MicLog: Towards Accurate and Efficient LLM-based Log Parsing via Progressive Meta In-Context Learning

Jianbo Yu[1], Yixuan Li[2], Hai Xu[3], Kang Xu[4], Junjielong Xu[5], Zhijing Li[6,7], Pinjia He[5], Wanyuan Wang[1*]

[1]School of Computer Science and Engineering, Southeast University, China
[2]College of Computing and Data Science, Nanyang Technological University, Singapore
[3]Focus Technology Co., Ltd., China
[4]School of Computer Science, Nanjing University of Posts and Telecommunications, China
[5]School of Data Science, The Chinese University of Hong Kong, Shenzhen (CUHK-Shenzhen), China
[6]School of Artificial Intelligence, Shenzhen University, China
[7]National Engineering Laboratory for Big Data System Computing Technology, Shenzhen University, China
{jianboyu,wywang}@seu.edu.cn, yixuan.li@ntu.edu.sg, xuhai@focuschina.com, kxu@njupt.edu.cn, junjielongxu@link.cuhk.edu.cn, hepinjia@cuhk.edu.cn, lizhijing@szu.edu.cn

Abstract

Log parsing converts semi-structured logs into structured templates, forming a critical foundation for downstream analysis. Traditional syntax and semantic-based parsers often struggle with semantic variations in evolving logs and data scarcity stemming from their limited domain coverage. Recent large language model (LLM)-based parsers leverage in-context learning (ICL) to extract semantics from examples, demonstrating superior accuracy. However, LLM-based parsers face two main challenges: 1) underutilization of ICL capabilities, particularly in dynamic example selection and cross-domain generalization, leading to inconsistent performance; 2) time-consuming and costly LLM querying. To address these challenges, we present MicLog, the first progressive meta in-context learning (ProgMeta-ICL) log parsing framework that combines meta-learning with ICL on small open-source LLMs (i.e., Qwen-2.5-3B). Specifically, MicLog: i) enhances LLMs' ICL capability through a zero-shot to k-shot ProgMeta-ICL paradigm, employing weighted DBSCAN candidate sampling and enhanced BM25 demonstration selection; ii) accelerates parsing via a multi-level pre-query cache that dynamically matches and refines recently parsed templates. Evaluated on Loghub-2.0, MicLog achieves 10.3% higher parsing accuracy than the state-of-the-art parser while reducing parsing time by 42.4%.

Introduction

In modern software systems, log data is crucial for maintenance and monitoring. These systems generate vast quantities of log messages, serving as indispensable resources for subsequent tasks including anomaly detection (Du et al. 2017; He et al. 2016; Zhang et al. 2019), root cause analysis (Amar and Rigby 2019; He et al. 2018; Lin et al. 2016; Huang et al. 2024b), and vulnerability prediction (Bilgin et al. 2020; Han et al. 2017). However, the immense scale and complexity of logs make manual analysis impractical (He et al. 2021), necessitating automated log parsing techniques. Log parsing, a fundamental step in log analysis, transforms messages into structured formats by extracting: 1) *log templates* (consistent parts from logging statements), and 2) *log parameters* (dynamic parts that vary per execution). As shown in Figure 1, the logging statement `client.LOG.info(f"session closed for user {user_name}")` generates messages like `"session closed for user cyrus"`, where the template is `"session closed for user <*>"` and the parameter is `"cyrus"`.

Industrial systems, such as black-box microservices, IoT devices, and proprietary SaaS tools often produce logs with unknown formats, making parsing essential for reconstructing structures without access to the logging statements. Given the impracticality of accessing source code, various automated parsers have been developed: (1) Syntax-based parsers (Dai et al. 2020; Du and Li 2016; He et al. 2017; Yu et al. 2023) use pattern recognition and clustering without prior format knowledge (2) Semantic-based parsers (Huo et al. 2023; Le and Zhang 2023; Liu et al. 2022) leverage labeled data to train classification models (3) Large language model (LLM) based parsers (Xu et al. 2024b; Jiang et al. 2024a; Xiao, Le, and Zhang 2024; Huang et al. 2024a; Ma, Kim, and Chen 2024; Wu, Yu, and Li 2024) utilize LLMs to capture complex textual patterns. These tools assist engineers in completing this critical initial step of log analysis.

While each log parsing approach has strengths, significant limitations persist. Syntax-based parsers struggle with highly variable formats, leading to inaccurate parsing. Semantic-based parsers require extensive labeled data and generalize poorly to unseen log formats. Although LLM-based parsers leverage in-context learning (ICL) for improved performance, recent studies (Le and Zhang 2023; Xu et al. 2024b) reveal they underutilize ICL capabilities for complex logs. Additionally, the time-consuming and costly nature of LLM querying, coupled with privacy concerns arising from leveraging commercial models like GPT to process sensitive log data, hinders their industrial deployment.

*Corresponding Author.

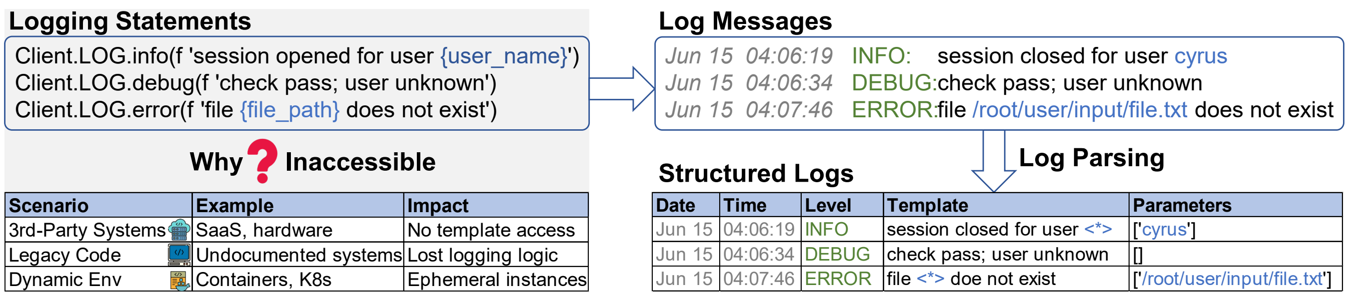

Figure 1: A simple process of Log Parsing.

