LogParser-LLM: Advancing Efficient Log Parsing with Large Language Models

Aoxiao Zhong*
zhongaoxiao@gmail.com
Harvard University
Cambridge, MA, US
Alibaba Group
Bellevue, WA, US

Dengyao Mo
dengyao.mo@alibaba-inc.com
Alibaba Group
Bellevue, WA, US

Guiyang Liu
wuming.lgy@alibaba-inc.com
Alibaba Group
Hangzhou, Zhejiang, China

Jinbu Liu
liujinbu.ljb@alibaba-inc.com
Alibaba Group
Hangzhou, Zhejiang, China

Qingda Lu
qingda.lu@alibaba-inc.com
Alibaba Group
Bellevue, WA, US

Qi Zhou
jackson.zhouq@alibaba-inc.com
Alibaba Group
Hangzhou, Zhejiang, China

Jiesheng Wu
jiesheng.wu@alibaba-inc.com
Alibaba Group
Bellevue, WA, US

Quanzheng Li
li.quanzheng@mgh.harvard.edu
CAMCA
Harvard Medical School,
Massachusetts General Hospital
Boston, MA, US

Qingsong Wen
qingsongedu@gmail.com
Alibaba Group
Bellevue, WA, US

Abstract

Logs are ubiquitous digital footprints, playing an indispensable role in system diagnostics, security analysis, and performance optimization. The extraction of actionable insights from logs is critically dependent on the log parsing process, which converts raw logs into structured formats for downstream analysis. Yet, the complexities of contemporary systems and the dynamic nature of logs pose significant challenges to existing automatic parsing techniques. The emergence of Large Language Models (LLM) offers new horizons. With their expansive knowledge and contextual prowess, LLMs have been transformative across diverse applications. Building on this, we introduce LogParser-LLM, a novel log parser integrated with LLM capabilities. This union seamlessly blends semantic insights with statistical nuances, obviating the need for hyper-parameter tuning and labeled training data, while ensuring rapid adaptability through online parsing. Further deepening our exploration, we address the intricate challenge of parsing granularity, proposing a new metric and integrating human interactions to allow users to calibrate granularity to their specific needs. Our method's efficacy is empirically demonstrated through evaluations on the Loghub-2k and the large-scale LogPub benchmark. In evaluations on the LogPub benchmark, involving an average of 3.6 million logs per dataset across 14 datasets, our LogParser-LLM requires only 272.5 LLM invocations on average, achieving a 90.6% F1 score for grouping accuracy and an 81.1% for parsing accuracy. These results demonstrate the method's high efficiency and accuracy, outperforming current state-of-the-art log parsers, including pattern-based, neural network-based, and existing LLM-enhanced approaches.

CCS Concepts

• **Computing methodologies** → **Natural language processing**; • **Applied computing** → **Document management and text processing**.

Keywords

Log parsing, Large language models, AIOps

ACM Reference Format:
Aoxiao Zhong, Dengyao Mo, Guiyang Liu, Jinbu Liu, Qingda Lu, Qi Zhou, Jiesheng Wu, Quanzheng Li, and Qingsong Wen. 2024. LogParser-LLM: Advancing Efficient Log Parsing with Large Language Models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24), August 25–29, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3637528.3671810

1 Introduction

Logs are pervasive records in the digital realm, vital for system diagnostics, security analysis, and performance optimization. As we navigate the complexities of contemporary digital environments, our systems, applications, and networks consistently generate vast amounts of logs. These abundant logs serve as an invaluable resource for understanding system behaviors, tracking activities, and uncovering hidden patterns. Their importance cannot be overstated, especially given the sophisticated nature of present-day systems and the crucial need for maintaining robust and efficient operations.

KDD '24, August 25–29, 2024, Barcelona, Spain.

ACM ISBN 979-8-4007-0490-1/24/08
https://doi.org/10.1145/3637528.3671810

*This work was completed during an internship at Alibaba Cloud US.

This rich information source of cloud computing aids in tasks such as anomaly detection [6, 48, 51], failure prediction [26, 50], and failure diagnosis [14, 52]. In this digital age, effective utilization of logs can be the deciding factor between seamless operations and significant downtimes, highlighting their paramount importance in ensuring system reliability and security.

Log parsing is a foundational step for many log-based diagnostic processes. Its main objective is to convert semi-structured log messages into a structured format, serving as the first step in a range of log analysis methods (e.g., [1, 6, 23]). This process entails identifying static components (referred to as log templates) and variable elements (known as log parameters) within log messages as shown in Figure 1. Traditional methods for log parsing often involve comparing raw log messages to logging statements found in the source code [33, 36, 37] or creating regex patterns manually. However, with the growing volume and diversity of log messages, as well as the rapid evolution of modern software systems, these methods have become increasingly impractical [52]. Consequently, there has been a significant push towards developing automated log parsers, given their pivotal role in Artificial Intelligence for IT Operations (AIOps) [3, 14].

To overcome the limitations of traditional log parsers, data-driven approaches have been developed by applying data mining techniques. Syntax-based methods were firstly developed, including frequent pattern mining [2, 32, 41, 42], clustering [7, 12, 30, 38, 39], and heuristic-based [5, 13, 18, 27, 28, 49]. These methods operate on the principle that tokens remaining consistent across logs are likely templates, while those that differ are treated as parameters. The task becomes extracting the common parts from raw log messages. The lack of consideration for semantic meanings in logs leads to inaccurate identification of parameters, particularly for infrequently occurring logs. The reliance on hyper-parameters, such as pre-defined frequency thresholds or similarity thresholds, is a notable drawback of syntax-based methods. It necessitates careful tuning for specific log sources, thereby significantly restricting the parser's ability to generalize across diverse log data sources. To take the semantic meaning into consideration, recent studies utilize neural networks for log parsing. Uniparser [25] uses LSTM aiming to build a universal parser that works for heterogeneous log data. LogPPT [21] uses a pretrained transformer with few-shot learning. The necessity of labeled data for these methods, combined with their demonstrated underperformance on large-scale evaluations [17], renders them impractical to implement due to the scarcity of computing resources and labeled data.

Large Language Models (LLMs) have demonstrated remarkable capabilities across various domains. Given the vast pretraining datasets that encompass code and logging-related data, LLMs possess immense potential for log parsing. Pioneering research, as referenced in studies such as [20, 24, 31, 46], delved into LLM-based log parsing. However, the predominant focus of these studies has been on prompt engineering to enhance template extraction. Such methods parse logs line by line, incurring significant computational overhead due to the billions of parameters in LLMs. This renders these approaches somewhat impractical for broader applications.

To harness the capabilities of LLMs for practical log parsing, we introduce LogParser-LLM, which stands for **Log Parser** with **L**arge **L**anguage **M**odels. At its core, LogParser-LLM blends a prefix tree with an LLM-based template extractor. The latter capitalizes on the robustness of LLMs to semantically extract log templates from individual log messages. Concurrently, the prefix tree provides efficient log clustering grounded in syntax. On the one hand, the enhanced accuracy of the LLM template extractor ensures that the prefix tree is meticulously constructed and updated. On the other hand, the prefix tree aids in trimming the computational overhead of LLM by eliminating repetitive LLM calls. We have also integrated an automatic merging mechanism to rectify any template imperfections stemming from LLMs. These elements come together harmoniously to form a synergized parsing framework. Moreover, we exploit the in-context learning (ICL) capabilities of LLMs and implement named entity recognition (NER) prompting to further boost the accuracy of our LLM template extractor.

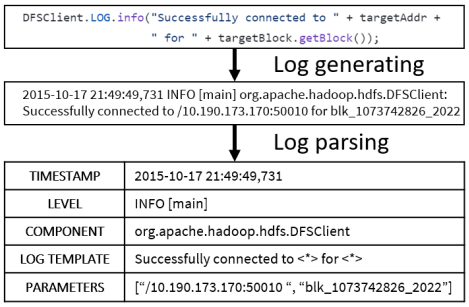

Figure 1: An example of log parsing.

During the evaluation of our method, we encountered an intriguing observation. While our method generally produces satisfactory results, it doesn't always align with the annotated labels from the benchmark. This discrepancy can be attributed to what we term as the *Granularity of Log Parsing*. Both the annotated labels and the model's outputs are logical in their own right. However, these differences in granularity can significantly impact existing metrics, as log entries parsed at differing granular levels are deemed incorrect. To more accurately quantify and understand this granularity discrepancy between parsing results, we introduce the metrics of *Granularity Distance*. We further integrate human interactions in our method to allow users to calibrate the granularity based on their specific needs.

