LLMLog: Advanced Log Template Generation via LLM-driven Multi-Round Annotation

Fei TENG
CSE, HKUST
fteng@connect.ust.hk

Haoyang LI*
Computing, PolyU
haoyang-comp.li@polyu.edu.hk

Lei CHEN
DSA, HKUST & HKUST (GZ)
leichen@cse.ust.hk

ABSTRACT

Modern computing systems, such as HDFS and Spark, produce vast quantities of logs that developers use for tasks like anomaly detection and error analysis. To simplify log analysis, template generation methods have been proposed to standardize log formats, transforming unstructured data into structured templates. Existing heuristic-based methods and neural network-based methods suffer from low accuracy problems due to the reliance on handcrafted heuristics or specific log patterns in training sets. Recently, large language models (LLMs) have shown great potential in log template generation. However, they often struggle with ambiguous, complex, or highly specific log content, which can lead to errors in generating accurate templates. To address these challenges, we propose LLMLog, a multi-round annotation framework with adaptive in-context learning. We first propose an edit-distance-based similarity metric to evaluate log similarity. Then, we introduce a method to select the most informative k unlabeled logs for annotation by considering both the representativeness of the logs and the confidence of LLM predictions. Additionally, we design an adaptive context selection strategy that adaptively selects labeled logs to ensure comprehensive keyword coverage for unlabeled logs. These labeled logs serve as the context for LLMs to better understand the unlabeled logs, thereby enhancing the accuracy of template generation. Extensive experiments on sixteen datasets demonstrate that LLMLog outperforms the state-of-the-art approaches.

PVLDB Reference Format:
Fei TENG, Haoyang LI and Lei CHEN. LLMLog: Advanced Log Template Generation via LLM-driven Multi-Round Annotation. PVLDB, 18(9): 3134 - 3148, 2025.
doi:10.14778/3746405.3746433

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/XinTT/LLMLog.

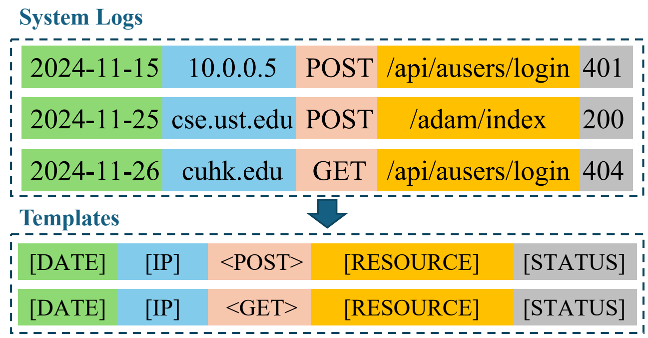


Figure 1: Template generation from system logs. As shown in the figure,we can generate two templates for the three logs from HTTP. Templates help structure the logs by assigning types or categories to each word in the logs.

1 INTRODUCTION

Modern computing systems, such as HDFS and Spark, generate vast quantities of logs, which offer a wealth of information about system runtime behavior. Developers can use the recordings to debug and maintain the computer system, including anomaly detection [14, 25, 41, 43, 63, 82] and error analysis [1, 24, 38]. However, the massive volume of logs causes the difficulty of system developers and maintain staffs to detect the anomalies and track errors. To facilitate easier understanding and analysis, motivated by pattern mining approaches [3, 44, 75], researchers have proposed generating templates for these massive logs. For example, as shown in Figure 1, template generation creates structured formats to consistently standardize the logs, making them easier to parse, analyze, and maintain. This process extracts structured patterns from unstructured log data, which transforms raw logs into more organized formats. As a result, developers can quickly identify fields with anomalies or errors, making it easier to debug issues and maintain system health. The diversity of logs within a system, such as configuration, user, and error logs, often maps to multiple templates rather than a single one, complicating their effective template generation. This challenge is further exacerbated by the ever-growing volume of logs and their variability over time due to system updates.

Depending on the technique of generating templates for logs, current approaches can be categorized into three types, i.e., heuristic-based methods [22, 23, 36, 47, 62], neural network-based methods [33, 49], and LLM-based methods [74]. Firstly, heuristic-based methods [22, 23, 36, 47, 62] separate user logs into different clusters using handcrafted rules or heuristics, such as assuming all logs are of equal length if they have the same template. However, such predefined heuristics or rules cannot completely match patterns of logs. Consequently, they lack the flexibility and adaptability to standardize diverse logs. Secondly, neural-network-based approaches train neural networks to predict the type of each word in the logs, such as `[DATE]` and `[IP]` in Figure 1. However, the success of supervised

Proceedings of the VLDB Endowment, Vol. 18, No. 9 ISSN 2150-8097.
doi:10.14778/3746405.3746433

*Corresponding Author.

approaches relies on numerous annotated labels to train the neural networks, which requires expensive human effort [33, 74].

