2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)

Unleashing the True Potential of Semantic-based Log Parsing with Pre-trained Language Models

Van-Hoang Le*, Yi Xiao†, and Hongyu Zhang†¶,
*The University of Newcastle, Australia
†Chongqing University, China
Email: hoang.le@newcastle.edu.au, yixiao@cqu.edu.cn, hyzhang@cqu.edu.cn

***Abstract*—Software-intensive systems often produce console logs for troubleshooting purposes. Log parsing, which aims at parsing a log message into a specific log template, typically serves as the first step toward automated log analytics. To better comprehend the semantic information of log messages, many semantic-based log parsers have been proposed. These log parsers fine-tune a small pre-trained language model (PLM) such as RoBERTa on a few labelled log samples. With the increasing popularity of large language models (LLMs), some recent studies also propose to leverage LLMs such as ChatGPT through in-context learning for automated log parsing and obtain better results than previous semantic-based log parsers with small PLMs. In this paper, we show that semantic-based log parsers with small PLMs can actually achieve better or comparable performance to state-of-the-art LLM-based log parsing models while being more efficient and cost-effective. We propose UNLEASH, a novel semantic-based log parsing approach, which incorporates three enhancement methods to boost the performance of PLMs for log parsing, including (1) an entropy-based ranking method to select the most informative log samples; (2) a contrastive learning method to enhance the fine-tuning process; and (3) an inference optimization method to improve the log parsing performance. We evaluate UNLEASH on a set of large-scale, public log datasets and the experimental results show that UNLEASH is effective and efficient compared to state-of-the-art log parsers.**

***Index Terms*—log parsing, log analytics, pre-trained LMs**

I. INTRODUCTION

To manage software reliability, software-intensive systems often record runtime information by printing console logs. Software logs are semi-structured data printed by logging statements (e.g., `printf()`, `logInfo()`) in source code. The primary purpose of system logs is to record system states and significant events at various critical points to help engineers better understand system behaviours and diagnose problems. The rich information included in log data enables a variety of log analytics tasks, such as ensuring application security [1], [2], identifying performance anomalies [3], [4], and diagnosing errors [5], [6].

To facilitate downstream tasks, an important first step of log analytics is log parsing, which parses raw log messages into a structured format [7], [8]. The structured log data from log parsing are fed to various machine learning or deep learning models to perform many downstream analysis tasks. Log parsing is a process to extract the static log template parts (or keywords) and the corresponding dynamic parameters (or variables) from free-text raw log messages. For example, in the first log message in Figure 1, the header (i.e., "`17/08/22 15:50:46`", "`ERROR`", and "`DAGScheduler`") can be easily distinguished through regular expressions. The log message consists of a template, "`Stage <*> was cancelled`" with three keywords (i.e., "`Stage`", "`was`", and "`cancelled`"), and the parameter of "`1`".

Log Message

```
17/08/22 15:50:46 ERROR DAGScheduler Stage 1 was cancelled
17/08/22 15:50:55 DEBUG ApplicationMaster Deleting staging directory .sparkStaging/application_14
17/08/22 15:51:02 INFO HookManager Deleting directory /opt/hdfs/nodemanager/
17/08/22 15:51:24 DEBUG BlockManager Putting block rdd_2_2 with replication took 0
17/08/22 15:52:36 ERROR DAGScheduler Stage 2 was cancelled
```

Structured Logs

Datetime	Level	Component	Log Template	Variables
17/08/22 15:50:46	ERROR	DAGScheduler	Stage <*> was cancelled	1
17/08/22 15:50:55	DEBUG	ApplicationMaster	Deleting staging directory <*>	.sparkStaging/application_14
17/08/22 15:51:02	INFO	HookManager	Deleting directory <*>	/opt/hdfs/nodemanager/
17/08/22 15:51:24	DEBUG	BlockManager	Putting block <*> with replication took <*>	rdd_2_2, 0
17/08/22 15:52:36	ERROR	DAGScheduler	Stage <*> was cancelled	2

Fig. 1. An example of log parsing

There have been tremendous efforts towards the goal of automated log parsing. Traditional log parsing methods utilize specially designed features or heuristics (e.g., n-gram [9], prefix tree [10], [11]) based on domain knowledge to extract common patterns for log parsing. These approaches, however, cannot comprehend semantic information in log messages, thus achieving sub-optimal performance when parsing logs whose template design does not match well with the handcrafted features. For example, Drain [10] assumes that logs from the same template share the same length, which is inadequate for logs with nested objects [12]. To address these limitations, semantic-based log parsers [13]–[15] train deep learning models in a supervised manner with labelled log-template pairs to enhance parsing accuracy with semantic comprehension. For example, UniParser [14] trains a bidirectional long short-term memory (Bi-LSTM) model on heterogeneous labelled log data for log parsing. LogPPT [15] introduces a novel log parsing paradigm by employing template-free prompt tuning [16] to fine-tune a small pre-trained language model (PLM), RoBERTa [17], for log parsing. These semantic-based approaches are trained with limited amount of labeled log messages, thus may encounter the issue about the representativeness of the data.

With the increasing popularity of large language models (LLMs), some of the recent studies [18]–[20] propose to

¶: Corresponding author

1558-1225/25/$31.00 ©2025 IEEE
DOI 10.1109/ICSE55347.2025.00174

leverage the text understanding capacity of *commercial* large language models (LLMs) (e.g., ChatGPT) for automated log parsing. Specifically, LLM-based parsers adopt the in-context learning (ICL) prompting technique to adapt LLMs to the log parsing task. For example, Le and Zhang [18] validated the potential of LLMs in log parsing and obtained promising results. DivLog [20] and LILAC [19] enhance the performance of large models by selecting demonstrations from labeled log data and utilizing the in-context learning capabilities of LLMs. They employ different methods to sample a labeled candidate log set. LLMs, however, with their billions of parameters, necessitate deployment on expensive GPU servers, resulting in substantial *financial costs* and significant *environmental impacts* due to high energy consumption and CO_2 emission. Besides, sharing logs to third-party LLM platforms could raise privacy concerns as logs often contain sensitive and personal information [21].

In this paper, we hypothesize that small pre-trained language models (PLMs) such as RoBERTa [17] can be leveraged to perform log parsing effectively and efficiently with proper designs. Small PLMs, despite having fewer parameters than LLMs, are capable of understanding language structure and word relationships and thus can comprehend the semantics of log messages. Besides, small PLMs are lightweight and do not require extensive computational resources for deployment and inference. However, applying these small PLMs to log parsing is a non-trivial task. From our empirical investigation (to be described in Section III), we find that existing semantic-based log parsers overlook several latent characteristics of log data such as small semantic differences between keywords and parameters, thus hinder their capability of adopting small PLMs in log parsing.

To unleash the true potential of semantic-based log parsing with small pre-trained language models, in this paper, we propose a novel end-to-end framework, namely UNLEASH, to fine-tune PLMs with a small amount of log data for log parsing. We follow the common design of semantic-based log parsing [15], [22] to (1) sample a small set of labelled log data, (2) fine-tune PLMs via few-shot learning, and (3) leverage model inference to extract log templates. However, different from previous works, we also propose three enhancement methods associated with three main phases of semantic-based log parsing to boost the performance of PLMs for log parsing to its limit, including (1) an entropy-based ranking method to select the most informative log samples, (2) a contrastive learning method to enhance the fine-tuning process, and (3) an inference optimization method to improve the log parsing performance.

We have evaluated UNLEASH on 14 large-scale public log datasets from Loghub-2.0 [23]. The experimental results show that our proposed approach can achieve better or comparable performance to state-of-the-art LLM-based log parsing models while being more efficient and cost-effective. More specifically, it achieves the best effectiveness in three out of four evaluation metrics, while being 2.2x faster and requiring 100x less model invocation time compared to the best-performing LLM-based log parser, LILAC [19].

Our major contributions are summarized as follows:

1) We propose an end-to-end log parsing framework, named UNLEASH, which can leverage small PLMs for effective log parsing. UNLEASH is designed to be lightweight and efficient, making it suitable for deployment in real-world scenarios.
2) In UNLEASH, we design three enhancement methods to boost the performance of PLMs for log parsing. Specifically, we propose (1) an entropy-based ranking method to select the most informative log samples, (2) a contrastive learning method to enhance the fine-tuning process, and (3) an inference optimization method to improve the log parsing performance.
3) We evaluate UNLEASH on a set of large-scale log datasets and compare it with state-of-the-art log parsing models. The results show that UNLEASH achieves competitive performance with LLM-based log parsers while being more efficient and cost-effective.