We comprehensively evaluated our approach on the loghub-2k [53] and the extensive logPub [17] datasets provided by the LogPAI team. Remarkably, LogParser-LLM surpasses previous state-of-the-art parsers, achieving a 48.3% and 32.0% increase in the F1 score for grouping and template accuracy, all without the need for domain-specific human effort. Notably, after calibrating granularity with ICL using a mere 32-shot labeled data for each domain, the performance enhancement reaches up to 56.8% and 69.7%. Given that each of the 14 datasets averaged 3.6 million logs, LLMs were only queried an average of 272.5 times, minimizing overhead and showcasing the feasibility of our method's real-world application.

The key contributions of this paper are summarized as follows:

(1) Introduction of *LogParser-LLM* , a novel method leveraging LLMs for log parsing that merges syntactic and semantic insights,

featuring an LLM template extractor and prefix tree to reduce LLM calls while processing millions of log lines efficiently.

(2) Enhancement of template extraction accuracy using ICL and NER prompting, characterized by low requirements for hyper-parameter tuning and labeled data, ensuring broad applicability and quick adaptation to new data.

(3) Development of new metrics for assessing parsing granularity, along with different options for users to adjust granularity effortlessly, making the parsing process more adaptable.

(4) Comprehensive validation of our approach through extensive testing on the *loghub-2k* and *logPub* benchmarks, demonstrating its effectiveness and efficiency, and confirming its suitability for addressing current challenges in log parsing.

2 Related Work and Motivation

Log parsing, extensively explored in research [19, 53], identifies static **templates** and dynamic **parameters** within log entries. As shown in Figure 1, the template "Successfully connected to <*> for <*>" includes dynamic elements like "/10.190.173.170:50010" and "blk_1073742826_2022" as parameters. We categorize log parsing techniques into syntax-based, semantic-based, interactive, and LLM-based methods. We assess their pros and cons and identify opportunities for innovation, particularly in leveraging Large Language Models to improve log parsing capabilities.

2.1 Syntax-based Log Parsers

Syntax-based parsers detect templates by identifying repeating patterns as static and others as parameters. Frequency-based parsers like SLCT [41], LFA [32], LogCluster [42], and Logram [2], build on token recurrence. Similarity-based parsers, including LKE [7], LogSig [39], LogMine [12], SHISO [30], and LenMa [38] cluster logs by similarity. Heuristics-based parsers such as AEL [18], IPLoM [27], Drain [13], Spell [5], Brain [49], and MoLFI [28], apply specific strategies including the longest common subsequence-based approach, iterative partitioning, prefix trees, and evolutionary algorithms for template extraction. These methods are fast and cost-efficient but may miss semantic details and require domain-specific tuning.

2.2 Semantic-based Log Parsers

Semantic parsers have evolved with neural networks like bidirectional LSTM, as seen in Semparser [16] and Uniparser [25], and pre-trained language models such as LogPPT [21]. VALB [22] further enhances the model's semantic understanding by classifying specific parameter categories. These models require labeled data for training and classify tokens into templates or parameters. They offer semantic understanding and can generalize across log types, but also demand resource-intensive training and periodic updates, presenting significant operational challenges.

2.3 Interactive Log Parsing

Recent studies [43, 44] have incorporated user feedback into log parsers, facilitating human-in-the-loop log parsing. This approach not only enables the parser to swiftly adapt to evolving logs but also enhances the accuracy of template mining.

2.4 LLMs-based Log Parsing

Large Language Models (LLMs) have emerged as transformative tools in numerous domains, demonstrating their prowess and versatility. Their pre-training on vast datasets, which include diverse content such as code and log data, makes them particularly adept for specialized tasks like log parsing. Studies like [20, 24, 31, 47] have begun to tap into this potential, primarily focusing on prompt engineering to improve template extraction efficiency. While these advancements highlight the promise of LLMs in log parsing, they predominantly utilize a **line-by-line parsing approach**. This method, although innovative, leads to high computational demands due to LLMs' extensive parameter spaces, making these approaches **impractical for real-world applications** due to the significant computational overhead.

The benefits of LLMs extend beyond their raw computational ability, offering **deep semantic understanding** and the capacity to **generalize across different log formats**, adapting seamlessly to new data types. This adaptability is crucial, as it reduces the need for extensive preprocessing, hyper-parameter tuning, and manual labeling, streamlining the deployment process.

Despite these advantages, the practical deployment of LLMs in log parsing is hindered by their **high operational costs**. Effective utilization requires careful **prompt tuning**, a process that can be as resource-intensive as the computational demands of the models themselves. This challenge underscores the need for more efficient approaches that can leverage the strengths of LLMs without incurring prohibitive costs, ensuring their viability for broader, real-world application.

3 Granularity of Log Parsing

In this section, we delve into the granularity of log parsing. Starting with Section 3.1, we characterize its two primary facets: Specificity and Applicability, elucidating them through an illustrative example. In Section 3.2, we first highlight the shortcomings of existing metrics, emphasizing their inability to capture granularity nuances. Concluding the section, we introduce the granularity distance, a novel metric adept at gauging granularity discrepancies on two distinct levels, effectively addressing the gaps in prior metrics.

3.1 Characterization of Granularity

The granularity of log parsing is pivotal for how the parsing result looks like. We primarily characterized the granularity by two dimensions: specificity and applicability.

3.1.1 Specificity. Specificity in log parsing indicates the depth of detail within a template. It is primarily driven by the **information and content** of templates. The more detailed they are, the higher the specificity.

High Specificity (High Granularity): Such templates have fewer, more detailed variable parts, aligning with a narrower set of logs due to their intricacy.

Low Specificity (Low Granularity): These are more general, with numerous variable components, catering to a broader log range.

The desired level of specificity often varies based on the log analysis context and user needs.

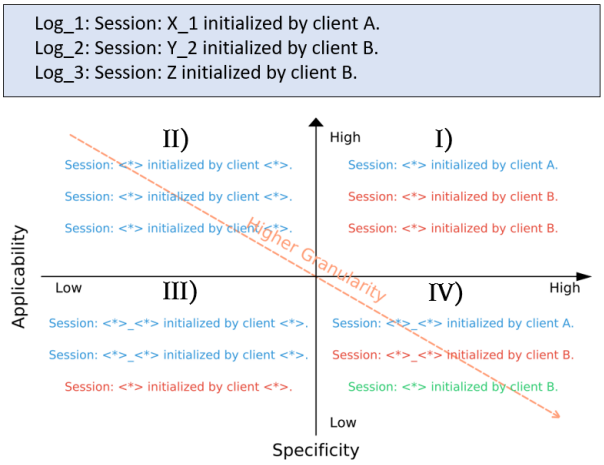

Figure 2: A demonstration of granularity variations in log parsing. Colors denote groups of templates. Applicability is represented on the vertical axis, while Specificity is represented on the horizontal axis.

3.1.2 Applicability. Applicability in log parsing gauges a template's adaptability across varied log entries, primarily based on the **structure** of its placeholders. The more structurally generic they are, the broader their reach, translating to higher applicability.

High Applicability (Low Granularity): Templates here have a wide-reaching, generic structure, suitable for numerous logs.

Low Applicability (High Granularity): These are designed for specific log subsets, with unique structural placeholders.

Both specificity and applicability play crucial roles in determining the outcome of log templates, subsequently affecting metrics that measure grouping and parsing accuracy of log parsing. Together, they delineate the granularity of log parsing. The ideal granularity often finds a midpoint between these two dimensions and is shaped by user preferences and the nuances of individual use cases. Notably, even a minor discrepancy in granularity can result in substantially different groupings, a difference that can be exaggerated when using inappropriate metrics. This highlights the pressing need for a well-conceived metric. It is essential to recognize the inherently subjective nature of granularity. As such, it is inappropriate to strictly label a particular granularity as dominant or to view benchmark dataset labels as definitive standards. Figure 2 illustrates the parsing outcomes at varying granularities, using three representative log messages from the Windows dataset in loghub-2k. Applicability is shown through session names, while specificity is shown through client names. High applicability and low specificity result in a generic structure that matches more log entries, indicating lower granularity. The benchmark uses granularity I) as its labeled ground truth.