Recently, large language models (LLMs) [18–20, 54, 56, 88] pre-trained on diverse corpora, such as text and code, have shown exceptional abilities in understanding text, achieving significant success in natural language processing tasks [20, 53, 59], such as question answering [50, 52, 71]. Therefore, researchers have explored using LLMs to comprehend text in logs and generate templates for them. Typically, they provide each user log along with candidate word labels to the LLMs, instructing them to label each word in the log to create its template. However, due to the presence of complex, specific, and ambiguous words in logs (e.g., a username `asjaks001`), LLMs may struggle to interpret these words accurately, leading to challenges in generating correct templates.

To enhance the understanding of words in logs, researchers [11, 60, 61, 70] have proposed a multi-round annotation framework for large language models (LLMs), which typically consists of an annotation component and an in-context learning (ICL) component. Firstly, in the annotation component, each round involves labeling a subset of logs that lack corresponding templates. Researchers typically define a similarity score among logs, such as cosine similarity between log embeddings [6, 46, 67, 72]. Based on this score, they select a set of logs within a budget k_a to represent the other unlabeled logs. Second, the in-context learning component involves selecting top-k_c similar logs with their templates to serve as context for unlabeled logs [16, 26, 74]. This contextual information helps the LLMs understand the correlation between words and labels, enabling them to predict labels for the unlabeled logs. After each round, the framework filters logs with lower prediction confidence as unlabeled. It then iteratively repeats the annotation and ICL steps until the LLMs generate templates for all logs.

Nevertheless, despite the success of existing frameworks, there are still three limitations. Firstly, they define log similarity based on log embeddings in both annotation and in-context learning components. However, this embedding similarity overlooks crucial aspects such as important words (e.g., POST), and more emphasizes useless words (e.g., time stamps or IP addresses in figure 1). Thus, in annotation or in-context learning, the similar logs may share duplicate time stamps while lack crucial words, offering insufficient information. Second, in the annotation component, existing approaches typically select the top-k_a unlabeled logs for annotation where selected logs are similar to as many logs as possible. However, this approach neglects the importance of LLM prediction confidence, where logs with low prediction confidence often require more annotations. Third, in the in-context learning component, existing approaches [28, 37, 39, 45] select a fixed number of top-k_c similar labeled logs to serve as context for each unlabeled log. However, using a fixed number of labeled logs may fail to provide sufficient context for the LLM, potentially leading to the generation of incorrect templates.

To address the aforementioned limitations, we propose an LLM-driven multi-round annotation framework with adaptive in-context learning, called LLMLog. Specifically, this framework defines an edit-distance based log similarity metric which emphasize the important keywords by measuring word insertion/deletion/replacement operations. The most useful logs in annotation and ICL can be respectively identified by adaptively varying the wordset. Benefit from similarity metric, we define activated logs for representativeness of annotated logs. In each round of the annotation component, we first adaptively determine the budget for that round. Then, we propose a greedy algorithm to annotate logs by jointly optimizing the confidence of LLM predictions and the total number of activated logs, taking into account logs with low confidence and high representativeness. In adaptive ICL component, we select the minimum number of demonstrative logs from annotation set for each input log by ensuring all keywords are covered from input log in a greedy manner. Compared to the fixed top-k_c strategy, our adaptive selection can precisely guide LLM by ensuring the informativeness of contextual information from each demonstrative log. The contributions of this paper are summarized as follows.

- We propose LLMLog, a novel LLM-driven multi-round annotation framework with adaptive in-context learning for log template generation. The framework addresses limitations in existing methods by iteratively improving annotation and template generation processes.
- We propose an semantic edit-distance-based metric for keyword coverage and an LLM feedback metric considering word consistency and confidence. Additionally, we introduce an adaptive strategy to dynamically allocate the annotation budget. Finally, we formulate the NP-hard multi-round log annotation problem and develop a greedy algorithm with theoretical guarantees.
- We develop an adaptive strategy for selecting the minimum number of demonstrative logs as context for each unlabeled log. This approach ensures that all keywords from the input log are covered, enhancing the LLM's understanding and improving template generation accuracy compared to fixed top-k strategies.
- Extensive experiments on 16 datasets demonstrate that LLMLog achieves higher accuracy than state-of-the-art baselines while reducing computational and API costs through adaptive demonstration selection and efficient log annotation.

2 PRELIMINARY AND RELATED WORKS

In this section, we first present the preliminaries on logs and log template generation. Then, we discuss the related works. Important notations of this paper are summarized in Table 1.

2.1 Log Template Generation Problem

Logs are textual sequences that document the behaviors and events of a system. Formally, a log s is one system message recording a system event, composed of a set of words, denoted as $s = (w_1^s, w_2^s, ..., w_{|s|}^s)$, i.e., `2024-11-14 192.168.1.1 GET /index.html 200 123ms` is a log. Logs serve as an essential tool for system monitoring, debugging, and performance analysis [63]. By capturing a detailed account of system activities, logs allow developers to trace events, identify issues, and ensure optimal functionality [41, 77]. Recent research in database and data management focuses on various aspects of log analysis, including log anomaly detection [43, 48, 63, 80], log-based retrieval [12, 27, 29, 78], and root cause analysis [7, 42, 68, 76].