II. BACKGROUND AND RELATED WORK

Log parsing is one of the first steps for log analysis tasks [7]. The straightforward way of log parsing relies on handcrafted regular expressions or grok patterns to extract log templates and parameters [7]. However, manually writing regular expressions to parse a huge volume of logs is time-consuming and error-prone [7]. Some studies [24], [25] extract the log templates from logging statements in the source code to compose regular expressions for log parsing. However, it is not applicable in practice since the source code is often unavailable, especially for third-party libraries [7]. To achieve the goal of automated log parsing, many data-driven approaches have been proposed to identify log templates as the frequent part of log messages.

Data-driven log parsing approaches can be divided into three main groups, including: (1) frequent pattern mining, (2) similarity-based clustering, and (3) heuristics-based parsing. *Frequent pattern mining* based log parsers [9], [26]–[28] extract log templates by identifying those patterns that emerge constantly across the entire log dataset. For example, Logram [9] finds frequent n-gram patterns that emerge constantly across the entire log dataset as templates. *Similarity-based clustering* log parsers [29]–[32] compute distances between two log messages or their signature to cluster them based on similarity. Log messages often have some unique characteristics that are different from general text data, such as the difference between the occurrence of constant tokens and variable tokens. *Heuristics-based* log parsers [10], [11], [33], [34] leverage these characteristics to extract log templates. For example, AEL [33] employs a list of heuristic rules to extract common templates. Drain [10] employs a fixed-depth tree structure to assist in dividing logs into different groups, assuming that all log parameters within specific templates possess an identical number of tokens.

These approaches, however, heavily rely on crafted rules, while a significant performance degradation would happen when the log data deviates from the established rules. To address this issue, semantic-based log parsers propose to

employ deep learning models to learn semantics and system-specific patterns from historical log data so as to parse new log messages. For example, UniParser [14] unifies log parsing for heterogeneous log data by training with labelled data from multiple log sources to capture common patterns of templates and parameters. Pre-trained Language Models (PLMs) have been shown effective in various natural language processing tasks [35], [36] and software engineering tasks [37], [38]. Small PLMs can be trained on relatively smaller datasets and require limited computational resources. LogPPT [15] introduces a novel paradigm for log parsing, employing template-free prompt-tuning to fine-tune a small PLM model, RoBERTa [17]. LogStamp [13] leverages the pre-trained BERT model [36] to formulate the log parsing task as a sequence tagging task by training a token classifier on top of the BERT model.

Recently, some studies [39]–[42] have proposed to utilize large language models (LLMs) owing to their extensive pre-trained knowledge. LLMParser [43] is the first LLM model fine-tuned for log parsing. Other studies have also achieved promising results in log parsing thanks to the strong in-context learning capability of LLMs. Le and Zhang [18] validated the potential of applying LLMs for log parsing. DivLog [20] and LILAC [19] employ different sampling methods to select demonstrations from labelled log data and utilize the in-context learning capabilities of LLMs to enhance the performance of large models. It is reported that these LLM-based parsers have superior performance compared to traditional and semantic-based parsers [19], [20]. Although, it is intriguing to adopt LLMs for log parsing, the high inference cost (e.g., financial cost and energy consumption) of LLMs hinders their practical application in log parsing tasks.

III. Motivation

A. Preliminaries of Semantic-based Log Parsing

Semantic-based log parsers such as LogPPT [15] and LogStamp [13], leverage PLMs to formulate the log parsing task as a sequence labelling problem. The key idea is to use PLMs to predict the semantic labels of tokens in a log message. Existing studies find that fine-tuning pre-trained models with few-shot labelled data achieve the best performance on log parsing [14], [43]. Therefore, in this paper, we focus on the fine-tuning of small PLMs with few-shot labelled data for log parsing. Figure 2 illustrates the overall process of existing semantic-based log parsing methods with PLMs.

Semantic-based log parsers require offline training to fine-tune a small PLM (e.g., RoBERTa [17] and T5-small [44]) on a few labelled log samples with few-shot learning mechanisms [15], [22], [43]. Specifically, a sampling method is used to select a small number of log samples to maximize the diversity of selected log patterns. Then, the selected log samples are tokenized and fed into the PLM to fine-tune the model. After the offline stage, the fine-tuned PLM is used to predict the semantic labels of log tokens in incoming logs. Finally, the predicted labels are converted into log templates. Although semantic-based log parsing has shown promising

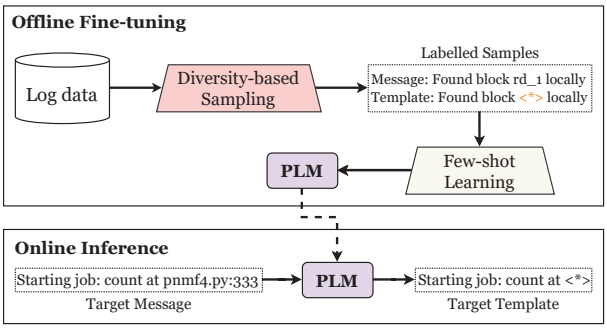

Fig. 2. The overall workflow of semantic-based log parsing

results, they still perform sub-optimally in practice in terms of both effectiveness and efficiency [19].

B. Empirical Investigation

In order to address the bottlenecks of semantic-based log parsing, we conduct an empirical investigation on the Loghub-2.0 benchmark [23] to understand the challenges and prospects of semantic-based log parsing. Through the empirical study, we obtain three observations:

Observation 1: Logs are sampled with high diversity but low informativeness. Existing log parsers typically leverage a sampling method to select a small number of log samples to fine-tune the PLM. For example, LogPPT [15] proposes an adaptive random sampling method to randomly select logs with high semantic diversity. LILAC [19] employs a hierarchical clustering algorithm to partition logs into clusters and randomly selects logs from each cluster. However, we find that logs sampled by these methods have high diversity but low informativeness. Specifically, we compute Shannon's entropy as follows to measure the informativeness of log samples:

$$E(l) = -\sum_{i=1}^{n} p(l_i) \log p(l_i) \tag{1}$$

where l_i is the i^{th} token in log l. Then, for each sampled log, we compare its entropy with the average entropy of all logs with the same template in the dataset. Considering 14 datasets from Loghub-2.0 [23], we find that more than 35% of the logs sampled by LogPPT [15] and LILAC [19] have lower entropy than the average entropy of all logs with the same template in the datasets. The low entropy indicates that these logs are less informative and may not be able to capture the most important log patterns.

Observation 2: The training process neglects the alignment between keywords and parameters. When reviewing the training process of semantic-based log parsing, we find that the fine-tuning process only optimizes the token classification capability of PLMs. However, the semantic meanings of keywords and parameters are not explicitly considered. In Figure 3, we visualize the semantic representations (extracted using BERT [36]) of keywords and parameters in the sampled from three datasets containing logs from different domains. We find that the differences between the semantic representations of

keywords and parameters are not significant. There are many cases where the semantic representations of keywords and parameters are similar to each other. This finding is consistent across all 14 datasets in Loghub-2.0 [23]. This misalignment may lead to the PLM failing to distinguish between keywords and parameters, resulting in incorrect log parsing results.

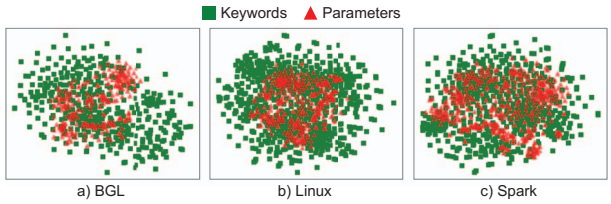

Fig. 3. Visualize semantic representations of keywords and parameters

Observation 3: The inference process is inefficient and time-consuming Existing semantic-based log parsers typically invoke the fine-tuned PLMs for every log message to predict the semantic labels of log tokens. This process is inefficient and time-consuming, especially when the number of logs is large. We take logs in Figure 1 as an example. Two log messages "`Stage 1 was cancelled`" and "`Stage 2 was cancelled`" have the same template. However, the PLM needs to process each log message separately, which is inefficient. In fact, logs, although produced in a large amount, often belong to a limited number of templates. Figure 4 shows that the templates in the three datasets exhibit a highly imbalanced distribution. Such a phenomenon can also be observed in all 14 datasets in Loghub-2.0 [23]. The inference process could be more efficient if we consider this phenomenon in log parsing instead of treating each log message separately.