3.2 Measuring Granularity Discrepancy

Existing evaluation metrics, while versatile, emphasize either the accuracy of grouping logs or the fidelity in extracting templates and parameters. Both dimensions are indispensable, especially considering their implications for downstream tasks like log anomaly detection. However, these metrics often overlook the subtle granularity differences inherent in log parsing. Existing benchmark datasets [17, 53] are anchored to the annotators' subjective interpretations, suggesting that multiple valid granular interpretations can exist for a single log. Such diversity challenges the conventional wisdom of treating annotated labels as an unequivocal gold standard. Instead of a myopic focus on exact matches, a more encompassing metric that can quantify and understand this granularity discrepancy is imperative.

3.2.1 Existing metrics. We examine four prevalent metrics in this section. The widely recognized message-level metrics, Grouping Accuracy (GA) [53] and Parsing Accuracy (PA) [2], focus on the volume of messages associated with each template, often prioritizing templates with a larger number of log messages. To address this bias, template-level metrics like F1-score of Group Accuracy (FGA) [17] and F1-score of Template Accuracy (FTA) [19] have been introduced, ensuring an equitable evaluation of each template. The detailed definitions can be found in Appendix B.

GA and PA primarily evaluate based on the volume of log messages, making them susceptible to biases from imbalanced templates. In real-world scenarios, less frequent templates, such as error messages, might be of paramount importance. Their misinterpretation could be detrimental, yet this might not be reflected effectively using these metrics. Template-level metrics ensures a holistic evaluation of log parsers, giving equal importance to each template. However, while these metrics minimize biases from frequent templates, they still present challenges. If a token is interpreted differently based on granularity nuances, whether designated as a static part or a parameter, it might result in considerable variances in template counts. Additionally, such metrics don't provide a clear insight into granularity differences.

3.2.2 Granularity Distance (GD). In light of the discussions above and the sensitivity of existing metrics to subtle granularity discrepancies, we introduce the *Granularity Distance* metric. Inspired by the traditional edit distance, this metric calculates *the minimum operations necessary to transform one parsing result into another.* It serves as a quantitative reflection of the least human intervention needed to attain the desired granularity. This metric can be dissected into two main components:

Grouping Granularity Distance(GGD): This aspect emphasizes the grouping of log messages. The aim is to match the expected grouping of log messages without mandating identical templates within those groups.

Parsing Granularity Distance(PGD): This is a more rigorous metric requiring an exact match for each log template. Disparities in the parsed templates increment the distance.

For the operations contributing to this distance:

Operations on GGD: *1) Merge*: Combine groups by changing one static section to variable. *2) Split*: Separate groups by switching one variable to static section.

Operations on PGD: *1) Static to Variable*: Convert a static section of the template to a variable. *2) Variable to Static*: Revert a variable within the template to a static section.

Similar to the edit distance, granularity distance possesses symmetrical properties, meaning the distance from one log template to another is the same as the distance from the second to the first.

Additionally, granularity distance satisfies the properties of non-negativity and identity of indiscernibles, akin to the traditional metrics in distance measurement. This ensures a consistent and logical comparison of log parsing granularity between different parsing results.

It is straightforward to compute GD when logs are accurately tokenized, and each token is categorized as either a parameter or a template. However, such precise labeling and tokenization are often absent. To circumvent this, an approximate version of GGD can be derived by merely tallying the merge and split operations required to transition from one grouping to another.

4 Methodology

In this section, we introduce **LogParser-LLM** tailored to tackle the challenges previously highlighted. Our approach is built upon four key pillars: **1) Enhanced Template Extraction**: Leveraging the prowess of LLMs, we aim to boost the accuracy of template extraction. **2) Efficient LLM Use**: We design an algorithm that harnesses the advanced capabilities of LLMs while optimizing resource consumption. **3) Reduced Human Effort with Broad Applicability**: Our method minimizes human intervention, especially in label annotation and hyper-parameter tuning, yet remains versatile across various domains and log formats. **4) Interactive Feedback Integration**: Our method is integrated with human feedback for parsing granularity calibration. The following sections delve deeper into these principles, elucidating the techniques and decisions underpinning our approach.

4.1 Preprocessing

Our method hinges on minimal preprocessing, using only a basic regular expression to extract log content. While many approaches demand greater domain knowledge, often employing regular expressions to substitute common variables like IP addresses and block IDs [13], we retain the original message, ensuring the LLM grasps the log's full context. Unlike other strategies that use distinct separators for log tokenization [9, 25, 49], we consistently tokenize using spaces. Hence, unless otherwise specified, tokens in following sections are space-separated, capitalizing on the LLM's native tokenizer. This streamlined preprocessing minimizes the need for specialized expertise, yet upholds strong log parsing efficacy.

4.2 Base Algorithm with Prefix Parse Tree

Central to our methodology is a base algorithm employing a prefix parse tree, inspired by the efficiency demonstrated in Drain [13]. This section elaborates on the data structures integral to the algorithm, detailing their design and their roles in addressing the aforementioned principles. Specifically, we'll elucidate how incoming logs are matched with existing clusters during tree traversal, how and when LLMs are invoked for template extraction, and the dynamics of updating the tree with new templates obtained from the LLM extractor.

4.2.1 Data Structures. Three primary data structures form the backbone of our approach: a set of log clusters, a template pool, and a prefix parse tree. Figure 3 offers a visual representation of this organizational structure. The subsequent discussion delineates their respective functionalities:

Log Cluster: A log cluster is a collection of logs with the same template. It keeps track of the individual log IDs and stores a log embedding, created by an LLM encoder, for future use. Each cluster is characterized by its *log template*, extracted via LLM, and possibly multiple *syntax templates* aiding the prefix tree in its traversal and template matching processes. While *syntax templates* correspond directly with the tokens of the raw logs, identifying static and variable parts, the *log templates* from the LLM may represent several tokens with a single placeholder. These *syntax templates* are stored in a dictionary, utilizing token counts as keys and corresponding template lists as values.

Template Pool: The *template pool* establishes a linkage, mapping *log templates* to their respective *log clusters*.

Prefix Parse Tree: In this tree structure, every node—bar the root—symbolizes a token. The wildcard token "<*>" serves as a universal matcher for any token. Crucially, not just the leaf nodes, but virtually all nodes (excluding the root) can possess pointers to log clusters matching the token sequence extending from the root. A unique feature to note is that a single log cluster might be accessible from multiple nodes, courtesy of the potential existence of various syntax template variants for a given log cluster.

4.2.2 cluster matching with tree search. Upon receiving a new log, our first step is tokenization. Tokens are then processed sequentially, with each token checked against nodes in the prefix tree. After matching the initial token, we proceed to the subsequent token, considering only the children of the previously matched node. This progression continues either until all tokens are matched or when no further matching tokens exist. Throughout this traversal, log clusters referenced by the encountered nodes are shortlisted as potential candidates for a thorough match evaluation.

At this juncture, we have pinpointed a subset of log clusters consistent with the rules encoded in the tree path. Our task now is to determine the genuine match from these candidates. Contrary to existing methodologies such as those in [13, 49], which deploy similarity metrics and predefined, dataset-specific thresholds, our approach crystallizes outcomes into three distinct categories: i) Strict match, ii) Loose match, and iii) No match. For each prospective cluster, an initial check compares the token count between the incoming log and the cluster's syntax templates. Discrepant token counts immediately exclude the possibility of a match. Following this, a 'loose match' is attempted, aligning tokens from the syntax template and the log. Here, any token within the syntax template containing the "<*>" wildcard can align with any log token. To illustrate, a token such as "prefetching...<*>" can loosely align with any log entry with a singular token. After achieving a loose match, regular expressions ensure a rigorous alignment with elements outside the "<*>" in the syntax template. A complete token alignment signifies a strict match. It is worth noting that the matching process stops upon achieving a strict match. In scenarios where a strict match is identified, the log is straightforwardly added to the matched cluster. Conversely, in the absence of a strict match, the LLM template extractor is invoked for template extraction, followed by the necessary updates to the data structures.