To address these limitations, we propose MicLog, an effective and efficient progressive meta in-context learning (ProgMeta-ICL) log parsing framework that leverages ProgMeta-ICL to enhance the ICL capabilities of LLMs in log parsing tasks. Specifically, MicLog comprises three main components: a weighted DBSCAN sampler, a progressive meta in-context training module, and a multi-level cache-enhanced LLM ICL log parser (MLCELI-Parser). The weighted DBSCAN sampler employs a carefully designed algorithm to extract samples from open-source datasets based on a predefined sampling ratio after log preprocessing. These extracted samples serve two purposes: (1) First, they are used for progressive meta in-context training to enhance ICL capabilities; (2) Second, the MLCELI-Parser employs them to construct ICL prompts when no matching templates are found in the cache. Before querying the LLM, MLCELI-Parser matches raw logs against cached templates and updates the cache with new templates derived from LLM query results to improve efficiency.

MicLog has been thoroughly evaluated on all 14 public datasets of Loghub-2.0 (Jiang et al. 2024b). The results show that MicLog achieves the highest average accuracy on all performance metrics, achieving (1) 97.6% Parsing Accuracy, (2) 95.3% Precision Template Accuracy, and (3) 90.5% Recall Template Accuracy when using same-source prompt examples. This outperforms the current state-of-the-art method AdaParser (Wu, Yu, and Li 2024) by 10.3%, 12.6%, and 6.1%, respectively. Moreover, driven by its multi-level cache mechanism, MicLog achieves a 42.4% reduction in total parsing time compared to AdaParser, while also surpassing the most efficient baseline Drain (He et al. 2017). The evaluation results demonstrate that MicLog is an effective and efficient log parsing framework in real-world deployment.

This paper presents the following key contributions:

- We propose MicLog, the first ProgMeta-ICL log parsing framework, which effectively addresses the limitations of existing LLM-based parsers.
- We introduce a meta in-context training paradigm that facilitates efficient meta-learning, gradually transitioning from zero-shot to few-shot learning. This enables LLMs to improve their ICL capabilities.
- We propose a multi-level cache-enhanced LLM ICL log parser (MLCELI-Parser) to matches input logs against cached templates and updates the cache with new templates derived from LLM ICL outputs when misses occur.
- We present the evaluation of MicLog on public datasets using three different performance evaluation metrics. The results show that MicLog achieves state-of-the-art performance, surpassing existing LLM-based parsers.

Related Work

Log Parsing

Log parsing, the foundational phase of automated log analysis (He et al. 2021; Zhu et al. 2019), extracts templates from raw messages by distinguishing variables from constants to produce structured logs. As Figure 1 illustrates, parsers first extract headers (timestamps and verbosity levels) using regular expressions (Li et al. 2024b) due to their predictable structure, with research primarily focusing on deriving templates and parameters from log message bodies. Industrial challenges arise from growing log volumes and evolving template complexity, where source code access would facilitate constant extraction but security and privacy constraints often prohibit this. Consequently, automated parsers have emerged—syntax-based (Shima 2016; Dai et al. 2020; Du and Li 2016; He et al. 2017; Yu et al. 2023), semantic-based (Huo et al. 2023; Le and Zhang 2023; Liu et al. 2022), and LLM-based (Xu et al. 2024b; Jiang et al. 2024a; Xiao, Le, and Zhang 2024; Huang et al. 2024a; Ma, Kim, and Chen 2024; Wu, Yu, and Li 2024; Zhong et al. 2024)—yet these exhibit limitations: syntax-based parsers rely on domain-specific features that may misalign with actual log content; semantic-based parsers struggle to generalize across highly diverse datasets; and LLM-based parsers, despite improved robustness, still underperform on complex datasets with inadequate robustness metrics. We therefore posit that fully leveraging LLMs' ICL potential is essential to enhance accuracy and robustness for complex log parsing tasks.

Large Language Models and In-Context Learning

LLMs (Zhao et al. 2023; Yao et al. 2024), trained on massive corpora via self-supervised learning, have revolutionized NLP with capabilities in text generation, language understanding, and complex reasoning, capturing rich semantic and syntactic patterns within their billion-parameter architectures. A key capability enabling their flexibility is ICL (Dong et al. 2022; Min et al. 2022; Akyürek et al. 2022;

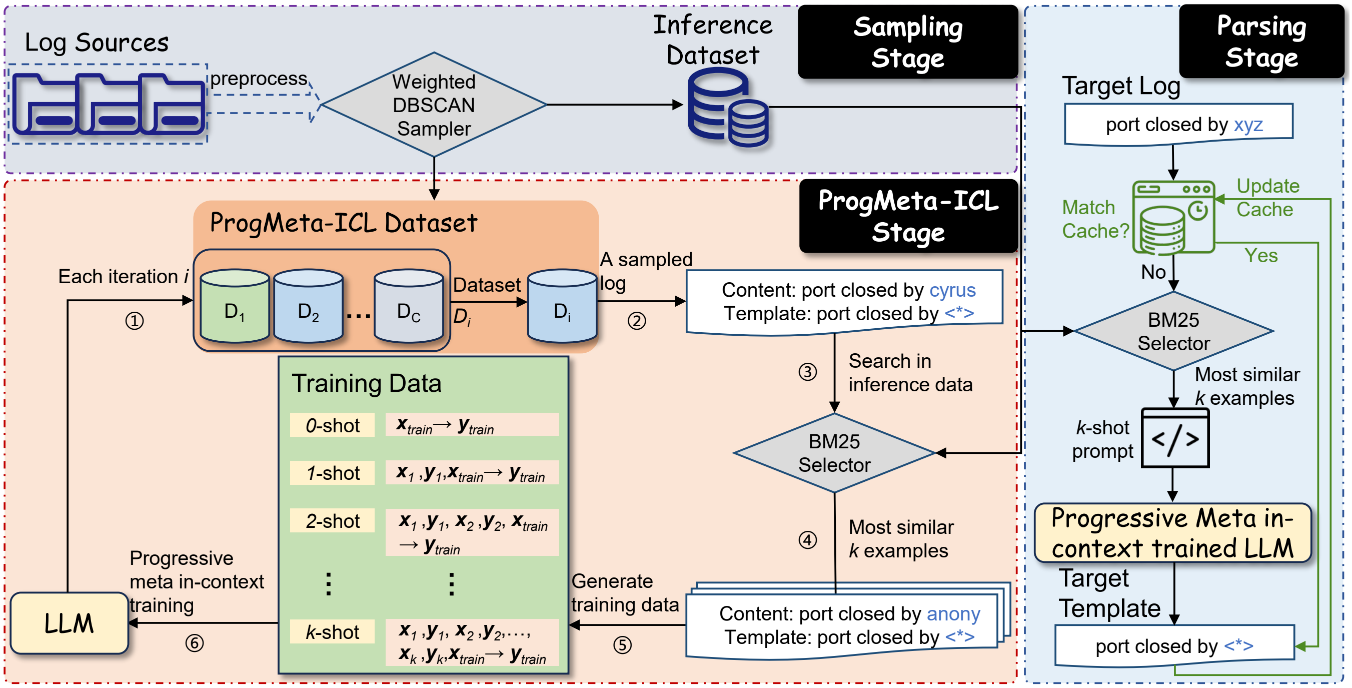

Figure 2: The workflow of MicLog framework. Sampling and ProgMeta-ICL are performed before online log parsing. The ProgMeta-ICL dataset and inference dataset need to be labeled before progressive meta in-context training and log parsing.

