustering System Log Messages using Length of Words. *CoRR*. **abs/1611.03213** (2016)

[35] Fu, Q., Lou, J., Wang, Y. & Li, J. Execution anomaly detection in distributed systems through unstructured log analysis. *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference On*. pp. 149-158 (2009)

[36] Tang, L., Li, T. & Perng, C. LogSig: Generating system events from raw textual logs. *Proceedings Of The 20th ACM International Conference On Information And Knowledge Management*. pp. 785-794 (2011)

[37] Du, M. & Li, F. Spell: Online streaming parsing of system event logs. *Proceedings Of The 16th International Conference On Data Mining (ICDM 2016)*. pp. 859-864 (2016)

[38] Khan, Z., Shin, D., Bianculli, D. & Briand, L. Guidelines for Assessing the Accuracy of Log Message Template Identification Techniques. *Proceedings Of The 44th International Conference On Software Engineering*. pp. 1095-1106 (2022), https://doi.org/10.1145/3510003.3510101

[39] Zhang, S. & Wu, G. Efficient Online Log Parsing with Log Punctuations Signature. *Applied Sciences*. (2021)

[40] Mohammad, H. Sulaiman, M. A REVIEW ON EVALUATION METRICS FOR DATA CLASSIFICATION EVALUATIONS. *International Journal Of Data Mining Knowledge Management Process*. (2015)

[41] He, S., Zhu, J., He, P. & Lyu, M. Loghub: A Large Collection of System Log Datasets towards Automated Log Analytics. *ArXiv*. **abs/2008.06448** (2020)

[42] Wilson, S. Mining Oblique Data with XCS. *IWLCS*. (2000)

[43] Ranawana, R. & Palade, V. Optimized Precision - A New Measure for Classifier Performance Evaluation. *2006 IEEE International Conference On Evolutionary Computation*. pp. 2254-2261 (2006)

[44] Gu, Q., Zhu, L. & Cai, Z. Evaluation Measures of the Classification Performance of Imbalanced Data Sets. *ISICA 2009, CCIS 51. Berlin, Heidelberg*. (2009)

[45] Han, S., Yuan, B. & Liu, W. Rare Class Mining: Progress and Prospect. *2009 Chinese Conference On Pattern Recognition*. pp. 1-5 (2009)

[46] He, H. & Ma, Y. Assessment Metrics for Imbalanced Learning. *Wiley IEEE Press, 2013, Pp. 187-210*. (2013)

[47] Sedki, I., Hamou-Lhadj, A., AitMohamed, O. & Shehab, M. An Effective Approach for Parsing Large Log Files. *2022 IEEE International Conference On Software Maintenance And Evolution (ICSME)*. (2022)

[48] Provost, F. & Fawcett, T. Analysis and Visualization of Classifier Performance: Comparison Under Imprecise Class and Cost Distributions. *Proceedings Of The Third International Conference On Knowledge Discovery And Data Mining*. **43-48** (1999,12)

[49] Provost, F. & Fawcett, T. Robust Classification for Imprecise Environments. *Machine Learning*. **42** pp. 203-231 (2001,1)

[50] Ho, T. & Basu, M. Complexity Measures of Supervised Classification Problems. *IEEE Trans. Pattern Anal. Mach. Intell.*. **24** pp. 289-300 (2002,3)

[51] Hossin, M. OAERP: A Better Measure than Accuracy in Discriminating a Better Solution for Stochastic Classification Training. *Journal Of Artificial Intelligence*. **4** pp. 187-196 (2011,6)

[52] Lingras, P. & Butz, C. Precision and Recall in Rough Support Vector Machines. *Proceedings - 2007 IEEE International Conference On Granular Computing, GrC 2007*. pp. 654-654 (2007,12)

[53] Diana, E., Petrillo, F., Guéhéneuc, Y., Hamou-Lhadj, A. & Bouziane, A. A Systematic Literature Review on Automated Log Abstraction Techniques. *Information And Software Technology*. **122** pp. 106-276 (2020,2)

[54] Chow, M., Meisner, D., Flinn, J., Peek, D. & Wenisch, T. The Mystery Machine: End-to-End Performance Analysis of Large-Scale Internet Services. *Proceedings Of The 11th USENIX Conference On Operating Systems Design And Implementation*. pp. 217-231 (2014)

[55] Mi, H., Wang, H., Zhou, Y., Lyu, M. & Cai, H. Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems. *IEEE Transactions On Parallel And Distributed Systems*. **24**, 1245-1255 (2013)

[56] He, P., Zhu, J., He, S., Li, J. & Lyu, M. An evaluation study on log parsing and its use in log mining. *2016 46th Annual IEEE/IFIP International Conference On Dependable Systems And Networks (DSN)*. pp. 654-661 (2016)

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

[58] Chen, B. & Jiang, Z. Characterizing logging practices in Java-based open source software projects - a replication study in Apache Software Foundation. *Empirical Software Engineering*. **22**, 330-374 (2017), http://dx.doi.org/10.1007/s10664-016-9429-5

[59] Patel, K., Faccin, J., Hamou-Lhadj, A. & Nunes, I. The sense of logging in the Linux kernel. *Empirical Software Engineering*. **27**, 153 (2022), https://doi.org/10.1007/s10664-022-10136-3

[60] Zhou, R., Hamdaqa, M., Cai, H. & Hamou-Lhadj, A. MobiLogLeak: A Preliminary Study on Data Leakage Caused by Poor Logging Practices. *27th IEEE International Conference On Software Analysis, Evolution And Reengineering, SANER 2020, London, ON, Canada, February 18-21, 2020*. pp. 577-581 (2020), https://doi.org/10.1109/SANER48275.2020.9054831

[61] Sasho Nedelkoski Self-supervised Log Parsing. *Machine Learning And Knowledge Discovery In Databases: Applied Data Science Track*. pp. 122-138 (2021), https://doi.org/10.1007