Our method's precision, rooted in the capabilities of LLMs, eliminates the need for meticulous hyperparameter tuning across log

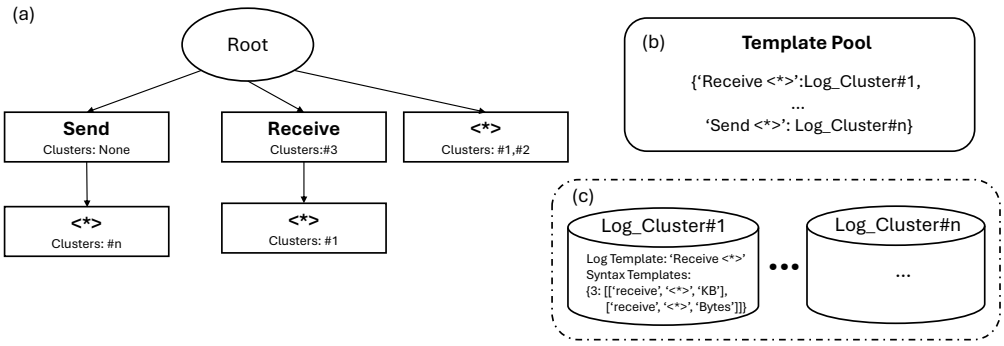

Figure 3: An example demonstrating the data structures in our method: (a) A prefix parse tree with nodes linking to log clusters, (b) a Template Pool mapping log templates to log clusters, and (c) Log Clusters containing collections of logs with the same log template.

sources. Leveraging LLMs' proficiency, which typically yields semantically accurate templates for individual logs, facilitates this stringent matching paradigm. Not only does it simplify the tuning process, but it also optimizes the number of calls to the LLM. Ideally, if we operate under the assumption that LLMs generate templates mirroring the ground truth, the volume of LLM calls is effectively capped at the total number of distinct syntax templates. This count is in the ballpark of the total number of log templates, offering a scalable approach.

4.2.3 Parse tree update. The comprehensive update rule is elucidated in Algorithm 1. As outlined in lines 8-9, for scenarios of either a loose match or no match, the LLM is invoked to derive a log template. If this extracted template already resides in the template pool, it suggests that the current log pertains to an existing cluster but with an alternative syntax template variant. In such cases, the associated cluster can be swiftly identified via the template pool mapping. It then becomes essential to integrate the novel syntax template into the cluster and adjust the tree to accommodate nodes that align with this new syntax template.

Conversely, if the template isn't found in the template pool yet a loose match has been identified, the LLM is once more consulted. Its task here is to determine if the loosely matched cluster can integrate this new log. A positive outcome leads to the generation of a merged template. Subsequently, both the syntax and log templates of the cluster undergo an update, with the merged template being added to the template pool.

If a log, after undergoing the entire aforementioned process, still has not been allocated to an existing cluster, it is indicative of a unique log template. Such instances mandate the creation of a new cluster, with corresponding updates made to the tree.

4.3 Enhancing LLM Template Extraction

While the base algorithm already paves the way for efficient and precise log cluster matching, there remains room to refine the accuracy of the LLM template extractor. To this end, we introduce variable-aware prompting, amalgamating it with in-context learning. This fusion not only amplifies the LLM's task comprehension but also augments its overall performance.

Algorithm 1 LogParser-LLM

```
   Input: logs, root_node    Result: log_clusters, tree
1: log_clusters ← {}
2: template_pool ← {}
3: for log in logs do
4:   matched_clusters ← search(tree, log)
5:   if strict_match then
6:     strict_matched_cluster.add(log)
7:     added ← True
8:   else if loose_match or no_match then
9:     template ← get_llm_template(log, log_clusters)
10:    if template in template_pool then
11:      update_tree(tree, log, template_pool[template])
12:      added ← True
13:    else
14:      for cluster in loose_matched_clusters do
15:        check_merge(log, cluster)
16:        if merge then
17:          cluster.update(merged_template)
18:          cluster.add(log)
19:          template_pool[merged_template] = cluster
20:          break
21:          added ← True
22:        end if
23:      end for
24:    end if
25:  end if
26:  if not added then
27:    new_cluster ← create_cluster(log, template)
28:    update_tree(tree, log, new_cluster)
29:    template_pool[template] = cluster
30:  end if
31: end for
```

Additionally, there exist other straightforward avenues to bolster template extraction capabilities. One could leverage more powerful LLMs available in the ever-evolving landscape of language models.

Alternatively, supervised fine-tuning of an LLM using labeled data presents another viable strategy.

4.3.1 Variable-Aware Prompting. Past research [22] has highlighted the benefits of identifying and classifying specific variables within logs. By categorizing these variables, not only is the accuracy of template extraction enhanced, but it also proves advantageous for subsequent tasks. Drawing inspiration from this research and the concept of chain-of-thought prompting [45], we restructure our prompts. These prompts now serve dual purposes: they identify variables and categorize them into one of the ten classifications as outlined in [22]. This refined approach prompts the model to understand and determine which components should be classified as variables and the reasoning behind such categorization.

4.3.2 In-Context Learning with K-Shot Demonstrations. In-context learning (ICL) has become a favored approach when using LLMs for downstream tasks without the need for finetuning [4]. Typically, ICL-based prompts contain three elements: *Instruction*: A task-specific description. *Demonstrations*: A set of examples, essentially pairs of queries coupled with their ground truth answers. *Query*: The direct question to which the LLM provides a response. Each time the LLM is called upon for template extraction, we draw a sample of $k = 3$ examples from our existing pool of log template pairs. Incorporating the principles of Variable-Aware Prompting, we include ten examples, each representing a distinct type of log parameter, as seed examples. Subsequent template extraction results expand this pool. To obtain these samples, we calculate cosine similarity between LLM embedding of query log and all embeddings present in the example pool. The top-k samples are then chosen as k-shot demonstrations within the prompt.

4.4 Optimal Granularity via Human-in-Loop

Integrating human expertise into the automated log parsing process is key to achieving the right granularity. Human input can be seamlessly incorporated at various stages of the parsing pipeline to enhance accuracy and maintain consistency:

1) Pre-Processing Intervention: Experts annotate a sample of logs before parsing begins. These annotations serve dual purposes: they can be used as seed examples for In-Context Learning (ICL) or to fine-tune LLMs, ensuring the model's output aligns more closely with specific parsing needs.

2) Real-Time Calibration: During the parsing process, human judgment can be applied to guide decisions on template merging, ensuring the parsing maintains the desired level of granularity throughout.

3) Post-Processing Refinement: After parsing, the system identifies potential merges or splits based on semantic similarity or template variability. Experts review these suggestions, making adjustments to achieve the optimal granularity.

In Table 1, we demonstrate how LogParser-LLM-C incorporates pre-processing intervention, enhancing the base LogParser-LLM's capabilities. For real-time calibration, human expertise can be used to refine the merging process in line with desired granularity levels, as outlined in line 15 of Algorithm 1. Post-processing refinement can integrate methods like those suggested in [43] for effective final adjustments, ensuring parsed logs accurately reflect the intended granularity.

5 Experiments

We assess the effectiveness of our method using two datasets: loghub-2k [53] and logPub [17]. First, we detail the experimental settings. Subsequently, we outline the evaluation metrics employed, highlighting a novel metric we introduce to gauge the granularity distance of parsing outcomes. In examining results from the loghub-2k dataset, our primary objective is to elucidate the contribution of each design component of our method. With the logPub benchmark, our intent is to demonstrate both the effectiveness and efficiency of our approach when handling large-scale datasets in practice.

5.1 Experimental Settings

5.1.1 Datasets. Loghub-2k is a widely recognized benchmark in the field of log parsing. It encompasses logs from 16 diverse systems, including distributed systems, supercomputers, operating systems, mobile platforms, server applications, and individual software packages. For every system source, 2,000 log messages are meticulously annotated. Complementing this, LogPub is a more recent, expansive iteration of Loghub-2k. It features 14 systems, with each averaging a substantial 3.6 million log lines, and showcases a pronounced increase in the number of log templates. This dataset offers a realistic, large-scale environment, paving the way for comprehensive evaluations of log parsing methodologies.