Xu et al. 2024a; Li, Wang, and Ke 2023), which allows LLMs to adapt to tasks without fine-tuning. By providing few-shot examples directly within prompts, LLMs infer patterns and generate appropriate responses while leveraging pre-trained knowledge. This approach eliminates the need for extensive labeled training data and has proven effective across diverse NLP tasks including sentiment analysis, QA, and code generation. For the critical automation task of log parsing, where traditional syntax or semantic-based approaches struggle with real-world complexity and diversity, LLMs with ICL offer a promising alternative by leveraging their natural language understanding to handle log semantics, adapting to new formats via example-based prompting, and operating effectively with minimal labeled data. When provided with log-template examples in prompts, LLMs can accurately parse unseen logs, an adaptability particularly valuable for heterogeneous systems with significantly varying log structures, as it eliminates the need for retraining.

Meta In-Context Learning

While ICL enables LLMs to adapt to tasks using few-shot examples in prompts, its effectiveness diminishes when task inputs are low-quality or inherently complex. Recent studies (Min et al. 2021; Chen et al. 2021; Li et al. 2024a; Coda-Forno et al. 2023) address this through meta in-context learning (Meta-ICL), which trains models explicitly to enhance ICL capabilities via multitask learning using fixed-shot examples. However, existing methods rely on fixed example counts per task during meta-training, which is suboptimal as tasks vary in their example requirements. To address the limitations of Meta-ICL, we extend this paradigm with ProgMeta-ICL: a flexible, dynamic approach featuring a two-stage training process. First, the model is trained across diverse tasks under few-shot conditions to capture foundational patterns. Subsequently, examples per log parsing task are progressively increased to refine task-specific understanding. This staged exposure enables adaptive learning, enhances complex task performance and improves robustness across diverse scenarios.

Method

In this section, we introduce MicLog, a ProgMeta-ICL LLM-based log parsing framework. As illustrated in Figure 2, MicLog is composed of three components, and its overall procedure is summarized in Algorithm 1.

Weighted DBSCAN Sampler

Both the ProgMeta-ICL and parsing stages begin by sampling a small subset of candidate log-template pairs. This subset must be diverse to avoid LLM overfitting and sufficiently representative to cover varied logs and their key characteristics.

After deduplication in preprocess, we propose a weighted DBSCAN sampling method to extract small, diverse, and representative log subsets. DBSCAN (Ester et al. 1996; Schubert et al. 2017) clusters densely packed points using parameters ϵ (neighborhood radius) and $MinPts$ (minimum cluster density), labeling sparse points as noise. We leverage DBSCAN's core points for clustering and incorporate weighting to efficiently sample representative logs. For each log $l \in \mathcal{D}$, complexity is computed via Equation 1.

Progressive meta in-context training	**Task:** C progressive meta in-context training tasks
	Data given: Training examples $T_i = \{(x_j^i, y_j^i)\}_{j=1}^{N_i}, \forall i \in [1, C] (N_i \gg K)$, max shot number K
	Objective: For each iteration, 1. Sample task $i \in [1, C]$ 2. Iterate over shot number k from 0 to K 3. Sample k examples from T_i: $(x_1, y_1), \cdots, (x_k, y_k)$ 4. Maximize $P(y_{train} \mid x_1, y_1, \cdots, x_k, y_k, x_{train})$
Inference	**Task:** An unseen *target* task
	Data given: Training examples $(x_1, y_1), \cdots, (x_k, y_k)$, Test input x_{target}
	Objective: $\arg\max_{c \in \mathcal{C}} P(c \mid x_1, y_1, \cdots, x_k, y_k, x_{target})$

Table 1: Progressive meta in-context training and inference process

Algorithm 1: MicLog: Progressive Meta ICL Log Parsing

```
Require: Raw log dataset 𝒟, Initial LLM parameters θ, Max shot number k, Sampling ratio α, LRU cache capacity Cache_LRU, Pattern cache Cache_Pattern
Ensure: Parsed templates 𝒯, Meta-trained LLM parameters θ*
Stage 1: Sampling
𝒟_dedup ← deduplicate(𝒟)
𝒮_meta, 𝒮_inf ← WeightedDBSCAN(𝒟_dedup, α)
Stage 2: ProgMeta-ICL
for shot = 0 to k do
    Sample task batch ℬ ∼ 𝒮_meta
    Concatenate shot examples: prompt ← {(x_j, y_j)}_{j=1}^{shot}
    Update θ ← θ − ∇ℒ_ProgMeta-ICL
end for
θ* ← θ
Stage 3: Parsing with Multi-Level Cache
for each raw log l_i ∈ 𝒟 do
    if l_i ∈ Cache_LRU then
        t_i ← Cache_LRU[l_i]
    else if ∃t_p ∈ Cache_Pattern s.t. validate(l_i, t_p) then
        t_i ← t_p, Update Cache_LRU with (l_i, t_i)
    else
        Retrieve top-k logs ℛ ← BM25(l_i, 𝒮_inf)
        Construct ICL prompt P ← {ℛ, l_i}, t_i ← LLM_θ*(P)
        Update Cache_LRU, Cache_Pattern with (l_i, t_i)
    end if
    𝒯 ← 𝒯 ∪ {t_i}
end for
```

$$\textit{complexity(l)} = {token_l}^{token_l} + length_l \tag{1}$$

Here, $token_l$ equals to the number of token of log l and $length_l$ means log's length. To prevent numerical instability, we introduce a constant smoothing factor $factor_s$ for computing weights that incorporate each log's complexity.

$$w = \frac{\textit{complexity}(l) + factor_s}{\sum_{l_i \in \mathcal{D}} [\textit{complexity}(l_i) + factor_s]} \tag{2}$$

DBSCAN also introduces neighborhood radius and minimum points to assist in completing the clustering process. Besides, the complexity is also utilized here to calculate the Euclidean distance of the complexity of a log. After clustering concludes, samples are extracted from each cluster in accordance with the weight computed using Equation 2, based on the specified sample ratio. We reuse this algorithm and obtained two datasets, one is ProgMeta-ICL dataset to complete meta in-context training, and the other is the inference dataset for generating ICL prompt.