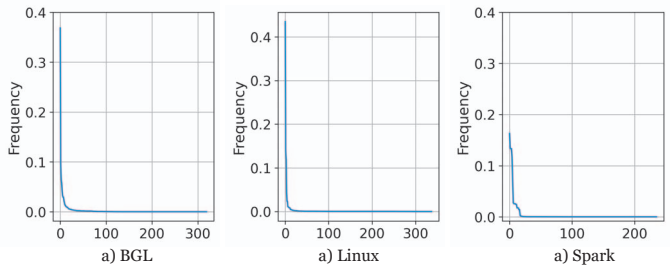

Fig. 4. Imbalanced distribution of log templates

IV. Methodology

Drawing upon the above observations, we propose UNLEASH, a novel framework that enhances the performance of semantic-based log parsing. The overview of UNLEASH framework is illustrated in Figure 5. We follow previous work [15], [43] to design UNLEASH with three main steps: sampling, fine-tuning, and inference. However, different from them, for each step, we propose a novel enhancement method to optimize the parsing performance. Specifically, in the sampling step, we propose an entropy-based sampling method attached to common diversity-based sampling to select diverse and informative samples. In the fine-tuning step, we propose a contrastive learning objective to align the representations of keywords and parameters along with commonly-used few-shot learning objective. Lastly, in the inference step, we employ an adaptive caching mechanism to store the parsed results to speed up the parsing process. We describe each enhancement method in detail in this section.

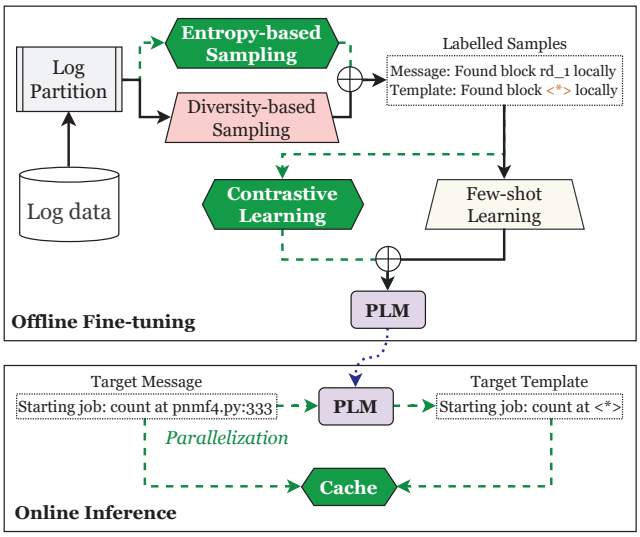

Fig. 5. An overview of the proposed UNLEASH framework

A. Sampling Enhancement

Sampling is an essential step in semantic-based log parsing. It is used to select a small yet representative subset of logs for fine-tuning the small PLM. Based on Observation 1, we propose an entropy-based sampling method aided by common diversity-based sampling to select diverse and informative samples. To this end, UNLEASH first groups log messages into multiple log partitions, where each partition contains logs that share some similarities. After that, UNLEASH selects some logs from each partition based on their information entropy to form the sample set for fine-tuning.

1) Log Partitioning: Inspired by previous studies [10], [45], we leverage several heuristics to partition log messages into different groups. Specifically, we first utilize log length (i.e., the number of tokens of a log message) to separate logs into several groups. Logs with the same number of tokens are more likely to share similar structures, which is a widely adopted heuristic to conduct initial grouping in log analytics [33], [46]. We use white space to split log messages into split to avoid over-fragmented logs and keep parameters intact because many special characters such as "`:`" and "`–`" appear frequently in parameters. After that, we further cluster logs into smaller partitions based on the top-k most frequent tokens in each group. Intuitively, the static parts (i.e, keywords) of a log generally appear more frequently than the dynamic parts (i.e., parameters). Therefore, logs that share the most frequent tokens are more likely to have the same templates [10], [45]. For each log groups obtained from the first step, we extract the top-k tokens based on the frequency and position of each token in a

log message. The frequency is counted among all log messages in the group. Logs with the same top-k pairs of tokens and their positions are grouped together.

2) Entropy-based Sampling: In this phase, we aim to select diverse and representative log messages as fine-tuning samples from the fine-grained log partitions. On one hand, logs from different partitions exhibit high diversity and thus should be selected to train the model. On the other hand, logs from the same partition share similar structures and thus should be carefully selected to avoid redundancy and over-fitting. To this end, we propose an entropy-based sampling method to select K diverse and informative samples from each partition based on the information entropy of log messages. Specifically, first, we equally assign a sample budget $B(p_i) = \frac{K}{|p_i|}$ to each partition p_i, where $|p_i|$ is the number of log messages in partition p_i. Then, we select $B(p_i)$ log messages from partition p_i (sorted by size in descending order) based on their information entropy calculated as in Equation 1. Algorithm 1 shows the detailed procedure of entropy-based sampling.

Algorithm 1: Entropy-based Sampling

Input: Log partitions $P = \{l_1, l_2, \ldots, l_n\}$; Sample budget B_p
Output: $\mathcal{D}$: a set of B_p-labelled samples

1 $\mathcal{D} \leftarrow \emptyset$
2 $E(l_i) \leftarrow$ Shannon Entropy$(l_i), \forall l_i \in P$
3 $P \leftarrow$ Sort$(P, E(l_i))$
4 **while** $B_p > 0$ **do**
5 $S_{ft} \leftarrow \emptyset$
6 **for** $l_i \in P$ **do**
7 **if** *first token of* $l_i \notin S_{ft}$ **then**
8 $S_{ft} \leftarrow S_{ft} \cup \{$first token of $l_i\}$
9 $\mathcal{D} \leftarrow \mathcal{D} \cup \{l_i\}$
10 $B_p \leftarrow B_p - 1$
11 **if** $B_p = 0$ **then**
12 **break**
13 **end**
14 $P \leftarrow P \setminus \{l_i\}$
15 **end**
16 **end**
17 **end**
18 **return** $\mathcal{D}_i$

Algorithm 1 takes a log partition p_i and a sample budget $B(p_i)$ as input and returns a set of selected log messages $\mathcal{D}$ as output. At line 1, the algorithm initializes an empty set $\mathcal{D}$ to store the selected log messages. Then, the information entropy, $E(l_i)$, of each log message in partition p_i is computed at line 2 using Equation 1. At line 3, log messages in partition p_i are sorted by their entropy in descending order. At lines 4-17, the algorithm iteratively selects log messages from partition P until the sample budget B_p is exhausted. Specifically, at lines 6-16, the algorithm iterates through all logs in partition P and adds the log message l_i to $\mathcal{D}$ if its first token has not appeared in the set S_ft. As logs in P are sorted by their entropy, the algorithm selects logs with higher entropy first. The set S_ft is used to ensure the diversity of selected logs. Finally, the algorithm returns the set $\mathcal{D}$ at line 18.

B. Fine-tuning Enhancement

To apply a small PLM to the log parsing task, it is necessary to fine-tune the model with a few labelled log samples selected in the previous stage. Based on Observation 2, we propose to align the representations of keywords and parameters during the fine-tuning process to enhance the performance of the model via contrastive learning aided by common few-shot learning objective.

1) Few-shot learning objective: Simply fine-tuning the PLM with a few labelled log samples may lead to overfitting and poor generalization, especially for log parsing when logs are heterogeneous [15], [43]. To address this issue, we follow previous work [15], [47] to resort to prompt tuning PLMs with a few labelled samples. Specifically, we reformulate the log parsing task as a label word prediction task to reuse the pre-training objective (i.e., masked token prediction) of PLMs. Given an input log message $X = \{x_1, x_2, ..., x_n\}$, where x_i is the i-th token in the log message, we construct a target sequence $Y = \{y_1, y_2, ..., y_n\}$ by replacing the parameter tokens with a special token "<parameter>" as follows:

$$y_i = \begin{cases} \text{“<parameter>”}, & \text{if } x_i \text{ is a parameter token} \\ x_i, & \text{otherwise} \end{cases} \quad (2)$$

Then, we fine-tune the PLM by minimizing the cross-entropy loss between the predicted logits and the target sequence Y. The fine-tuning process can be formulated as follows:

$$\mathcal{L}_{\text{CE}} = -\sum_{i=1}^{n} \log P(x_i = y_i | X) \quad (3)$$

where $P(x_i = y_i|X)$ is the probability of predicting the i-th token x_i as the target token y_i given the input log message X, which is computed by the softmax function over the output logits of the PLM. This few-shot learning objective is inspired by LogPPT [15] and has been shown to be effective in enhancing the performance of PLMs on log parsing tasks. Different from LogPPT, we randomly assign the embedding of the special token "<parameter>" and let the model adjust the embedding during fine-tuning instead of using a fixed embedding for the special token computed using the most frequent parameter tokens. The reason is that in complex datasets with highly imbalanced template distributions, the most frequent parameter tokens are not always representative of all parameter tokens, which may lead to suboptimal performance [20].