5.1.2 Implementation Details. Our experimental setup involves a server powered by Ubuntu 20.04.3 LTS with 512GB of RAM. We use both ChatGPT (version gpt-3.5-turbo-0301) and GPT-4 (version gpt-4-0613) for template extraction. For embedding the logs, the text-embedding-ada-002 method is adopted. All interactions with these models are facilitated through the official OpenAI API. To guarantee consistency in our findings and support reproducibility, we maintain the temperature parameter at 0 to minimize variability. For fine-tuning our LLM, the Llama-2-13b model [40] serves as the foundation. Comprehensive details regarding this fine-tuning process can be found in Appendix B.2. For in-context learning, we uniformly sample 32 log-template pairs from the first 10% of each dataset based on token length as candidate logs. The same samples are employed for fine-tuning.

5.2 Evaluation Metrics

In alignment with prevailing methods outlined in [17, 19, 25], we utilize the GA, PA, FGA, PTA, RTA and FTA metrics delineated in Appendix B for evaluation. Furthermore, we use the Grouping Granularity Distance (GGD) proposed in Section 3.2.2 as a more intuitive metric to gauge the granularity discrepancies in parsing outcomes.

5.3 Evaluation on Loghub-2k

Our primary objective in conducting experiments with the smaller-scale Loghub-2k dataset is to assess the efficacy of our method's key components. Additionally, we employ this dataset as a development set, refining our prompts for LLM template extraction, merging check and verification. The final prompts we adopted are

Table 1: Comparison of various log parser algorithms on large-scale logPub dataset. The best results are in bold.

	Drain						Uniparser						LogPPT						LogParser-LLM						LogParser-LLM-C					
	GA	PA	FGA	FTA	GGD	PGD	GA	PA	FGA	FTA	GGD	PGD	GA	PA	FGA	FTA	GGD	PGD	GA	PA	FGA	FTA	GGD	PGD	GA	PA	FGA	FTA	GGD	PGD
Proxifier	69.2	68.8	20.6	17.6	4	14	50.9	63.4	28.6	45.7	5	10	**98.9**	**100.0**	**87.0**	**95.7**	**1**	**1**	51.0	63.4	40.0	53.3	5	9	**98.9**	**100.0**	**87.0**	**95.7**	**1**	**1**
Linux	**68.6**	11.1	77.8	25.9	30	432	28.5	16.4	45.1	23.2	108	274	20.5	16.8	71.2	42.8	29	104	27.0	16.3	80.1	46.6	18	81	53.4	49.4	**91.1**	**74.0**	**10**	**68**
Apache	**100.0**	72.7	100.0	51.7	**0**	21	94.8	94.2	68.7	26.9	11	31	78.6	94.8	60.5	36.8	6	23	**100.0**	85.7	**100.0**	65.5	**0**	8	**100.0**	**99.5**	**100.0**	**82.8**	**0**	**5**
Zookeeper	99.4	84.3	90.4	61.4	2	30	98.8	**98.8**	66.1	51.0	14	31	96.7	84.5	91.8	80.9	4	10	98.8	81.9	86.2	72.4	2	19	**99.5**	96.8	**92.9**	**85.7**	**1**	**13**
Hadoop	92.1	54.1	78.5	38.4	18	210	69.1	88.9	62.8	47.6	38	119	48.3	66.6	52.6	43.4	46	81	93.8	67.6	87.3	55.0	14	108	**94.5**	**90.6**	**88.9**	**81.0**	**11**	**41**
HealthApp	86.2	31.2	1.0	0.4	11	138	46.1	81.7	74.5	46.2	16	60	99.8	**99.7**	94.7	82.2	4	8	99.8	58.2	95.6	81.8	4	5	**100.0**	98.2	**96.5**	**89.0**	**3**	**7**
OpenStack	75.2	2.9	0.7	0.2	6618	23**	**100.0**	51.6	96.9	28.9	1	7	53.4	40.6	87.4	73.8	4	4	**100.0**	49.6	**100.0**	79.2	**0**	11	**100.0**	**100.0**	**100.0**	**97.9**	**0**	**1**
HPC	79.3	72.1	30.9	15.2	10	178	77.7	94.1	66.0	35.1	10	58	78.2	99.7	78.0	76.8	12	31	**86.4**	94.2	76.0	72.6	**6**	180	**86.4**	**99.8**	**76.8**	**74.6**	**6**	**29**
Mac	76.1	35.7	22.9	6.9	102	1347	73.7	68.8	69.9	28.3	73	624	54.4	39.0	49.3	27.4	177	489	89.7	30.3	84.7	36.2	42	444	**91.5**	**76.4**	**86.4**	**60.6**	**33**	**297**
OpenSSH	70.7	58.6	87.2	48.7	3	33	27.5	28.9	0.9	0.5	15	26	27.7	65.4	8.1	10.5	17	26	**78.0**	69.0	**96.1**	88.3	**1**	9	**78.0**	**100.0**	**96.1**	**98.7**	**1**	**2**
Spark	88.8	39.4	86.1	41.2	18	239	85.4	79.5	2.0	1.2	62	186	47.6	95.2	37.4	29.9	75	221	**97.6**	80.2	85.2	46.3	16	148	**97.6**	**99.7**	**88.2**	**68.1**	**11**	**101**
Thunderbird	83.1	21.6	23.7	7.1	137	2043	57.9	65.4	68.2	29.0	194	976	56.4	40.1	21.6	11.7	282	1012	73	57.1	80.0	56.0	104	662	**67.5**	**64.3**	**83.1**	**59.3**	**88**	**615**
BGL	91.9	40.7	62.4	19.3	48	434	91.8	94.9	62.4	21.9	43	209	24.5	93.8	25.3	26.1	69	164	**93.8**	81.0	78.9	50.0	34	154	88.9	**97.6**	**84.0**	**71.6**	**24**	**85**
HDFS	99.9	62.1	93.5	60.9	2	6	**100.0**	94.8	96.8	58.1	1	1	72.1	94.3	39.1	31.2	18	59	**100.0**	94.8	74.7	57.8	5	26	**100.0**	**100.0**	**96.8**	**96.8**	**1**	**1**
Average	84.3	46.8	55.4	28.2	500.2	394.2	71.6	73.0	57.8	31.7	42.2	186.6	61.2	73.6	57.4	47.8	53.1	159.5	**90.9**	68.3	85.7	63.1	17.9	133.1	89.7	**90.9**	**90.6**	**81.1**	**13.6**	**90.4**

Table 2: Efficiency and effectiveness of the LogParser-LLM with different LLMs. The best results are in bold.

	Avg. # of LLM Calls	Avg. Time(s)			Avg. Metrics					
		Per Infer.	Base	Total	GA	PA	FGA	FTA	GGD	PGD
LogParser-LLM w/ GPT-3.5-turbo	566.4	0.52	522.88	817.33	82.0	64.5	81.6	58.3	26.9	198.9
LogParser-LLM-C w/ GPT-3.5-turbo (32shot)	289.7	0.52	461.67	**612.31**	**91.3**	90.5	90.2	77.5	15.4	120.7
LogParser-LLM with GPT-4	427.2	4.18	452.15	2237.85	90.9	68.3	85.7	63.1	17.6	133.1
LogParser-LLM-C w/ GPT-4 (32shot)	**272.5**	4.18	433.22	1572.27	89.7	**90.9**	**90.6**	**81.1**	**13.6**	**90.4**
LogParser-LLM w/ fine-tuned Llama-2-13b(32shot)	6620.3	2.54	621.33	2272.63	78.5	72.0	66.8	49.9	45.9	194.0

presented in Appendix. B. The most effective configurations determined through Loghub-2k are subsequently applied unchanged to the LogPub dataset for evaluation.

Table 3: Comparison with existing LLM-based method on Loghub-2k.

	# of labeled logs	GA	PA	PTA	RTA
Eval of chatgpt[20]	0	72.1	54.3	/	/
	4	76.1	79.0	/	/
DivLog[47]	200	92.8	98.1	92.0	92.9
LogParser-LLM	0	91.8	69.9	67.8	66.8
	4	92.1	74.7	72.6	74.9
	32	94.8	90.5	84.9	85.3
	200	95.9	98.0	96.8	96.9

Comparison with Existing LLM-based Parsers Existing LLM-based parsers, which process logs line-by-line, are impractical for evaluation on the expansive LogPub dataset due to the immense number of LLM calls required. We therefore use the Loghub-2k dataset for comparison, but advise caution in interpreting these results because of the dataset's limited scope and the possibility that a few well-chosen labeled samples might cover the majority of templates. In Table 3's results, our method either matches or exceeds the performance of existing approaches with an equivalent number of labeled logs, highlighting our method's effective use of LLMs for log parsing despite the constraints.