Progressive Meta In-Context Training

Table 1 provides an overview of the progressive meta in-context training process in MicLog. The key idea is to use a multi-task learning scheme over a large collection of meta-training tasks, in order for the LLMs to learn how to condition on a small set of ICL examples, understand the core content of the current task from it, and provide the corresponding log template output. The meta in-context training examples are concatenated as a single input to the LLM, which sequentially presents the 0-shot to k-shot (e.g., $k = 5$) learning procedure. At test time, the meta in-context trained LLM is evaluated on unseen target tasks that come with k training examples, and inference directly follows the same data format as in meta-training.

By analyzing the work of *Zhang et al.* (Zhang et al. 2023), in the ProgMeta-ICL stage, the goal is to minimize the meta-loss across a range of tasks $\mathcal{T}$, where each task $\mathcal{T}_i$ requires its own ICL setting. Meta-learning aims to train a model that can rapidly adapt to new tasks by leveraging prior learning. For each task $\mathcal{T}_i$ drawn from a meta-distribution $\mathcal{P}(\mathcal{T})$, let $S_T^{(\mathcal{T}_i)}$ denotes the demonstration set for task $\mathcal{T}_i$, and the meta-objective is to minimize the expected ICL loss over tasks:

$$\mathcal{L}_{ProgMeta-ICL} = E_{\mathcal{T} \sim \mathcal{P}(\mathcal{T})} \left[\mathcal{L}_{ICL}(\mathcal{T}, S_T^{(\mathcal{T})}) \right] \tag{3}$$

The meta-learner aims to minimize this loss across tasks by learning a shared representation θ. For a meta trained LLM, when facing any new task $\mathcal{T}_j$ by leveraging ICL paradigm, the model can quickly adapt to minimize:

$$\mathcal{L}_{ICL}(\mathcal{T}_j) = \frac{1}{T} \sum_{t=1}^{T} -\log P(r_t \mid x_t, S_{t-1}, \theta) \tag{4}$$

where r_t represents the verbalizer associated with input x_t for task $\mathcal{T}_i$, x_t is the input at step t, S_{t-1} is the set of input-label pairs up to step $t-1$. The parameter θ is optimized by the meta-learner to be task-agnostic and it can quickly adapt for different tasks.

Multi-level Cache Enhanced LLM ICL Log Parser

MicLog employs a multi-level caching mechanism to minimize redundant LLM invocations and accelerate parsing by exploiting structural patterns in log streams. The cache architecture consists of two synergistic components:

- **LRU Cache:** Maintains recently parsed $\langle$raw log, template$\rangle$ pairs in an ordered dictionary with fixed capacity $Cache_{\text{LRU}}$. Entries satisfy $\|l_i\|_{\text{tokens}} \leq \tau$, with least-recently-used eviction when exceeding capacity:

$$\text{Cache}_{\text{LRU}} = \{(l_i, t_i) \mid i \in [1, C_{\text{LRU}}]\} \tag{5}$$

- **Pattern Cache:** Stores template patterns t_p for structural matching, where $\eta(\cdot)$ performs pattern normalization:

$$\text{Cache}_{\text{Pattern}} = \{t_p \mid t_p = \eta(t_{\text{parsed}})\} \tag{6}$$

Cache lookup follows a two-stage process. First, the raw log l_{raw} is checked against the LRU cache for an exact match. If found, the corresponding template is returned immediately. On LRU miss, l_{raw} is normalized and compared against all patterns in the pattern cache. The validation function validate(l_{norm}, t_p) decomposes t_p into constant segments $\{s_j\}$ separated by "<*>", verifying their ordered occurrence in l_{norm} with strictly increasing positions pos_j:

$$\forall s_j \in \text{segments}(t_p),\ \exists\, \text{pos}_j \mid l_{\text{norm}}[\text{pos}_j : \text{pos}_j + \|s_j\|] = s_j \tag{7}$$

This design achieves $\mathcal{O}(1)$ exact-match lookups and $\mathcal{O}(k)$ pattern-match lookups ($k = \|\text{Cache}_{\text{Pattern}}\|$). Upon successful pattern match, the template t_p is cached in the LRU and returned. On full cache miss, newly generated templates, which returned by the LLM ICL log parser, update both caches.

Our LLM-based log parser begins with constructing effective prompts. We select k semantically relevant logs from the inference dataset. Studies demonstrate that example selection critically impacts LLM performance (Rubin, Herzig, and Berant 2021), with recent work (Xu et al. 2024b) showing that ordering k-shot examples by ascending similarity maximizes log parsing accuracy.

We implement an enhanced BM25 algorithm (Robertson, Zaragoza et al. 2009) for efficient similarity search. For a log corpus $\mathcal{D}$ containing N entries, we compute the Inverse Document Frequency (IDF) (Robertson 2004) for each w as:

$$\text{IDF}(w) = \log\left(\frac{N - f(w) + 0.5}{f(w) + 0.5} + 1\right) \tag{8}$$

where $f(w)$ denotes the document frequency of w. The BM25 score between query log q and candidate log d is:

$$\text{BM25} = \sum_{w \in q} \text{IDF}(w) \cdot \frac{\text{TF}(w,d) \cdot (k_1 + 1)}{\text{TF}(w,d) + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{\text{avg\_l}}\right)} \tag{9}$$

where $\text{TF}(w,d)$ is w's term frequency in d, $|d|$ is log length, avg_l is average log length in $\mathcal{D}$, and k_1, b are tunable parameters controlling term frequency saturation and length normalization.

The top k candidates identified by BM25 are sorted in descending similarity order to generate the prompt. The parsed log template for the current raw log is retrieved via the meta in-context trained LLM query.

Evaluation

In this section, we outline the experimental setup, followed by the evaluation results and analysis conducted on public datasets to address the following research questions:
RQ1: How effective and stable is MicLog?
RQ2: How does each component contribute to MicLog?
RQ4: How efficient is MicLog?
RQ4: How do different training strategies affect MicLog?

Experiment Setup

Datasets. Our experiments utilize Loghub-2.0 (Jiang et al. 2024b), a comprehensive log parsing datasets from LogPAI (Zhu et al. 2019). Loghub-2.0 contains 14 datasets of system logs from diverse sources such as distributed systems, supercomputer systems, and server-side applications, totaling over 50 million log messages and 3,488 log templates.