2) Contrastive learning objective: Aligning the representations of keywords and parameters is crucial yet challenging for log parsing, as the model needs to distinguish between them to accurately extract the log templates. The few-shot learning objective only focuses on predicting the target tokens, thus being insufficient to learn the representation of the label token "<parameter>". To address this issue, we propose a contrastive learning objective to enhance the fine-tuning process by aligning the representations of keywords and parameters. Specifically, we introduce a contrastive loss to encourage the model to pull the representations of the parameter tokens closer

to the representation of the special token "<parameter>" and push them away from the representations of the keyword tokens. The contrastive loss is formulated as follows:

$$\mathcal{L}_{\text{CL}} = -log\frac{\exp(\text{sim}(r_i, r_p))}{\exp(\text{sim}(r_i, r_p)) + \sum_{j=1}^{n}\exp(\text{sim}(r_i, r_j^-))} \quad (4)$$

where r_i is the representation of the i-th token in the log message, r_p is the representation of the special token "<parameter>", r_j^- is the representation of the j-th token in the log message that is not a parameter token, and $\text{sim}(r_i, r_j)$ is the cosine similarity between two representations. This objective allows the model to align the embedding of the parameter tokens with the special token "<parameter>" to enhance the ability of the model to distinguish between keywords and parameters.

The contrastive loss is added to the cross-entropy loss to form the final fine-tuning objective:

$$\mathcal{L}_{\text{FT}} = \mathcal{L}_{\text{CE}} + \lambda \times \mathcal{L}_{\text{CL}} \quad (5)$$

C. Inference Enhancement

During the inference phase, we parse log messages line by line using the fine-tuned PLM to enable online parsing. Specifically, we first tokenize the input log message and feed it into the fine-tuned PLM to obtain the corresponding target tokens. If a token is predicted as the special token "<parameter>", we mark it as a parameter; otherwise, we keep it as a keyword. We follow previous work [15], [48] to apply a heuristic post-processing step to merge adjacent parameter tokens into a single parameter. This step is necessary because the PLM may predict each token as a separate parameter, leading to fragmented parameters. After that, we concatenate the keywords and parameters to form the final log template. This standard inference process, however, is time-consuming and less consistent when parsing large-scale and imbalanced log datasets [19]. Based on Observation 3, we propose an adaptive caching mechanism and a parallelization strategy to enhance the parsing efficiency and consistency. Specifically, a parsing tree is used as a cache to store the parsed results and match new log messages with the existing ones to avoid redundant parsing and improve the parsing consistency. We also introduce a parallelization strategy to further enhance the parsing efficiency.

1) Adaptive Caching: Caching is essential to guarantee the efficiency and consistency of the parsing process. Therefore, we leverage the adaptive parsing tree [19] to store the parsed results. Figure 6 illustrates the parsing tree, which is a prefix tree, in which each node represents a token in the log message and the leaves store the log templates.

The parsing tree include two main operations, including *cache matching* and *cache updating*, to facilitate the parsing process. Specifically, when parsing a new log message, we first search the parsing tree to find whether there exists a template that matches the log message. If a match is found, we directly return the corresponding template. Otherwise, we parse the log message using the fine-tuned PLM and update the parsing tree with the new template. This adaptive caching mechanism can significantly reduce the parsing time by avoiding redundant parsing and improve the parsing consistency by reusing the parsed results.

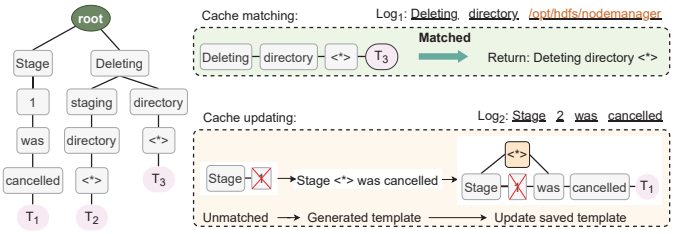

Fig. 6. An illustration of the parsing tree

Cache Matching. Given a new log message, UNLEASH first searches the parsing tree for the corresponding template, which can avoid redundant parsing. To match the log message with the parsing tree, we first split the log message into a set of tokens by common delimiters (e.g., white space, brackets, and equal signs). Then, we traverse the parsing tree using deep-first search from the root node to the leaf node based on the sequential order of the tokens. Note that, with the wildcard token "<*>" as a node, we employ recursive matching [45] to match it with more than one token. This recursive step continues until the following token matches the next node in the parsing tree. If the search process reaches a leaf node, we return the template stored in the corresponding leaf node. If no match is found, UNLEASH proceeds to the parsing step using the fine-tuned PLM.

Cache Updating. The trained PLM may not always predict the correct log template due to the complexity and heterogeneity of log messages. Therefore, we update the parsing tree with the new template to improve the parsing consistency. Specifically, we compare the extracted template with the relevant templates in the parsing tree during cache updating by recursively traversing the parsing tree as in the cache matching step. If the newly extracted template exhibits high similarity with an existing template, we merge them into a single template. Otherwise, we add the new template to the parsing tree as a new leaf node. Specifically, UNLEASH computes the similarity between two templates using LCS (Longest Common Subsequence) [11], [19], [49]. If the similarity is higher than 0.8 [19], we merge the two templates into a single template by replacing the discriminate tokens with "<*>". In Figure 6, the stored template "`Stage 1 was cancelled`" is merged with the new template "`Stage <*> was cancelled`". In this case, UNLEASH refines the "`1`" node to "<*>" to update the parsing tree.

2) Parallelization: Our UNLEASH framework is designed to parse log messages line-by-line. When parsing large-scale log datasets, it is necessary to parallelize the parsing process to improve the efficiency. UNLEASH is equipped with a small PLM, which is computationally efficient (i.e., only require 0.3GB of GPU memory), and thus allowing parallelization with even a single GPU. Specifically, given a large log dataset, we divide it into several chunks and assign each chunk to be

parsed by a subprocess. Each subprocess holds a parsing tree to store the parsed results and updates the parsing tree with the new templates. After parsing all log messages in the chunk, the subprocess returns the parsed results to the main process, which merges the results from all subprocesses to form the final parsing results. This process allows UNLEASH to leverage the parallel computing power to significantly speed up the parsing process, especially for large-scale log datasets.

V. EXPERIMENTAL DESIGN

A. Research Questions

In this paper, we comprehensively evaluate UNLEASH by answering the following research questions (RQs):

- RQ1: How does UNLEASH perform compared to state-of-the-art log parsing methods?
- RQ2: How does UNLEASH generalize across different training scenarios and scale with increasing log sizes?
- RQ3: How does the proposed sampling enhancement contribute to the performance of UNLEASH?
- RQ4: How does the proposed fine-tuning enhancement contribute to the performance of UNLEASH?
- RQ5: How does the proposed inference enhancement contribute to the performance of UNLEASH?

B. Datasets

We conduct experiments on 14 large-scale public log datasets (i.e., Loghub-2.0 [23]) originated from the LogPai project [7]. Loghub-2.0 is annotated with ground-truth templates for 14 diverse log datasets, which covers a variety of systems from distributed systems, standalone software, supercomputers, PC operating systems, mobile systems, microservices, etc. Compared with the original Loghub [50], Loghub-2.0 is equipped with 3× templates and 1,800× log messages on average, which can support a more comprehensive evaluation on accuracy and efficiency. On average, each dataset in Loghub-2.0 comprises 3.6 million log messages. In total, the collection contains approximately 3,500 unique log templates.

C. Evaluation Metrics

Following recent studies [14], [19], [48], we use the following four evaluation metrics in our experiments:

- *Grouping Accuracy (GA)* [7] is a log-level metric that measures the amount of log messages of the same template that are grouped together by the parser. It is computed as the ratio of correctly grouped log messages over all log messages, where a log message is regarded as correctly grouped if and only if its predicted template has the same group of log messages as the oracle.
- *Parsing Accuracy (PA)* [14] is a log-level metric that measures the correctness of extracted templates and variables. It is defined as the ratio of correctly parsed log messages over all log messages, where a log message is considered to be correctly parsed if and only if all its static text and dynamic variables are identical with the oracle.
- *F1 score of Grouping Accuracy (FGA)* [48] is a template-level metric that measures the ratio of correctly grouped templates. It is computed as the harmonic mean of precision and recall of grouping accuracy, where the template is considered as correct if log messages of the predicted template have the same group of log messages as the oracle.
- *F1 score of Template Accuracy (FTA)* [19] is a template-level accuracy computed as the harmonic mean of precision and recall of Template Accuracy. A template is regarded as correct if and only if it satisfies two requirements: log messages of the predicted template have the same group of log messages as the oracle, and all tokens of the template are the same as the oracle template.

D. Baselines

In accordance with recent benchmark studies [7], [48], we select four open-source and state-of-the-art log parsers for comparison with our method. The first two, UniParser [14] and LogPPT [15], are chosen due to their highest parsing accuracy among semantic-based log parsers [23]. LLMParser [43] is the first LLM model fine-tuned for log parsing. We also compare our method with LILAC, state-of-the-art in-context learning based log parsers that leverage commercial LLMs (GPT-3.5) for log parsing. To ensure a fair comparison, we use the implementations of all baselines from their replication repositories, choosing the default settings or hyper-parameters. Existing studies [15], [19], [20] have shown that semantic-based log parsing methods are more accurate than traditional methods. Besides, due to space constraints, we do not present the comparison with traditional methods in this paper. They are included in the supplementary material instead.