Accuracy Evaluation We compare our model to three state-of-the-art methods: two syntax-based methods, Drain [13] and Brain [49], and one semantic-based method, LogPPT [21]. As indicated in Table 4, the previous state-of-the-art methods achieved higher GA and PA values because they were meticulously tuned with hyperparameters on each dataset to optimize these metrics. However, these values alone do not necessarily indicate superior performance. When evaluating with template-level metrics such as FGA and FTA, as well as our proposed GGD, our model outperforms them without the need for any domain-specific configuration.

Ablation The results of our ablation study for different components, including in-context learning (ICL), variable-aware prompt (VA), and automatic template merge (Merge), are presented in Table 4. The numbers clearly demonstrate that each proposed component positively impacts the method's performance, as evidenced by the reduction in GGD. Notably, the most significant performance boost comes from the transition from GPT-3.5 to GPT-4. Furthermore, the enhancements from other components are even more pronounced with GPT-4, underscoring the potency of more powerful LLMs. Using GPT-4 on its own, even without ICL, yields impressive results, showcasing its capacity to adhere to specific instructions and complete tasks in a zero-shot scenario. However, it is important to note that integrating these components also increases the associated costs when invoking the LLM.

Table 4: Ablation studies of LogParser-LLM on Loghub-2k.

	GA	PA	FGA	FTA	GGD
Drain	87.2	40.0	75.1	34.4	9.00
Brain	**96.6**	40.4	90.8	42.7	3.35
LogPPT	92.3	**86.5**	89.2	69.5	6.25
GPT-3.5	91.5	68.4	86.0	64.7	5.88
GPT-3.5+ICL+VA	89.8	67.2	86.1	64.8	5.81
GPT-3.5+ICL+VA+Merge	90.1	61.7	86.9	59.7	5.50
GPT-4	92.5	75.6	91.6	**75.7**	3.69
GPT-4+ICL+VA+Merge	91.8	78.5	**92.2**	67.2	**2.88**

5.4 Evaluation on LogPub

Accuracy and Generalizability Results from the expansive logPub dataset are shown in Table 1. We use LogParser-LLM-C to denote calibrated variants of our method. It is clear that our model, LogParser-LLM , even without granularity calibration, significantly surpasses all baseline methods in GA, FGA, and PTA, marking improvements of 7.8%, 48.3%, and 32.0% compared to the best baseline results. However, PA performance lags, mostly due to granularity nuances complicating the LLM's ability to generate templates that perfectly match annotated labels. A standout point is the consistent performance of our method across the 14 datasets, achieved without domain-specific tweaks, maintaining uniform settings throughout. Upon introducing domain-specific granularity calibration with ICL in LogParser-LLM-C, there is a noticeable boost, especially in template parsing metrics such as PA and FTA. This highlights the reduced discrepancy in the applicability of log parsing achieved through ICL.

Granularity Discrepancy Evaluation Both Grouping Granularity Distance (GGD) and Parsing Granularity Distance (PGD) are calculated and shown in Table 1 . PGD is computed using spaces as delimiters for tokenization, representing a lower bound since precise tokenization isn't feasible for such large datasets. This approximation remains valuable for consistent cross-method comparisons.

Unlike message-level GA and PA metrics, which depend on log volume, the proposed metrics avoid template imbalance and provide a clearer performance indicator. For example, in the Linux dataset, Drain's GA is 68.6 compared to our 53.4. However, Drain's GGD is 30 versus our 10, indicating significantly more effort needed to align Drain's results with the ground truth.

Compared to template-level metrics, GGD and PGD show that smaller GD correlates with higher FGA and FTA. However, FGA and FTA can overly penalize repetitive differences. For example, if a ground truth template "instance: <*>" has many instance IDs not correctly identified as variables, it increases the number of identified templates, skewing precision calculations. GGD and PGD count such differences only once, offering a fairer measurement. For instance, GA and PA for Uniparser on OpenSSH are 0.9 and 0.5, respectively—values that indicate a significant gap compared to other methods and are not informative. Conversely, GGD and PGD for Uniparser on OpenSSH are 15 and 26, respectively, providing an informative and intuitive comparison. This robustness is also observed in HealthApp, OpenStack, and Thunderbird datasets.

Evaluation with Different LLMs By design, our framework is versatile enough to be compatible with any language model that can process individual log messages and accordingly generate log templates. This evaluation's primary objective is to assess the impact of different LLMs on the efficacy and efficiency of our approach.

Our results, as presented in Table 2, demonstrate that using GPT-4 as the LLM template extractor paired with ICL yields optimal performance. However, this comes at the cost of increased computational time due to GPT-4's extensive parameter count. Notably, both granularity calibration and ICL enhance performance and concurrently decrease the number of required LLM calls. This is congruent with our framework's foundational assumption that LLMs can generate nearly perfect log templates. For the fine-tuned LLM, we commenced with the widely-recognized open-source LLM, Llama-2-13b. Despite its performance not being optimal, it remains competitive, closely paralleling the results of prior state-of-the-art semantic-based models like Uniparser and LogPPT. This subpar performance may be attributed to our not having meticulously curated the fine-tuning dataset and its limited size. While a more thoughtfully curated, expansive training dataset and hyperparameter tuning could enhance its performance, this conflicts with our intent: to construct a robust log parser that necessitates minimal human intervention and domain-specific knowledge.

For runtime efficiency, we delineate runtime into two facets: the time required for LLM calls and the time for our log parser's other operations. This distinction is crucial, given that OpenAI service calls depend on service availability and are subject to rate limitations, which complicates consistent performance evaluation. To estimate cumulative processing time, we multiply the average response time (in optimal scenarios) by the total number of LLM calls. Our base algorithm's average runtimes stand at 461.67s and 433.22s. If we disregard potential rate caps, the mean response times are 0.52s for GPT-3.5 and 4.18s for GPT-4. With an average of 272.5 LLM calls, this equates to overall runtimes of 612.31s for GPT-3.5 and 1572.27s for GPT-4. For comparison, Drain, one of the fastest existing methods, averages 483.2s to process about 3.6 million logs. In contrast, a conventional line-by-line LLM parsing approach would necessitate 3.6 million LLM calls, leading to an impractical runtime of approximately 22 days for GPT-3.5, not accounting for other operational overheads. This analysis not only emphasizes our method's competitive efficiency but also its practicality, overcoming the inherent impracticality of existing LLM-based parsers by significantly reducing the number of necessary LLM calls.

6 Conclusion

In this study, we introduce LogParser-LLM, a novel approach to log parsing that seamlessly integrates the strengths of Large Language Models (LLMs). Centralizing around a prefix tree and an LLM-based template extractor, LogParser-LLM not only streamlines the extraction of semantically rich log templates but also ensures efficiency through strategic LLM call reductions. While demonstrating compelling results, we also uncovered nuances in parsing granularity, prompting the creation of the Granularity Distance metric. Our rigorous tests on benchmark datasets reveal that LogParser-LLM significantly outshines existing parsers in accuracy and efficiency, demonstrating its potential as a valuable tool for both researchers and practitioners in the field of log analysis.

Acknowledgments

This work was supported by Alibaba Group through Alibaba Research Intern Program.

References

[1] Mike Chen, Alice X Zheng, Jim Lloyd, Michael I Jordan, and Eric Brewer. 2004. Failure diagnosis using decision trees. In *International Conference on Autonomic*

**The reason Drain has a GGD of 6618 but a PGD of 23 is that its preprocessing converts all numbers to the variable "<*>". Complicated instance IDs such as "3edec1e4-9678-4a3a" are preprocessed to"<*>edec<*>e<*>-<*>-<*>a<*>a". This results in a significant number of redundant log clusters, leading to a high GGD. However, when calculating the PGD, this is considered a single variable token which is correctly parsed and thus does not contribute to the PGD.

Computing, 2004. Proceedings. IEEE, 36–43.

[2] Hetong Dai, Heng Li, Che-Shao Chen, Weiyi Shang, and Tse-Hsun Chen. 2020. Logram: Efficient Log Parsing Using n n-Gram Dictionaries. *IEEE Transactions on Software Engineering* 48, 3 (2020), 879–892.