Baselines. We select Drain (He et al. 2017), Brain (Yu et al. 2023), LogPPT (Le and Zhang 2023), LUNAR (Huang et al. 2024a), LibreLog (Ma, Kim, and Chen 2024), LILAC (Jiang et al. 2024a) and AdaParser (Wu, Yu, and Li 2024) as our baselines for comparison. The first two parsers demonstrate superior performance among all syntax-based parsers. The semantic-based parser, LogPPT leverages template-free prompt-tuning (Ma et al. 2021) to fine-tune a pre-trained language model, RoBERTa (Liu 2019). LLM-based paser LUNAR leverages log contrastive units to facilitate effective comparisons by the LLM. LibreLog uses open-source LLMs to parse logs by syntactic similarity in the static text. LILAC utilizes the ICL capability to adapt LLMs to parse various log data with adaptive parsing cache. AdaParser improves parsing accuracy based on SG-ICL and selfcorrection. Due to the unavailability of the original GPT-3 versions (Brown 2020) required by LUNAR, LibreLog, LILAC and AdaParser, we replicate their experiments using *gpt-3.5-turbo-0125* for comparative analysis.

Metrics. In line with recent studies (Khan et al. 2022; Xu et al. 2024b), our evaluation employs three metrics: Parsing Accuracy (PA), Precision Template Accuracy (PTA), and Recall Template Accuracy (RTA), with the latter two metrics collectively referred to as Template Accuracy (TA). The definitions of these metrics are as follows:

- Parsing Accuracy (PA) is used to evaluate the ability to correctly extract log templates, defined as the ratio of correctly parsed log messages to the total number of logs.
- Template Accuracy (TA) is a template-level metric calculated based on the proportion of correctly identified templates. Using the number of correctly identified templates (N_c), identified templates (N_i) and ground-truth templates (N_g), we calculate the Precision ($PTA = \frac{N_c}{N_i}$) and Recall ($RTA = \frac{N_c}{N_g}$).

Dataset	Drain			Brain			LogPPT			LUNAR			LibreLog			LILAC			AdaParser			MicLog		
	PA	PTA	RTA	PA	PTA	RTA	PA	PTA	RTA	PA	PTA	RTA	PA	PTA	RTA	PA	PTA	RTA	PA	PTA	RTA	PA	PTA	RTA
Apache	72.3	48.3	50.0	28.7	45.2	50.0	94.8	39.8	34.6	50.8	69.0	69.0	99.6	85.4	92.9	99.5	82.8	82.8	99.9	93.1	93.1	**100.0**	**100.0**	**100.0**
BGL	40.7	16.4	21.3	40.2	15.9	22.5	93.8	30.5	22.4	60.6	79.0	82.2	92.9	86.4	84.1	95.3	73.9	80.6	98.0	80.8	79.1	**99.0**	**92.5**	**85.3**
Hadoop	51.2	32.0	46.0	14.1	15.2	29.5	66.6	49.6	38.4	85.9	73.4	75.8	87.1	87.9	87.7	83.9	77.2	75.8	**95.5**	83.4	85.2	94.0	**97.9**	**96.6**
HDFS	62.1	58.7	58.7	92.9	63.4	56.5	94.3	28.0	35.2	87.4	97.8	97.8	**100.0**	98.4	98.4	**100.0**	95.7	97.8	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
HealthApp	18.3	0.2	36.6	17.1	29.4	35.8	99.7	82.2	82.2	98.2	86.9	**89.1**	97.4	87.9	90.4	73.6	85.0	87.2	86.4	88.5	**89.1**	**100.0**	**93.9**	88.5
HPC	72.1	9.0	46.8	66.3	14.4	48.4	99.7	70.8	81.4	99.0	69.7	93.2	97.3	88.4	81.9	70.3	77.3	78.4	80.3	**89.2**	**89.2**	**99.6**	80.0	54.1
Linux	9.9	17.7	25.3	8.7	21.1	29.2	16.8	26.8	52.3	84.3	72.6	76.0	90.2	89.6	90.4	84.2	73.9	73.7	73.2	75.0	76.3	**99.1**	**96.6**	**92.9**
Mac	28.1	3.8	25.2	32.4	24.9	33.4	39.0	25.6	32.6	59.2	55.4	56.5	65.4	50.2	60.5	68.5	54.0	64.1	57.9	59.4	58.8	**95.4**	**96.4**	**94.4**
OpenSSH	58.5	37.5	42.9	48.1	26.5	37.1	65.4	8.1	14.3	69.1	80.5	86.8	49.6	34.8	30.6	**96.8**	85.7	78.9	94.2	92.3	**94.7**	94.2	**94.6**	92.1
OpenStack	2.9	0.1	14.6	14.1	29.2	29.2	40.6	70.4	78.3	94.2	82.4	87.5	83.1	72.9	68.2	**100.0**	93.8	93.8	**100.0**	97.9	97.9	**100.0**	**100.0**	**100.0**
Proxifier	68.8	8.8	45.5	70.3	87.5	63.6	**100.0**	95.7	95.7	51.1	72.7	72.7	89.7	90.5	86.1	**100.0**	90.9	90.9	98.4	91.7	**100.0**	**100.0**	**100.0**	**100.0**
Spark	39.4	38.8	42.6	39.3	5.5	38.7	95.2	36.0	27.8	97.0	70.1	74.6	88.9	70.5	62.7	97.3	80.4	78.4	98.2	80.3	79.2	**100.0**	**93.8**	**90.3**
Thunderbird	21.6	4.0	26.6	26.1	23.7	30.3	40.1	13.5	9.2	63.5	60.7	65.6	69.4	52.0	48.6	62.6	46.4	64.1	64.4	62.3	63.1	**85.6**	**92.9**	**81.6**
Zookeeper	84.3	64.9	57.5	82.2	57.4	62.1	84.5	80.9	80.9	71.2	88.3	76.4	85.0	80.1	78.9	68.5	89.7	87.6	92.0	90.8	88.8	**99.3**	**95.3**	**91.0**
Average	45.0	24.3	38.5	41.5	32.8	40.5	73.6	47.0	49.0	76.5	75.6	78.8	85.4	76.8	75.8	85.8	79.1	81.0	88.5	84.6	85.3	**97.6**	**95.3**	**90.5**

Table 2: Performance of Various Log Parsing Methods on Loghub-2.0 Datasets (%)