E. Environment and Implementation

We conduct all the experiments on a Ubuntu 20.04.4 LTS server with 32GB RAM and an NVIDIA RTX 3090 24G GPU. We implement UNLEASH with Python 3.11 and PyTorch 2.3. Following recent studies [19], [23], in the sampling module, we set $k = 3$ as the default for top-k most frequent tokens in log partitioning. We choose RoBERTa [17] (base model with 125M parameters) as the PLM for the fine-tuning module. To simulate the practical usage of UNLEASH, we follow recent studies [19], [23] to use the sampling algorithm to select 32 labelled samples as training data from the first 20% of the log messages in each dataset. During the training process, we utilize AdamW [51] optimizer and set the initial learning rate to 5e-5. We set the batch size as 16 and train the model for 1000 steps, resulting in a training process of less than 1 minute for each dataset. AdamW optimizer is used with a linear decaying schedule with 10% warm-up steps. We set the weight for contrastive loss, $\lambda = 0.1$, for the fine-tuning module. We use the same hyper-parameters for all the datasets to ensure a fair comparison.

VI. EVALUATION RESULTS

A. RQ1: Comparison with the state-of-the-art methods

In this section, we evaluate our proposed UNLEASH framework from three aspects: accuracy, robustness, and efficiency.

TABLE I
ACCURACY COMPARISON BETWEEN BASELINES AND UNLEASH ON LOG DATASETS

	LogPPT				UniParser				LLMParser				LILAC				UNLEASH			
	GA	FGA	PA	FTA	GA	FGA	PA	FTA	GA	FGA	PA	FTA	GA	FGA	PA	FTA	GA	FGA	PA	FTA
Apache	0.786	0.605	0.948	0.368	0.948	0.687	0.942	0.269	0.871	0.800	0.390	0.308	**1.000**	**1.000**	**0.999**	**0.897**	**1.000**	**1.000**	0.995	0.800
HDFS	0.721	0.391	0.943	0.312	**1.000**	0.968	0.948	0.581	—	—	—	—	**1.000**	**0.968**	0.999	**0.946**	**1.000**	**0.968**	**1.000**	0.925
Hadoop	0.483	0.526	0.666	0.434	0.691	0.628	0.889	0.476	0.786	0.636	0.694	0.519	0.872	**0.962**	0.832	**0.774**	**0.982**	0.948	**0.902**	0.740
Zookeeper	0.967	0.918	0.845	0.809	0.988	0.661	**0.988**	0.510	0.897	0.492	0.899	0.453	**1.000**	0.967	0.687	**0.868**	**1.000**	**1.000**	0.938	0.832
HealthApp	0.998	0.947	**0.997**	0.822	0.461	0.745	0.817	0.462	0.884	0.836	0.982	0.661	**1.000**	0.981	0.729	0.873	**1.000**	**1.000**	0.996	**0.936**
HPC	0.782	0.780	**0.997**	0.768	0.777	0.660	0.941	0.351	0.836	0.353	0.923	0.242	0.869	0.907	0.705	**0.813**	**0.996**	**0.940**	0.991	0.808
Linux	0.205	0.712	0.168	0.428	0.285	0.451	0.164	0.232	0.347	0.622	0.257	0.091	**0.971**	**0.931**	0.767	**0.743**	0.922	0.773	**0.852**	0.635
OpenSSH	0.277	0.081	0.654	0.105	0.275	0.900	0.289	0.500	0.358	0.397	0.893	0.321	0.690	0.838	0.941	0.865	**0.748**	**0.877**	**1.000**	**0.904**
OpenStack	0.534	0.874	0.406	0.738	**1.000**	0.969	0.516	0.289	0.742	0.217	0.265	0.286	**1.000**	**1.000**	**1.000**	0.958	**1.000**	**1.000**	**1.000**	**0.979**
Proxifier	0.989	0.870	1.000	0.957	0.509	0.286	0.634	0.457	0.994	0.833	0.697	0.583	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
Mac	0.544	0.493	0.390	0.274	0.737	0.699	0.688	0.283	0.685	0.267	0.367	0.128	0.877	**0.827**	0.638	**0.555**	**0.913**	0.767	**0.747**	0.503
Spark	0.476	0.374	0.952	0.299	0.854	0.020	0.795	0.012	—	—	—	—	**1.000**	0.899	**0.973**	**0.759**	0.985	**0.918**	0.850	0.703
Thunderbird	0.564	0.216	0.401	0.117	0.579	0.682	0.654	0.290	—	—	—	—	0.806	0.792	0.558	**0.569**	**0.915**	**0.848**	**0.589**	0.441
BGL	0.245	0.253	0.938	0.261	0.918	0.624	0.949	0.219	—	—	—	—	0.894	0.859	**0.958**	0.749	**0.905**	**0.924**	0.863	**0.771**
Average	0.612	0.574	0.736	0.478	0.716	0.578	0.730	0.317	0.740	0.545	0.637	0.359	0.927	0.924	0.842	**0.812**	**0.955**	**0.926**	**0.909**	0.784

Note: "—" denotes timeout. The best results are in bold.

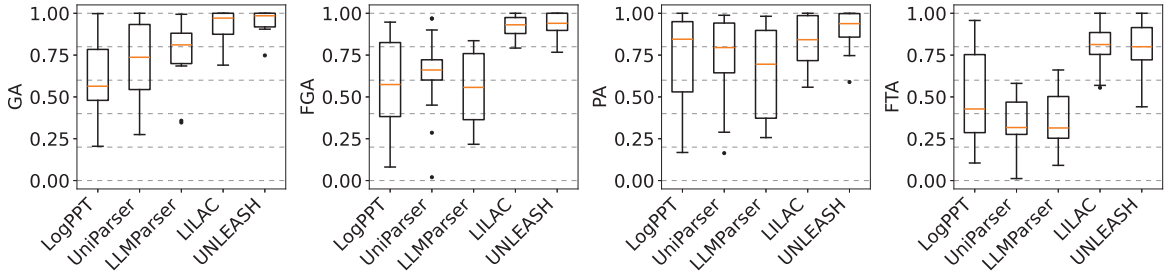

Fig. 7. Robustness comparison between baselines and UNLEASH

1) Accuracy: We compare our method with four state-of-the-art log parsers, including two semantic-based log parsers (LogPPT [15] and UniParser [14]) and two LLM-based log parsers (LLMParser [43] and LILAC [19]). Table I shows a comparative analysis of these methods in terms of four evaluation metrics. For a fair comparison, we adopt the default settings of all methods and set $K = 32$ as default number of few-shot labeled samples. Note that on some large datasets (e.g., BGL), LLMParser cannot complete the sampling process within 5 hours, so we mark the corresponding results as "—".

The experimental results show that UNLEASH outperforms all baseline methods in terms of three out of four evaluation metrics, i.e., GA, FGA, and PA. It can achieve comparable FTA with the top-performing method, LILAC. For example, UNLEASH achieves the highest GA on 12 out of 14 datasets with an average of 0.955, which is 2.8% higher than the best LLM-based parser, LILAC. Compared to semantic-based parsers (i.e., UniParser and LogPPT), UNLEASH significantly exceeds them on all datasets by 23.9-34.3% and 34.8-35.2% on average in terms of GA and FGA. Regarding template-level metrics (i.e., PA and FTA), UNLEASH achieves the highest PA on 9 out of 14 datasets with an average of 0.909, which is 6.7% higher than the second best. In terms of FTA, UNLEASH achieves an average of 0.784, which is 2.8% lower than the best LLM-based parser, LILAC, but 30.6% higher than the best semantic-based parser, LogPPT. To further verify the effectiveness of UNLEASH, we conduct a pairwise *t-test* on the parsing results of UNLEASH and the baselines. The *t-test* results (at the significance level of 0.05) show no statistically significant difference between the results of UNLEASH and the top-performing baseline, LILAC, regarding all metrics, indicating that they achieve similar performance. Compared to other baselines, the *t-test* results show that UNLEASH significantly outperforms them in terms of all evaluation metrics. Overall, UNLEASH achieves the best performance in terms of GA, FGA, and PA, and comparable performance in terms of FTA. This demonstrates the effectiveness of our proposed method in log parsing, which can achieve high accuracy in both log-level and template-level parsing tasks.