[3] Yingnong Dang, Qingwei Lin, and Peng Huang. 2019. Aiops: real-world challenges and research innovations. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 4–5.

[4] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, and Zhifang Sui. 2022. A survey for in-context learning. *arXiv preprint arXiv:2301.00234* (2022).

[5] Min Du and Feifei Li. 2016. Spell: Streaming parsing of system event logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 859–864.

[6] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 1285–1298.

[7] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. 2009. Execution anomaly detection in distributed systems through unstructured log analysis. In *2009 ninth IEEE international conference on data mining*. IEEE, 149–158.

[8] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. 2009. Execution anomaly detection in distributed systems through unstructured log analysis. In *2009 ninth IEEE international conference on data mining*. IEEE, 149–158.

[9] Ying Fu, Meng Yan, Jian Xu, Jianguo Li, Zhongxin Liu, Xiaohong Zhang, and Dan Yang. 2022. Investigating and improving log parsing in practice. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1566–1577.

[10] Justine Gangneux. 2019. Rethinking social media for qualitative research: The use of Facebook Activity Logs and Search History in interview settings. *The Sociological Review* 67, 6 (2019), 1249–1264.

[11] Nentawe Gurumdimma, Arshad Jhumka, Maria Liakata, Edward Chuah, and James Browne. 2015. Towards detecting patterns in failure logs of large-scale distributed systems. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*. IEEE, 1052–1061.

[12] Hossein Hamooni, Biplob Debnath, Jianwu Xu, Hui Zhang, Guofei Jiang, and Abdullah Mueen. 2016. Logmine: Fast pattern recognition for log analytics. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. 1573–1582.

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

[14] Shilin He, Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, Michael R Lyu, and Dongmei Zhang. 2018. Identifying impactful service system problems via log analysis. In *Proceedings of the 2018 26th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*. 60–70.

[15] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685* (2021).

[16] Yintong Huo, Yuxin Su, Cheryl Lee, and Michael R Lyu. 2023. SemParser: A Semantic Parser for Log Analytics. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 881–893.

[17] Zhihan Jiang, Jinyang Liu, Junjie Huang, Yichen Li, Yintong Huo, Jiazhen Gu, Zhuangbin Chen, Jieming Zhu, and Michael R Lyu. 2023. A Large-scale Benchmark for Log Parsing. *arXiv preprint arXiv:2308.10828* (2023).

[18] Zhen Ming Jiang, Ahmed E Hassan, Parminder Flora, and Gilbert Hamann. 2008. Abstracting execution logs to execution events for enterprise applications (short paper). In *2008 The Eighth International Conference on Quality Software*. IEEE, 181–186.

[19] Zanis Ali Khan, Donghwan Shin, Domenico Bianculli, and Lionel Briand. 2022. Guidelines for assessing the accuracy of log message template identification techniques. In *Proceedings of the 44th International Conference on Software Engineering*. 1095–1106.

[20] Van-Hoang Le and Hongyu Zhang. 2023. An Evaluation of Log Parsing with ChatGPT. *arXiv preprint arXiv:2306.01590* (2023).

[21] Van-Hoang Le and Hongyu Zhang. 2023. Log Parsing with Prompt-based Few-shot Learning. *arXiv preprint arXiv:2302.07435* (2023).

[22] Zhenhao Li, Chuan Luo, Tse-Hsun Chen, Weiyi Shang, Shilin He, Qingwei Lin, and Dongmei Zhang. 2023. Did We Miss Something Important? Studying and Exploring Variable-Aware Log Abstraction. *arXiv preprint arXiv:2304.11391* (2023).

[23] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. 2016. Log clustering based problem identification for online service systems. In *Proceedings of the 38th International Conference on Software Engineering Companion*. 102–111.

[24] Yilun Liu, Shimin Tao, Weibin Meng, Jingyu Wang, Wenbing Ma, Yanqing Zhao, Yuhang Chen, Hao Yang, Yanfei Jiang, and Xun Chen. 2023. LogPrompt: Prompt Engineering Towards Zero-Shot and Interpretable Log Analysis. *arXiv preprint arXiv:2308.07610* (2023).

[25] Yudong Liu, Xu Zhang, Shilin He, Hongyu Zhang, Liqun Li, Yu Kang, Yong Xu, Minghua Ma, Qingwei Lin, Yingnong Dang, et al. 2022. Uniparser: A unified log parser for heterogeneous log data. In *Proceedings of the ACM Web Conference 2022*. 1893–1901.

[26] Chuan Luo, Pu Zhao, Bo Qiao, Youjiang Wu, Hongyu Zhang, Wei Wu, Weihai Lu, Yingnong Dang, Saravanakumar Rajmohan, Qingwei Lin, et al. 2021. NTAM: Neighborhood-temporal attention model for disk failure prediction in cloud platforms. In *Proceedings of the Web Conference 2021*. 1181–1191.

[27] Adetokunbo AO Makanju, A Nur Zincir-Heywood, and Evangelos E Milios. 2009. Clustering event logs using iterative partitioning. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1255–1264.

[28] Salma Messaoudi, Annibale Panichella, Domenico Bianculli, Lionel Briand, and Raimondas Sasnauskas. 2018. A search-based approach for accurate identification of log message formats. In *Proceedings of the 26th Conference on Program Comprehension*. 167–177.

[29] Meta. 2023. *Meta Reports First Quarter 2023 Results.* https://investor.fb.com/investor-news/press-release-details/2023/Meta-Reports-First-Quarter-2023-Results/default.aspx

[30] Masayoshi Mizutani. 2013. Incremental mining of system log format. In *2013 IEEE International Conference on Services Computing*. IEEE, 595–602.

[31] Priyanka Mudgal and Rita Wouhaybi. 2023. An Assessment of ChatGPT on Log Data. *arXiv preprint arXiv:2309.07938* (2023).

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

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

[34] Xiang Rao, Huaimin Wang, Dianxi Shi, Zhenbang Chen, Hua Cai, Qi Zhou, and Tingtao Sun. 2011. Identifying faults in large-scale distributed systems by filtering noisy error logs. In *2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 140–145.

[35] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3505–3506.

[36] Daan Schipper, Maurício Aniche, and Arie van Deursen. 2019. Tracing back log data to its log statement: from research to practice. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 545–549.

[37] Weiyi Shang. 2012. Bridging the divide between software developers and operators using logs. In *2012 34th international conference on software engineering (ICSE)*. IEEE, 1583–1586.

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

[39] Liang Tang, Tao Li, and Chang-Shing Perng. 2011. LogSig: Generating system events from raw textual logs. In *Proceedings of the 20th ACM international conference on Information and knowledge management*. 785–794.

[40] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[41] Risto Vaarandi. 2003. A data clustering algorithm for mining patterns from event logs. In *Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM 2003)(IEEE Cat. No. 03EX764)*. Ieee, 119–126.

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

[43] Liming Wang, Hong Xie, Ye Li, Jian Tan, and John Lui. 2023. Interactive Log Parsing via Light-weight User Feedbacks. *arXiv preprint arXiv:2301.12225* (2023).

[44] Xuheng Wang, Xu Zhang, Liqun Li, Shilin He, Hongyu Zhang, Yudong Liu, Lingling Zheng, Yu Kang, Qingwei Lin, Yingnong Dang, et al. 2022. SPINE: a scalable log parser with feedback guidance. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1198–1208.

[45] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems* 35 (2022), 24824–24837.

[46] Junjielong Xu, Ruichun Yang, Yintong Huo, Chengyu Zhang, and Pinjia He. 2023. Prompting for Automatic Log Template Extraction. *arXiv preprint arXiv:2307.09950* (2023).

[47] Junjielong Xu, Ruichun Yang, Yintong Huo, Chengyu Zhang, and Pinjia He. 2024. DivLog: Log Parsing with Prompt Enhanced In-Context Learning. In *2024 46th international conference on software engineering (ICSE)*. IEEE.