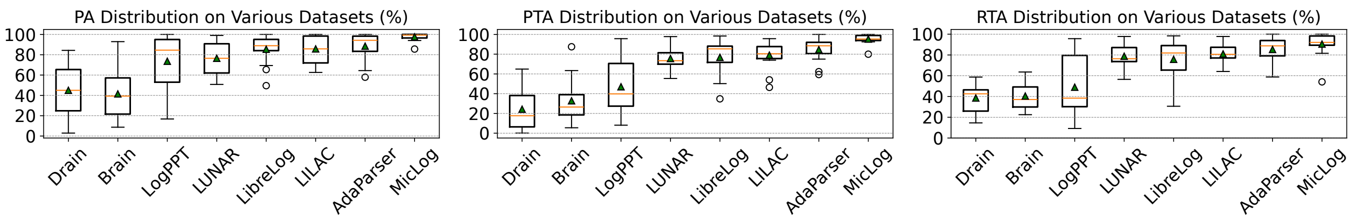

Figure 3: Robustness comparison between baselines and MicLog on public datasets (%)

Environment and Implementation. We conduct experiments using the local open-source LLM Qwen2.5-3B (Yang et al. 2024) for meta-learning and ICL inference on an RTX 4090 GPU to obtain raw prompt responses. We also employed Python 3.10 to implement the weighted DBSCAN sampler, BM25 selector, multi-level cache, and evaluation scripts on Ubuntu 22.04 LTS server. For meta in-context training, MicLog samples only 0.009% (326 logs avg.) logs from Loghub-2.0 for maximum training efficiency. During ICL inference, 5 labeled examples per query are selected from the candidate set for prompt construction. We repeated each experimental configuration five times and calculated their mean as the final result. All configurations remain fixed throughout evaluation.

RQ1: Effectiveness

In this section, we perform a comprehensive evaluation of the accuracy and robustness of MicLog, along with the baselines, across all datasets from Loghub-2.0. We assess the accuracy of the parsers using PA, PTA, and RTA, and evaluate their robustness by analyzing the statistical distribution.

Accuracy. Table 2 presents comprehensive evaluation results on Loghub-2.0, with best metrics per dataset in **bold**. MicLog averages 97.6% PA, 95.3% PTA, and 90.5% RTA, outperforming the SOTA pasers AdaParser by 10.3%, 12.6%, and 6.1%, respectively. To assess if MicLog significantly outperforms AdaParser, we performed one-sided Wilcoxon signed-rank tests (Rey and Neuhäuser 2011) for the three metrics across 14 datasets. The p-values (0.0038, 0.0036, 0.0356) are all below 0.05 and test statistics are (63.0, 84.0, 62.0), providing strong evidence for MicLog's superior performance.

Test Condition	PA	PTA	RTA
MicLog *epoch=5*	97.6	95.3	90.5
MicLog *epoch=1*	96.3 (↓1.3)	93.9 (↓1.4)	89.7 (↓0.8)
w/o ProgMeta-ICL	87.3 (↓10.3)	60.2 (↓35.1)	62.0 (↓28.5)
w/ k-means sampling	97.1 (↓0.5)	92.0 (↓3.3)	88.5 (↓2.0)
w/ random sampling	93.7 (↓3.9)	76.8 (↓18.5)	72.9 (↓17.4)
w/ LILAC cache	97.6 (-)	94.4 (↓0.9)	89.0 (↓1.5)
w/ random selection	94.7 (↓2.9)	79.1 (↓16.2)	73.4 (↓17.1)
w/ Proxifier only	95.8 (↓1.8)	90.2 (↓5.1)	87.1 (↓3.4)

Table 3: Average accuracy comparison on Loghub among MicLog with different strategies (%)

Robustness. Following recent work (Xu et al. 2024b), we compare the robustness of MicLog and baselines using box plots of three metrics across datasets. The orange line marks the median and the green triangle marks the mean. As shown in Figure 3, MicLog attains the highest accuracy and the smallest variance, reflected by its narrowest distribution. This demonstrates MicLog's strong robustness.

RQ2: Ablation Study

In this section, we conduct ablation study to discuss the contribution of each component in MicLog. We have developed seven variants of MicLog, the first one reset the hyperparam-

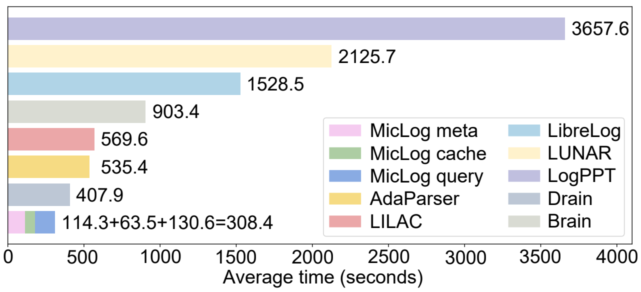

Figure 4: Efficiency of MicLog and baselines on Loghub-2.0

eter *epoch*, five of them were replaced based on the components of MicLog, and the last one was studied based on 1/14 of the training data, namely: 1) MicLog *epoch=1*: reset epoch from 5 to 1; 2) MicLog w/o ProgMeta-ICL: remove the ProgMeta-ICL design; 3) MicLog w/ k-means sampling: replace the weighted DBSCAN sampler by a k-means sampler; 4) MicLog w/ random sampling: replace the weighted DBSCAN sampler by a random sampler; 5) MicLog w/ LILAC cache: replace the multi-level cache by cache in LILAC; 6) MicLog w/ random selection: replace the BM25 selector by a random selector; 7) MicLog w/ Proxifier only: employ meta in-context training only in dataset Proxifier and test on all datasets.

The detailed evaluation results are shown in Table. 3, in which the following points can be made: (1) Progressive meta in-context train only 1 *epoch* leads to good enough result to surpass the SOTA method. (2) Removal of the ProgMeta-ICL module causes the most significant performance degradation, confirming its critical role in enhancing the LLM's in-context learning capability. (3) Substituting the weighted DBSCAN sampler with k-means yields marginal performance drops, while random sampling results in substantially larger reductions. (4) Caching hardly affects MicLog's accuracy, since this component is designed to optimize efficiency (discussed in RQ3). (5) Replacing the BM25 selector with random selection severely impacts average performance. (6) Generalization of ProgMeta-ICL paradigm: Proxifier represents a distinct distribution from other datasets. Our framework significantly enhances LLMs' ICL capabilities for log parsing using only Proxifier, achieving performance comparable to MicLog. This confirms our method genuinely improves ICL capabilities rather than dataset familiarity, and demonstrates effective generalization to diverse datasets with minimal training data.

RQ3: Efficiency

We evaluate MicLog and baselines on Logpub-2.0, recording average parsing times across datasets. The efficiency results are demonstrated in the Figure 4. The time cost analysis for MicLog incorporates both meta in-context training and parsing phase operations (cache retrieval and LLM queries). Results demonstrate that MicLog achieves lower average times across all Loghub-2.0 datasets than any baselines, reducing total parsing time by 42.4% compared to AdaParser while outperforming even the most efficient baseline, Drain.