2) Robustness: In practice, log parsers are required to handle various types of log messages with different lengths and formats. Therefore, we evaluate the robustness of UNLEASH and the baselines on different types of logs. Figure 7 shows the distribution of accuracy results across different log datasets. The results show that UNLEASH is robust and performs well

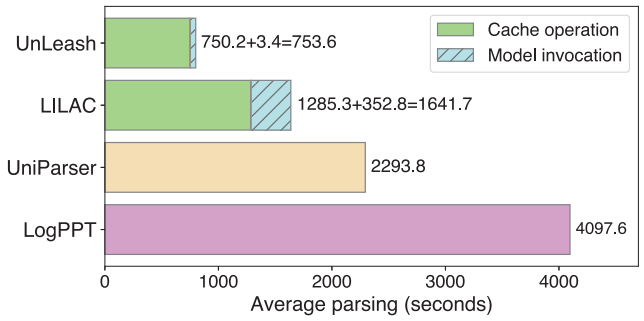

Fig. 8. Efficiency of baselines and UNLEASH. The parsing time of LILAC and UNLEASH includes cache operation and model invocation time. We do not show the parsing time of LLMParser as it cannot complete on large datasets.

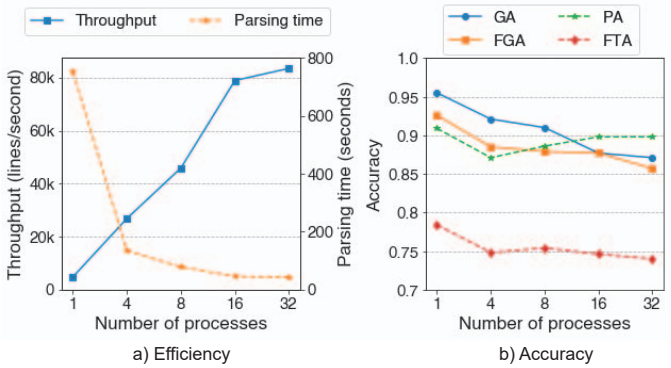

Fig. 9. Experimental results on the scalability of UNLEASH with parallelization

on most of datasets. It yields a median of 0.985 and 0.938 for GA and PA robustness, which is 1.4% and 9.6% higher than the best LLM-based parser, LILAC. In terms of FGA and FTA, it achieves a median of 0.940 and 0.800, which is 0.9% higher and only 1.3% lower than LILAC. Compared to other baselines, UNLEASH exceeds them by 9.3%-48.5% in terms of all evaluation metrics' median values. The results indicate that UNLEASH is more robust and stable in handling various types of log messages, which is crucial for log parsing in practice.

3) Efficiency: As logs are produced massively in practice, the efficiency of log parsing is crucial for real-world applications. In this experiment, we evaluate the efficiency of UNLEASH and the baselines in terms of parsing time. For a fair comparison, we use a single process for the inference of UNLEASH to compare with other methods in terms of efficiency. Figure 8 shows the average parsing time across 14 datasets, with an average of 3.6 million log messages per dataset. The results show that UNLEASH takes an average of 753.6 seconds to process an average of 3.6 million log messages, which is 2.2×, 3.0×, and 5.4× faster than LILAC, UniParser, and LogPPT, respectively. It is worth noting that although both UNLEASH and LILAC adopt a parsing cache to store the intermediate results, UNLEASH achieves a significantly higher efficiency than LILAC. Specifically, UNLEASH takes 1.7× less time for cache operations and 103.8× less model inference time than LILAC. The main reason is that UNLEASH utilizes a small PLM with only 125M parameters, which is more efficient. The high model efficiency also allows UNLEASH to offload more workload to the model side, and thus reduces the cache operation time as we do not always try to match logs with templates in the cache as LILAC does. Overall, UNLEASH achieves superior efficiency in log parsing, which is crucial in practice.

B. RQ2: On the scalability and generalization of UNLEASH

1) Scalability: UNLEASH can easily scale up with a parallelization acceleration. As it is equipped with a small PLM, the computational resources required for inference are relatively low (0.3GB of GPU memory), thus allowing for parallelization with a single GPU. We run UNLEASH with different numbers of subprocesses (i.e., executors) from 1 to 32. Figure 9 shows the results in terms of average parsing time and accuracy across all experimented log datasets. The results show that as the number of executors increases, the parsing time decreases linearly, while the accuracy remains relatively stable. Specifically, UNLEASH achieves a 17.5× speedup with 32 executors. It can process about 83,500 log lines per second. In terms of accuracy, UNLEASH witnesses a slight decrease in GA and FGA. The reason is that different executors store their own cache to avoid conflicts, which leads to a slight decrease in cache matched rate, resulting in a certain degree of inconsistency in the parsing results. Nevertheless, the overall performance is still reasonable especially for PA and FTA. Overall, the results demonstrate the scalability of UNLEASHwith parallelization.

2) Generalization over different PLMs: In this paper, we use RoBERTa as the default base model for UNLEASH. To evaluate the generalization of UNLEASH over different PLMs, we conduct experiments with three other small PLMs, i.e., DeBERTa [52] and two variants (small and base) of CodeBERT [37]. These models are pre-trained using different objectives and corpora, thus suitable to evaluate the generalization of UNLEASH. Table II shows the average results of GA, PA, FGA, and FTA. It is obvious that UNLEASH achieves stable performance across different PLMs despite the differences in their semantic understanding ability. For example, UNLEASH achieves an average of more than 0.9 in terms of GA, PA, and FGA with all experimented PLMs. These results indicate that UNLEASH is generalizable to different PLMs.

TABLE II
PERFORMANCE OF UNLEASH WITH DIFFERENT PLMS

	#Params	GA	FGA	PA	FTA
UNLEASH w/ RoBERTa	125M	0.955	0.926	0.909	0.784
w/ CodeBERT-base	125M	0.944	0.922	0.911	0.764
w/ CodeBERT-small	84M	0.921	0.912	0.823	0.706
w/ DeBERTa-base	140M	0.946	0.934	0.927	0.766

3) Generalization over different fine-tuning size: We next evaluate the generalization of UNLEASH over different numbers of labelled samples for fine-tuning PLMs. Specifically, we select the base models of RoBERTa, CodeBERT, and DeBERTa to

fine-tune with the number of samples varying from 32 to 256. From the average accuracy results shown in Figure 10, we can see that as the fine-tuning size increases from 32 to 256 shots, the performance of UNLEASH gradually improves in all evaluation metrics. These results imply that by increasing the number of labelled samples for fine-tuning, the parsing accuracy of the models can be effectively enhanced, allowing for more precise and reliable results. Nevertheless, UNLEASH is still able to achieve good performance with a limited fine-tuning size (i.e., 32 shots).

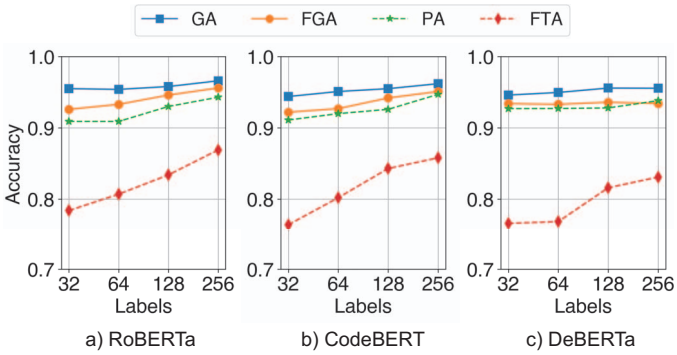

Fig. 10. Performance of UNLEASH with different numbers of labels

C. RQ3: On the effectiveness of the sampling enhancement

Recent works [15] [19] [43] have proposed various sampling methods aimed at covering as many logs or templates as possible to enhance model fine-tuning or in-context learning. To demonstrate the advantages of our proposed entropy-based sampling, we compare it with other sampling methods, including: (1) *random sampling*: we randomly select 32 logs as fine-tuning samples; (2) *adaptive random sampling*: we adopt the adaptive random sampling method proposed in LogPPT [15]; and (3) *hierarchical sampling*: we adopt the hierarchical sampling method proposed in LILAC [19]. Table III shows the performance of UNLEASH under these different sampling methods. It is clear that when using other sampling methods, UNLEASH witnesses noticeable performance degradation in all evaluation metrics, especially with random sampling. For example, the PA decreases 8.0% with hierarchical sampling because it cannot select informative samples. In contrast, our proposed method can both select diverse logs across different semantically similar groups and informative logs within each group, which leads to the best performance in all evaluation metrics. The results demonstrate that our proposed entropy-based sampling method is more effective in selecting diverse and informative samples for training semantic-based log parsers.

D. RQ4: On the effectiveness of the fine-tuning enhancement

In this section, we evaluate the effectiveness of the proposed fine-tuning enhancement by comparing UNLEASH with three variants: (1) *w/o Contrastive Learning*: we remove the contrastive learning objective from the fine-tuning process; (2) *w/ Fine-tuning*: we fine-tune the PLM with the standard binary token classification objective; and (3) *w/ LogPPT Prompt Tuning*: we adopt the few-shot prompt-tuning objective from LogPPT [15]. The results in Table IV show that UNLEASH with the proposed fine-tuning enhancement achieves the best performance in all evaluation metrics.