[48] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. 2009. Detecting large-scale system problems by mining console logs. In *Proceedings of*

the ACM SIGOPS 22nd symposium on Operating systems principles. 117–132.
[49] Siyu Yu, Pinjia He, Ningjiang Chen, and Yifan Wu. 2023. Brain: Log Parsing with Bidirectional Parallel Tree. *IEEE Transactions on Services Computing* (2023).
[50] Shenglin Zhang, Ying Liu, Weibin Meng, Zhiling Luo, Jiahao Bu, Sen Yang, Peixian Liang, Dan Pei, Jun Xu, Yuzhi Zhang, et al. 2018. Prefix: Switch failure prediction in datacenter networks. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 2, 1 (2018), 1–29.
[51] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, et al. 2019. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 807–817.
[52] Xu Zhang, Yong Xu, Si Qin, Shilin He, Bo Qiao, Ze Li, Hongyu Zhang, Xukun Li, Yingnong Dang, Qingwei Lin, et al. 2021. Onion: identifying incident-indicating logs for cloud systems. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1253–1263.
[53] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R Lyu. 2019. Tools and benchmarks for automated log parsing. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 121–130.

A Additional Discussion

A.1 Challenges of Log Parsing in Practice

The challenges associated with log parsing encompass several key aspects.

Huge Volume. Modern systems generate vast amounts of log data, which are difficult to manage, store, and analyze. For instance, services like Amazon, Alibaba, and Facebook generate billions of visits per day, each creating multiple log entries [10, 29]. Log parsing, along with tasks like anomaly detection and root cause analysis, is crucial for minimizing system downtime and financial loss [8, 11, 34]. The requirement for real-time, streaming log parsing makes handling such vast volumes challenging.

Constantly Evolving. Systems and technologies continuously evolve, leading to changes in log entry types, formats, structures, and content. New features and components introduce novel log formats, necessitating updates to log templates for accurate parsing. Without timely template updates, parsing algorithms may fail to extract relevant information, leading to inaccuracies and incomplete analysis. Proactively updating log templates ensures effective parsing and adaptation to dynamic log generation.

Diverse Sources. Logs from different systems often have diverse formats, posing a challenge for log parsing algorithms. Each system's unique log format can vary significantly in structure, syntax, and content. Effective log parsing algorithms must generalize to handle various formats without relying on system-specific rules or assumptions.

A.2 Insights and Opportunities of Log Parsing with LLMs

The relentless growth in log volumes, the ever-evolving nature of logs, and the vast diversity in log sources have presented daunting challenges in the realm of log parsing. Syntax-based parsers, while efficient, often grapple with the dynamic nuances introduced by log evolution and diverse sources. LLMs, with their deep semantic understanding and adaptability, are poised as a promising solution but need prompt tuning and optimization to handle vast volumes.

Moreover, the vital role of log data in modern systems underscores the need for log parsing tools that embody certain foundational principles. In practice, a log parser must be **Accurate**, ensuring accurate interpretation of every piece of information. **Efficiency** is paramount to handling the voluminous log data churned out by popular platforms and sprawling systems. The parser's **Evolvability** will be its asset, granting it the flexibility to keep pace with system updates and new feature integrations. To confront the multifarious log formats from diverse sources, it is imperative that a parser is **Generalizable**, ensuring it doesn't rely too heavily on system-specific constructs.

Furthermore, for real-time responsiveness, the parser needs to operate in an **Online** manner. This demands the tool's agility to adapt and recalibrate as new log entries stream in. Addressing the challenge of different granularities in log parsing is also of utmost importance. Ensuring the capability to **Calibrate Granularity** provides flexibility in parsing logs, given that a single log can be interpreted in multiple, yet reasonable, ways based on granularity.

Given the unique strengths and challenges of each approach, a compelling motivation emerges: to amalgamate the adaptability and depth of LLMs with the efficiency intrinsic to syntax-based parsers. This convergence promises a robust and versatile log parsing solution, aptly suited to address both present and future challenges in log management.

B Existing metrics

Grouping Accuracy (GA) GA measures the ratio of correctly grouped log messages. A message is considered correctly grouped if and only if its template group is exactly aligned with ground truth grouping.

Parsing Accuracy (PA) PA assesses the ability to extract templates accurately, critical for tasks like anomaly detection. it is the fraction of messages parsed correctly, meaning all template and variable tokens are identified accurately.

F1 score of Grouping Accuracy (FGA) FGA is a template-level metric that evaluates the fraction of correctly grouped templates. Using the true number of templates (N_g), parsed templates (N_p), and correctly parsed templates (N_c), we calculate the Precision ($PGA = \frac{N_c}{N_p}$) and Recall ($RGA = \frac{N_c}{N_g}$) of Grouping Accuracy. FGA is their harmonic mean.

F1 score of Template Accuracy (FTA) FTA is the harmonic mean of **Recall of Template Accuracy (RTA)** and **Precision of Template Accuracy (PTA)**. Like FGA, FTA evaluates correct template identification at the template level. A template is correct if log messages with the same parsed template share the same ground-truth template and the parsed template matches the ground-truth template exactly. Using $\hat{N}_c$ to denote the number of templates identified accurately by a parser, PTA is then given by $\frac{\hat{N}_c}{N_p}$, and RTA by $\frac{\hat{N}_c}{N_g}$, allowing us to compute FTA as $2 \times \frac{PTA \times RTA}{PTA + RTA}$.

C Additional Implementation Details

Fine-tuning Settings The llmama-2-13b model was finetuned on a server equipped with 8 Tesla A100 80GB GPUs using the Hugging Face Transformers package. The model was finetuned for 50 epochs with 32 samples for each dataset. During inference, we utilized DeepSpeed[35] with 8-bit quantization to expedite the inference process on a single Tesla A100 80GB GPU. Additionally, the model was fine-tuned using LoRA [15] with rank r set to 64. For

```
"""Below is an instruction that describes a task. Write
a response that appropriately completes the request
### Instruction:
Analyze the input log and identify dynamic variables.
Substitute dynamic variables with <XXX>.
### Input:
{log}
### Response:
{template}
### End """
```

Figure 5: Prompt for fine-tuning

```
"""
{task prompt in Figure 4}

Given the following logs, output the parse result for
each of them first, then determine whether they are
instances from the same event template. The output
should use the following format:

EventTemplate_1: {parse result for Log_1}
EventTemplate_2: {parse result for Log_2}
...
EventTemplate_N: {parse result for Log_N}

Reason: {brief reason whether they should be unified}

Answer: {"Yes" or "No"}

Unified Template: {one unified template if yes. Make
sure there are static parts in the template. "None" if
the anwser is no}
"""
```

Figure 6: Prompt for Merge Verification

```
""" {task prompt in Figure 4}
Does the template: "{merged_template}" apply to the
following logs? Please answer with yes or no.

Answer:
"""
```

Figure 7: Prompt for Merge Checking

optimization, we employed the AdamW optimizer with an initial learning rate of 2e-4, which was linearly scheduled down to 0. The batch size was 16.

Prompts We demonstrate the final prompt used for the ICL-based method in Figure 4 and the fine-tuning-based method in Figure 5. The prompts for automatic template merge check and verification are shown in Figure 6 and Figure 7, respectively.

```
"""As a log parser, your task is to analyze logs
and identify dynamic variables. These variables
are distinct from static parts, which are hardcoded
sections in the logging code. The categories of dynamic
variables are concluded as:

Object ID (OID): Includes variables like session IDs
and user IDs.
Location Indicator (LOI): Path information, URIs, and
IP addresses.
Object Name (OBN): Domain names, task names, job names.
Type Indicator (TID): Category for type indicators.
Switch Indicator (SID): Category for switch indicators
(only numerical ones).
Time/Duration of an Action (TDA): Timespan or duration
of actions.
Computing Resources (CRS): Memory, disk space, number
of bytes.
Object Amount (OBA): Number of errors, nodes, etc.
Status Code (STC): Error codes (only numerical ones).
Other Parameters (OTP): All other types of variables.

To parse the logs, substitute dynamic variables with
their respective category tokens, denoted by <XXX>.
Everything outside the <XXX> should remain exactly
unchanged! Do not fix any typo! If a variable comprises
several smaller, fine-grained variables, don't dissect
it. Instead, replace the entire compound variable with
a single <XXX> token. Do not substitute all content in
the log as a variable; only genuine dynamic variables
should be replaced.

Examples:
Log: {example log message}
Parsed Log: {example template}
...
Log: {log to be parsed}
Parsed Log: """
```

Figure 4: Variable-aware Prompt for Log Parsing