Test Condition	PA	PTA	RTA
MicLog	97.6	95.3	90.5
w/ 0-shot only	94.4 (↓3.2)	85.4 (↓9.9)	79.4 (↓11.1)
w/ 1-shot only	94.6 (↓3.0)	86.8 (↓8.5)	79.7 (↓10.8)
w/ 3-shot only	94.6 (↓3.0)	85.7 (↓9.6)	80.1(↓10.4)
w/ 5-shot only	95.8 (↓1.8)	87.5 (↓7.8)	81.4 (↓9.1)
w/o ProgMeta-ICL	87.3 (↓10.3)	60.2 (↓35.1)	62.0 (↓28.5)

Table 4: Average accuracy comparison among MicLog with different shot meta in-context learning strategies (%)

Notably, MicLog's average cache time (63.5s) is substantially lower than LILAC's 376.5s (not list in the figure) despite both employing caching mechanisms. This efficiency stems from MicLog's multi-level architecture, which capitalizes on the temporal locality of log data - where identical or similar log entries frequently recur in short intervals. Our design significantly increases cache hit rates in such scenarios, accelerating parsing throughput. Moreover, by leveraging the open-source Qwen-3B LLM, our approach effectively addresses privacy concerns while achieving substantially lower training and inference costs than methods relying on proprietary models like the GPT-3.5 series or resource-intensive alternatives such as Llama3-8B.

RQ4: Impact of Different Training Stratgies

Previous Meta-ICL methods rely on fixed-shot demonstrations during meta-training, limiting adaptability to diverse task complexities. MicLog overcomes this constraint through 0-shot to k-shot ProgMeta-ICL demonstrations.

As shown in Table 4, MicLog utilizes meta-training data from 0-shot to 5-shot, while other variants rely solely on single-shot. All models are evaluated with 5-shot ICL prompts. Results indicate that LLMs meta-trained on any single shot (0/1/3/5) perform similarly across metrics and significantly outperform non-Meta-ICL baselines, showing that minimal meta-training substantially improves ICL utilization. Moreover, MicLog's progressive training yields further significant improvements, achieving over 90% accuracy on two TA metrics. This demonstrates its superior ability to build in-context understanding.

Conclusion

In this paper, we introduce MicLog, an effective and efficient log parsing framework that boosts LLMs ICL capability via ProgMeta-ICL. MicLog employs a progressive meta-learning process (from 0-shot to few-shot), supported by a weighted DBSCAN algorithm for sampling highly representative examples. Prior to LLM querying, a multi-level cache matches raw logs with existing templates and updates new templates from the LLM response on mismatches. Furthermore, we propose an efficient enhanced BM25 method for retrieving similar examples for prompt generation. Rigorous evaluation on benchmark datasets demonstrates that MicLog significantly outperforms existing parsers in both accuracy and robustness, highlighting its potential for log analysis research and practice.

Acknowledgements

This research was supported by the Key Research and Development Projects in Jiangsu Province (No. BE2021001-2), the National Natural Science Foundation of China (Nos. 62476121 and 12201619).

References

Akyürek, E.; Schuurmans, D.; Andreas, J.; Ma, T.; and Zhou, D. 2022. What learning algorithm is in-context learning? investigations with linear models. *arXiv preprint arXiv:2211.15661*.

Amar, A.; and Rigby, P. C. 2019. Mining historical test logs to predict bugs and localize faults in the test logs. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 140–151. IEEE.

Bilgin, Z.; Ersoy, M. A.; Soykan, E. U.; Tomur, E.; Çomak, P.; and Karaçay, L. 2020. Vulnerability prediction from source code using machine learning. *IEEE Access*, 8: 150672–150684.

Brown, T. B. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Chen, Y.; Zhong, R.; Zha, S.; Karypis, G.; and He, H. 2021. Meta-learning via language model in-context tuning. *arXiv preprint arXiv:2110.07814*.

Coda-Forno, J.; Binz, M.; Akata, Z.; Botvinick, M.; Wang, J.; and Schulz, E. 2023. Meta-in-context learning in large language models. *Advances in Neural Information Processing Systems*, 36: 65189–65201.

Dai, H.; Li, H.; Chen, C.-S.; Shang, W.; and Chen, T.-H. 2020. Logram: Efficient log parsing using n n-gram dictionaries. *IEEE Transactions on Software Engineering*, 48(3): 879–892.

Dong, Q.; Li, L.; Dai, D.; Zheng, C.; Ma, J.; Li, R.; Xia, H.; Xu, J.; Wu, Z.; Liu, T.; et al. 2022. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*.

Du, M.; and Li, F. 2016. Spell: Streaming parsing of system event logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, 859–864. IEEE.

Du, M.; Li, F.; Zheng, G.; and Srikumar, V. 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.

Ester, M.; Kriegel, H.-P.; Sander, J.; Xu, X.; et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, 226–231.

Han, Z.; Li, X.; Xing, Z.; Liu, H.; and Feng, Z. 2017. Learning to predict severity of software vulnerability using only vulnerability description. In *2017 IEEE International conference on software maintenance and evolution (ICSME)*, 125–136. IEEE.

He, P.; Zhu, J.; Zheng, Z.; and Lyu, M. R. 2017. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE international conference on web services (ICWS)*, 33–40. IEEE.

He, S.; He, P.; Chen, Z.; Yang, T.; Su, Y.; and Lyu, M. R. 2021. A survey on automated log analysis for reliability engineering. *ACM computing surveys (CSUR)*, 54(6): 1–37.

He, S.; Lin, Q.; Lou, J.-G.; Zhang, H.; Lyu, M. R.; and Zhang, D. 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.

He, S.; Zhu, J.; He, P.; and Lyu, M. R. 2016. Experience report: System log analysis for anomaly detection. In *2016 IEEE 27th international symposium on software reliability engineering (ISSRE)*, 207–218. IEEE.

Huang, J.; Jiang, Z.; Chen, Z.; and Lyu, M. R. 2024a. LUNAR: Unsupervised LLM-based log parsing. *arXiv preprint arXiv:2406.07174*.

Huang, J.; Jiang, Z.; Liu, J.; Huo, Y.; Gu, J.; Chen, Z.; Feng, C.; Dong, H.; Yang, Z.; and Lyu, M. R. 2024b. Demystifying and Extracting Fault-indicating Information from Logs for Failure Diagnosis. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*, 511–522. IEEE.

Huo, Y.; Su, Y.; Lee, C.; and Lyu, M. R. 2023. Semparser: A semantic parser for log analytics. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 881–893. IEEE.