TABLE III
PERFORMANCE OF UNLEASH WITH DIFFERENT SAMPLING METHODS

	GA	FGA	PA	FTA
UNLEASH	0.955	0.926	0.909	0.784
w/ Random Sampling	$0.913_{(\downarrow 4.6\%)}$	$0.825_{(\downarrow 12.2\%)}$	$0.855_{(\downarrow 6.3\%)}$	$0.564_{(\downarrow 39.0\%)}$
w/ Adaptive Random Sampling	$0.930_{(\downarrow 2.7\%)}$	$0.909_{(\downarrow 1.9\%)}$	$0.873_{(\downarrow 4.1\%)}$	$0.731_{(\downarrow 7.3\%)}$
w/ Hierarchical Sampling	$0.925_{(\downarrow 3.2\%)}$	$0.909_{(\downarrow 1.9\%)}$	$0.842_{(\downarrow 8.0\%)}$	$0.758_{(\downarrow 3.4\%)}$

TABLE IV
PERFORMANCE OF UNLEASH WITH DIFFERENT TUNING METHODS

	GA	FGA	PA	FTA
UNLEASH	0.955	0.926	0.909	0.784
w/o Contrastive Learning	$0.954_{(\downarrow 0.1\%)}$	$0.923_{(\downarrow 0.3\%)}$	$0.895_{(\downarrow 1.4\%)}$	$0.781_{(\downarrow 0.3\%)}$
w/ Fine-tuning	$0.654_{(\downarrow 30.1\%)}$	$0.258_{(\downarrow 66.8\%)}$	$0.447_{(\downarrow 42.6\%)}$	$0.161_{(\downarrow 62.3\%)}$
w/ LogPPT Prompt Tuning	$0.932_{(\downarrow 2.3\%)}$	$0.895_{(\downarrow 3.1\%)}$	$0.896_{(\downarrow 1.3\%)}$	$0.761_{(\downarrow 2.3\%)}$

E. RQ5: On the effectiveness of the inference enhancement

In Section VI-B, we have discussed the advantages of the proposed parallelization acceleration. In this section, we evaluate the effectiveness of the adaptive caching mechanism used in the inference enhancement component. To this end, we run UNLEASH without the adaptive cache and compare the performance with the full version. The results in Table V, in terms of average accuracy and parsing time, show that without the adaptive cache, UNLEASH performs worse in terms of both effectiveness and efficiency. Specifically, it achieves much lower GA and FGA while taking almost 16× longer to parse the same amount of logs. This demonstrates the advantages of the proposed adaptive caching mechanism in enhancing the performance of UNLEASH.

TABLE V
PERFORMANCE OF UNLEASH WITHOUT ADAPTIVE CACHING

	GA	FGA	PA	FTA	Parsing time (s)
UNLEASH	0.955	0.926	0.909	0.784	735.6
$\text{w/o}_{\text{Adaptive Caching}}$	$0.764_{(\downarrow 25.0\%)}$	$0.748_{(\downarrow 23.8\%)}$	$0.912_{(\uparrow 0.3\%)}$	$0.672_{(\downarrow 16.7\%)}$	$11411.1_{(\uparrow 15.5\times)}$

VII. DISCUSSION

A. Limitations

The inherent limitation of UNLEASH is that it is sensitive to the quality of labelled logs. When the logging behaviours of a system are inconsistent, the performance of UNLEASH may degrade. For example, on the Thunderbird dataset, for the log message "`running '/usr/sbin/up2date --nox -i ganglia-gmond' with root privileges on behalf of 'root'`", UNLEASH considers the character "`'`" as a part of parameters. However, the ground truth sometimes considers it as a part of keywords; in this case, it is

"`running '<*>' with <*> privileges on behalf of <*>`". Although this does not affect the ability to identify variables of UNLEASH, it may affect the performance of UNLEASH in terms of PA and FTA (0.589 and 0.441 on Thunderbird). We notice that this phenomenon also occurs in some other datasets, such as Linux and Mac, leading to a decrease in the performance of UNLEASH. In the future, we will investigate robust training techniques and incorporate syntactic information of logs to improve the performance of UNLEASH in handling the inconsistency in logging behaviours.

B. Why NOT LLM-based log parsing?

Recently, some LLM-based log parsers (e.g., LILAC [19]) have been proposed. These parsers leverages the text semantic understanding capabilities of LLMs to extract log templates and obtained promising results. We have shown that semantic-based log parsers with small PLMs can achieve better or comparable performance to these state-of-the-art LLM-based log parsers. Furthermore, we have identified some challenges for employing LLM-based log parsing in practice:

(1) Inference cost: Typically, to host an LLM, it requires deployment on expensive GPU servers, resulting in substantial *financial costs* (e.g., approximately $700,000 per day for ChatGPT [53]) and significant *environmental impacts* due to high energy consumption and CO_2 emission (e.g., 12,800 metric tons of CO_2 per year for GPT-3 [54]). Log parsers typically are integrated into reliability management pipelines to work in a real-time manner. As system logs are produced in a huge amount [55], incorporating LLM-based log parsers into reliability management pipelines could result in more frequent and intensive LLM requests.

(2) Privacy concern: Logs often contain sensitive and personal information, particularly when logs are unified across a diverse collection of software components [21]. Specifically, developers record various information during software development for troubleshooting. These information can be *direct identifiers* such as email addresses, phone numbers, and user names; *indirect identifiers* such as device MAC addresses, IMEI, and serial numbers; *user location* such as fine- and coarse-grain GPS coordinates. Collecting these logs raises privacy concerns, especially when logs are shared to third-party LLM platforms to facilitate log parsing.

(3) Unstable performance: Randomness is the inherent variability in the outputs of LLMs. Given the same input, an LLM may produce different outputs. While this randomness can aid the model's creativity and flexibility, it may also result in unpredictable and sometimes irrelevant responses. Besides, the behavior of the "same" LLM service can change substantially in a relatively short amount of time because these LLMs themselves can be updated over time. These make it challenging to stably integrate LLMs into larger workflows [56].

C. Threats to Validity

While our study demonstrates promising results, we have identified the following potential threats to validity:

Data quality. In this paper, we used public log datasets [23] for our evaluation. These datasets, although in large-scale, are quite new and may contain errors in the ground truth labels. In the future work, further validation on these datasets is needed to ensure the correctness of the ground truth labels.

Bias in experiments. Randomness may affect the performance through two aspects: (1) the randomness during model optimization, and (2) the randomness in the sampling phase. To reduce the random bias from the sampling phase, we repeat the experiments five times and report the average results following previous work [47], [57].

VIII. CONCLUSION

This paper aims to unleash the true potential of semantic-based log parsing with small pre-trained language models. We have proposed UNLEASH, a novel end-to-end framework, which follows the common design of semantic-based log parsing and also incorporates three novel enhancement methods to boost the performance of log parsing. The main finding of this paper is that, in the log parsing task, PLM-based solutions can achieve better or comparable performance to LLM-based solutions regarding effectiveness, efficiency, and scalability, with a proper design. Our experimental results on large-scale datasets show that UNLEASH can achieve better or comparable performance to state-of-the-art LLM-based log parsing models, while being more efficient and cost-effective.

Therefore, this paper can be beneficial for (1) Researchers: as leveraging LLMs like ChatGPT is trending, this study offers insights into how smaller models can be enhanced with a proper design to perform equally well. This suggests that researchers should not overlook the potential of simpler methods before resorting to expensive LLMs; and (2) Practitioners: this study provides a practical solution for small businesses or individuals with limited resources to create in-house log parsers without relying on expensive LLMs to effectively analyze large-scale and sensitive logs.

IX. DATA AVAILABILITY

Our source code, experimental data, and additional results are available at our project webpage: https://github.com/LogIntelligence/UNLEASH.

ACKNOWLEDGMENTS

This work is supported by Australian Research Council (ARC) Discovery Projects (DP200102940, DP220103044). We also thank anonymous reviewers for their insightful and constructive comments, which significantly improve this paper.

REFERENCES

[1] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. Venkatakrishnan, "Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting," in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pp. 1795–1812, 2019.

[2] A. Oprea, Z. Li, T.-F. Yen, S. H. Chin, and S. Alrwais, "Detection of early-stage enterprise infection by mining large-scale log data," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 45–56, IEEE, 2015.

[3] M. Chow, D. Meisner, J. Flinn, D. Peek, and T. F. Wenisch, "The mystery machine: End-to-end performance analysis of large-scale internet services," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pp. 217–231, 2014.

[4] K. Nagaraj, C. Killian, and J. Neville, "Structured comparative analysis of systems logs to diagnose performance problems," in *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pp. 353–366, 2012.

[5] T. Jia, L. Yang, P. Chen, Y. Li, F. Meng, and J. Xu, "Logsed: Anomaly diagnosis through mining time-weighted control flow graph in logs," in *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pp. 447–455, IEEE, 2017.