Jiang, Z.; Liu, J.; Chen, Z.; Li, Y.; Huang, J.; Huo, Y.; He, P.; Gu, J.; and Lyu, M. R. 2024a. LILAC: Log parsing using LLMs with adaptive parsing cache. *Proceedings of the ACM on Software Engineering*, 1(FSE): 137–160.

Jiang, Z.; Liu, J.; Huang, J.; Li, Y.; Huo, Y.; Gu, J.; Chen, Z.; Zhu, J.; and Lyu, M. R. 2024b. 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*, 223–234.

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

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

Li, G.; Wang, P.; and Ke, W. 2023. Revisiting large language models as zero-shot relation extractors. *arXiv preprint arXiv:2310.05028*.

Li, G.; Wang, P.; Liu, J.; Guo, Y.; Ji, K.; Shang, Z.; and Xu, Z. 2024a. Meta In-Context Learning Makes Large Language Models Better Zero and Few-Shot Relation Extractors. *arXiv preprint arXiv:2404.17807*.

Li, Z.; Fu, Q.; Huang, Z.; Yu, J.; Li, Y.; Lai, Y.; and Ma, Y. 2024b. Revisiting Log Parsing: The Present, the Future, and the Uncertainties. *IEEE Transactions on Reliability*.

Lin, Q.; Zhang, H.; Lou, J.-G.; Zhang, Y.; and Chen, X. 2016. Log clustering based problem identification for online service systems. In *Proceedings of the 38th International Conference on Software Engineering Companion*, 102–111.

Liu, Y. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

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

Ma, R.; Zhou, X.; Gui, T.; Tan, Y.; Li, L.; Zhang, Q.; and Huang, X. 2021. Template-free prompt tuning for few-shot NER. *arXiv preprint arXiv:2109.13532*.

Ma, Z.; Kim, D. J.; and Chen, T.-H. 2024. LibreLog: Accurate and Efficient Unsupervised Log Parsing Using Open-Source Large Language Models. *arXiv preprint arXiv:2408.01585*.

Min, S.; Lewis, M.; Zettlemoyer, L.; and Hajishirzi, H. 2021. Metaicl: Learning to learn in context. *arXiv preprint arXiv:2110.15943*.

Min, S.; Lyu, X.; Holtzman, A.; Artetxe, M.; Lewis, M.; Hajishirzi, H.; and Zettlemoyer, L. 2022. Rethinking the role of demonstrations: What makes in-context learning work? *arXiv preprint arXiv:2202.12837*.

Rey, D.; and Neuhäuser, M. 2011. Wilcoxon-signed-rank test. In *International encyclopedia of statistical science*, 1658–1659. Springer.

Robertson, S. 2004. Understanding inverse document frequency: on theoretical arguments for IDF. *Journal of documentation*, 60(5): 503–520.

Robertson, S.; Zaragoza, H.; et al. 2009. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4): 333–389.

Rubin, O.; Herzig, J.; and Berant, J. 2021. Learning to retrieve prompts for in-context learning. *arXiv preprint arXiv:2112.08633*.

Schubert, E.; Sander, J.; Ester, M.; Kriegel, H. P.; and Xu, X. 2017. DBSCAN revisited, revisited: why and how you should (still) use DBSCAN. *ACM Transactions on Database Systems (TODS)*, 42(3): 1–21.

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

Wu, Y.; Yu, S.; and Li, Y. 2024. Log Parsing using LLMs with Self-Generated In-Context Learning and Self-Correction. *arXiv preprint arXiv:2406.03376*.

Xiao, Y.; Le, V.-H.; and Zhang, H. 2024. Free: Towards More Practical Log Parsing with Large Language Models. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 153–165.

Xu, J.; Cui, Z.; Zhao, Y.; Zhang, X.; He, S.; He, P.; Li, L.; Kang, Y.; Lin, Q.; Dang, Y.; et al. 2024a. Unilog: Automatic logging via llm and in-context learning. In *Proceedings of the 46th ieee/acm international conference on software engineering*, 1–12.

Xu, J.; Yang, R.; Huo, Y.; Zhang, C.; and He, P. 2024b. DivLog: Log Parsing with Prompt Enhanced In-Context Learning. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 1–12.

Yang, A.; Yang, B.; Hui, B.; Zheng, B.; Yu, B.; Zhou, C.; Li, C.; Li, C.; Liu, D.; Huang, F.; Dong, G.; Wei, H.; Lin, H.; Tang, J.; Wang, J.; Yang, J.; Tu, J.; Zhang, J.; Ma, J.; Xu, J.; Zhou, J.; Bai, J.; He, J.; Lin, J.; Dang, K.; Lu, K.; Chen, K.; Yang, K.; Li, M.; Xue, M.; Ni, N.; Zhang, P.; Wang, P.; Peng, R.; Men, R.; Gao, R.; Lin, R.; Wang, S.; Bai, S.; Tan, S.; Zhu, T.; Li, T.; Liu, T.; Ge, W.; Deng, X.; Zhou, X.; Ren, X.; Zhang, X.; Wei, X.; Ren, X.; Fan, Y.; Yao, Y.; Zhang, Y.; Wan, Y.; Chu, Y.; Liu, Y.; Cui, Z.; Zhang, Z.; and Fan, Z. 2024. Qwen2 Technical Report. *arXiv preprint arXiv:2407.10671*.

Yao, Y.; Duan, J.; Xu, K.; Cai, Y.; Sun, Z.; and Zhang, Y. 2024. A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing*, 100211.

Yu, S.; He, P.; Chen, N.; and Wu, Y. 2023. Brain: Log parsing with bidirectional parallel tree. *IEEE Transactions on Services Computing*, 16(5): 3224–3237.

Zhang, X.; Xu, Y.; Lin, Q.; Qiao, B.; Zhang, H.; Dang, Y.; Xie, C.; Yang, X.; Cheng, Q.; Li, Z.; 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.

Zhang, Y.; Zhang, F.; Yang, Z.; and Wang, Z. 2023. What and how does in-context learning learn? bayesian model averaging, parameterization, and generalization. *arXiv preprint arXiv:2305.19420*.

Zhao, W. X.; Zhou, K.; Li, J.; Tang, T.; Wang, X.; Hou, Y.; Min, Y.; Zhang, B.; Zhang, J.; Dong, Z.; et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 1(2).

Zhong, A.; Mo, D.; Liu, G.; Liu, J.; Lu, Q.; Zhou, Q.; Wu, J.; Li, Q.; and Wen, Q. 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*, 4559–4570.

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