[6] V.-H. Le and H. Zhang, "Prelog: A pre-trained model for log analytics," *Proceedings of the ACM on Management of Data*, vol. 2, no. 3, pp. 1–28, 2024.

[7] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pp. 121–130, IEEE, 2019.

[8] S. He, P. He, Z. Chen, T. Yang, Y. Su, and M. R. Lyu, "A survey on automated log analysis for reliability engineering," *ACM computing surveys (CSUR)*, vol. 54, no. 6, pp. 1–37, 2021.

[9] H. Dai, H. Li, C. S. Chen, W. Shang, and T.-H. Chen, "Logram: Efficient log parsing using n-gram dictionaries," *IEEE Transactions on Software Engineering*, 2020.

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

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

[12] Y. Fu, M. Yan, J. Xu, J. Li, Z. Liu, X. Zhang, and D. Yang, "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*, pp. 1566–1577, 2022.

[13] S. Tao, W. Meng, Y. Cheng, Y. Zhu, Y. Liu, C. Du, T. Han, Y. Zhao, X. Wang, and H. Yang, "Logstamp: Automatic online log parsing based on sequence labelling," *ACM SIGMETRICS Performance Evaluation Review*, vol. 49, no. 4, pp. 93–98, 2022.

[14] Y. Liu, X. Zhang, S. He, H. Zhang, L. Li, Y. Kang, Y. Xu, M. Ma, Q. Lin, Y. Dang, *et al.*, "Uniparser: A unified log parser for heterogeneous log data," in *Proceedings of the ACM Web Conference 2022*, pp. 1893–1901, 2022.

[15] V.-H. Le and H. Zhang, "Log parsing with prompt-based few-shot learning," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pp. 2438–2449, IEEE, 2023.

[16] R. Ma, X. Zhou, T. Gui, Y. Tan, L. Li, Q. Zhang, and X.-J. Huang, "Template-free prompt tuning for few-shot ner," in *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 5721–5732, 2022.

[17] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[18] V.-H. Le and H. Zhang, "Log parsing: How far can chatgpt go?," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 1699–1704, IEEE, 2023.

[19] Z. Jiang, J. Liu, Z. Chen, Y. Li, J. Huang, Y. Huo, P. He, J. Gu, and M. R. Lyu, "Lilac: Log parsing using llms with adaptive parsing cache," *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 137–160, 2024.

[20] J. Xu, R. Yang, Y. Huo, C. Zhang, and P. He, "Divlog: Log parsing with prompt enhanced in-context learning," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pp. 1–12, 2024.

[21] A. Lyons, J. Gamba, A. Shawaga, J. Reardon, J. Tapiador, S. Egelman, and N. Vallina-Rodriguez, "Log: It's big, It's heavy, It's filled with personal data! measuring the logging of sensitive information in the android ecosystem," in *32nd USENIX Security Symposium (USENIX Security 23)*, (Anaheim, CA), pp. 2115–2132, USENIX Association, Aug. 2023.

[22] Y. Wu, B. Chai, S. Yu, Y. Li, P. He, W. Jiang, and J. Li, "Log-ptr: Variable-aware log parsing with pointer network," *arXiv preprint arXiv:2401.05986*, 2024.

[23] Z. Jiang, J. Liu, J. Huang, Y. Li, Y. Huo, J. Gu, Z. Chen, J. Zhu, and M. R. Lyu, "A large-scale evaluation for log parsing techniques: How far are we?," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2024.

[24] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pp. 117–132, 2009.

[25] M. Nagappan, K. Wu, and M. A. Vouk, "Efficiently extracting operational profiles from execution logs using suffix arrays," in *2009 20th International Symposium on Software Reliability Engineering*, pp. 41–50, IEEE, 2009.

[26] R. Vaarandi, "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)*, pp. 119–126, Ieee, 2003.

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

[28] R. Vaarandi and M. Pihelgas, "Logcluster-a data clustering and pattern mining algorithm for event logs," in *2015 11th International conference on network and service management (CNSM)*, pp. 1–7, IEEE, 2015.

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

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

[31] K. Shima, "Length matters: Clustering system log messages using length of words," *arXiv preprint arXiv:1611.03213*, 2016.

[32] M. Mizutani, "Incremental mining of system log format," in *2013 IEEE International Conference on Services Computing*, pp. 595–602, IEEE, 2013.

[33] Z. M. Jiang, A. E. Hassan, P. Flora, and G. Hamann, "Abstracting execution logs to execution events for enterprise applications (short paper)," in *2008 The Eighth International Conference on Quality Software*, pp. 181–186, IEEE, 2008.

[34] S. Yu, P. He, N. Chen, and Y. Wu, "Brain: Log parsing with bidirectional parallel tree," *IEEE Transactions on Services Computing*, 2023.

[35] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "Electra: Pre-training text encoders as discriminators rather than generators," in *International Conference on Learning Representations*, 2020.

[36] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, 2019.

[37] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "CodeBERT: A pre-trained model for programming and natural languages," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, (Online), pp. 1536–1547, Association for Computational Linguistics, Nov. 2020.

[38] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. LIU, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu, M. Tufano, S. K. Deng, C. Clement, D. Drain, N. Sundaresan, J. Yin, D. Jiang, and M. Zhou, "Graphcodebert: Pre-training code representations with data flow," in *International Conference on Learning Representations*, 2021.

[39] Y. Xiao, V.-H. Le, and H. Zhang, "Demonstration-free: Towards more practical log parsing with large language models," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ASE '24, (New York, NY, USA), p. 153–165, Association for Computing Machinery, 2024.

[40] J. Huang, Z. Jiang, Z. Chen, and M. R. Lyu, "Ulog: Unsupervised log parsing with large language models through log contrastive units," *arXiv preprint arXiv:2406.07174*, 2024.

[41] C. Pei, Z. Liu, J. Li, E. Zhang, L. Zhang, H. Zhang, W. Chen, D. Pei, and G. Xie, "Self-evolutionary group-wise log parsing based on large language model," in *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 49–60, IEEE, 2024.

[42] H. Ju, "Reliable online log parsing using large language models with retrieval-augmented generation," in *2024 IEEE 35th International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 99–102, IEEE, 2024.

[43] Z. Ma, A. R. Chen, D. J. Kim, T.-H. Chen, and S. Wang, "Llmparser: An exploratory study on using large language models for log parsing," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pp. 1–13, 2024.

[44] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020.

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

[46] X. Li, H. Zhang, V. Le, and P. Chen, "Logshrink: Effective log compression by leveraging commonality and variability of log data," in *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*, (Los Alamitos, CA, USA), pp. 243–254, IEEE Computer Society, apr 2024.

[47] C. Wang, Y. Yang, C. Gao, Y. Peng, H. Zhang, and M. R. Lyu, "No more fine-tuning? an experimental evaluation of prompt tuning in code intelligence," in *Proceedings of the 30th ACM joint European software engineering conference and symposium on the foundations of software engineering*, pp. 382–394, 2022.

[48] Z. A. Khan, D. Shin, D. Bianculli, and L. Briand, "Guidelines for assessing the accuracy of log message template identification techniques," in *Proceedings of the 44th International Conference on Software Engineering*, pp. 1095–1106, 2022.

[49] X. Li, P. Chen, L. Jing, Z. He, and G. Yu, "Swisslog: Robust and unified deep learning based log anomaly detection for diverse faults," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, pp. 92–103, IEEE, 2020.

[50] LogPAI, "A large collection of system log datasets for ai-powered log analytics." https://github.com/logpai/loghub. Accessed: August 01, 2024.

[51] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *International Conference on Learning Representations*, 2018.

[52] P. He, X. Liu, J. Gao, and W. Chen, "Deberta: Decoding-enhanced bert with disentangled attention," in *International Conference on Learning Representations*, 2021.

[53] A. Mok, "How much does chatgpt cost to run?." https://www.businessinsider.com/how-much-chatgpt-costs-openai-to-run-estimate-report-2023-4. Accessed: July 20, 2024.

[54] A. A. Chien, L. Lin, H. Nguyen, V. Rao, T. Sharma, and R. Wijayawardana, "Reducing the carbon impact of generative ai inference (today and in 2035)," in *Proceedings of the 2nd workshop on sustainable computer systems*, pp. 1–7, 2023.

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

[56] L. Chen, M. Zaharia, and J. Zou, "Analyzing chatgpt's behavior shifts over time," in *R0-FoMo: Robustness of Few-shot and Zero-shot Learning in Large Foundation Models*, 2023.

[57] V.-H. Le and H. Zhang, "Log-based anomaly detection with deep learning: How far are we?," in *Proceedings of the 44th International Conference on Software Engineering*, pp. 1356–1367, 2022.