In modern systems, logs are produced in massive volumes daily. Each system generates multiple system events, producing logs in various formats corresponding to different templates. However, the large size of logs makes it challenging and time-consuming for developers and operators to trace events and identify issues efficiently.

Recent studies have explored knowledge base template generation for user queries [3, 57, 75, 84], while several researchers have proposed methods for generating templates for SQL queries [15, 17, 44, 86]. There are also several works proposed to generate templates for Web pages [4, 9, 30, 73]. Motivated by template generation works in database area [44, 75], we can generate templates for logs, converting unstructured logs into structured templates, enabling efficient storage, analysis, and debugging [43, 63]. Formally, given a log $s = (w_1^s, \ldots, w_{|s|}^s)$ and word type candidates $\mathcal{T} = \{\tau_1, \cdots, \tau_{|\mathcal{T}|}\}$ and system keyword candidates $\mathcal{K} = \{k_1, \cdots, k_{|\mathcal{K}|}\}$, the target of log template generation is to generate a template $t = (w_1^t, \cdots, w_{|t|}^t)$ for s, which maps each word $w_i^s \in s$ to a corresponding word type $w_i^t \in \mathcal{T}$ or a system keyword $w_i^t \in \mathcal{K}$.

For example, the template of the log `2024-11-14 192.168.1.1 GET /index.html 200 123ms` is `[DATE] [IP] <GET> [RESOURCE] [STATUS] [LATENCY]`, where $[\cdot]$ is a word type and $< \cdot >$ is a keyword. The template of log `2024-11-14 192.168.1.2 DELETE /test.html 200 123ms` is `[DATE] [IP] <DELETE> [RESOURCE] [STATUS] [LATENCY]`. By identifying structured templates from raw logs, log template generation streamlines the management and analysis of large-scale log data from systems such as Apache [2, 87], HDFS [2, 87], and Spark [66, 87]. These templates make it easier to track IP addresses, enabling the detection of anomalies or errors in database systems [43, 63].

Table 1: Important Notations

Notation	Definition
s, t	A log $s \in S$ and its template $t \in T$
$\hat{t}$	Predicted template $\hat{t}$ of log s, $t \in \hat{T}$
w_i^s	The i-th word in the log s
w_i^t	The i-th type in template t
$\mathcal{K}$	Keyword set
$\mathcal{T}$	Candidate word type set
$G = (S, W)$	Bipartite graph between log set S and words W
r	The r-th round
L_r	The annotated logs at the r-th round
L	Annotated logs
U	Unlabeled logs
$cosine(\cdot)$	Cosine similarity score
$SED(\cdot)$	Semantic-based edit distance between two logs
I_{s_i}	Representative score of s_i
$P(s_i, \hat{t}_i)$	Average probability of predicted template $\hat{t}_i$
$\mathbb{I}(s_i, \hat{s}_i)$	Prediction consistency indicator
$C(s_i, \hat{t}_i, \hat{s}_i)$	Confidence score
B_r	Annotation budget at the r -th round
W_r	Identified words at the r -th round
$IS(\cdot)$	The informative score in Equation (8)
λ	Trade-off parameter in Equation (8)
D_s	Demonstrative logs of s
$UW(D_s)$	Word set in D_s

2.2 Log Template Generation Approaches

Depending on the techniques used for generating log templates, existing works can be classified into three categories: heuristic-based approaches [22, 23, 36, 47, 62], neural network-based approaches [33, 49], and LLM-based approaches [74].

2.2.1 Heuristic-based Approaches. Heuristic-based models [22, 23, 36, 47, 62] depend on handcrafted rules or heuristics to group logs into multiple clusters. In practice, heuristics cannot match the characteristics of logs from multiple domains well, resulting in low flexibility and adaptability in labelling diverse logs. For instance, Drain [23] assumes that the first word of a log must always be a keyword. However, this assumption is not consistent across logs from different systems. Moreover, logs from specific datasets may follow unique patterns. For example, in the HDFS dataset [87], the pattern `blk_3651` represents a block with ID `3651`, which can be interpreted using a specific regular expression like *blk* \ *d+*. However, such patterns do not exist in other datasets, such as Mac [87] and Android [87], requiring the re-writing of regular expressions for each dataset, which incurs significant human effort. As each system have logs in diverse templates, the developers have to check the whole log sets from each system to separately define rules or threshold for template generation, which is impractical in real-world log datasets [87] with hundreds of templates.

2.2.2 Neural Network-based Approaches. Neural network-based models [33, 49] train neural network models (i.e. transformers) to predict each word type in logs. However, the training process demands large-scaled logs with annotated word type. The annotation of word type in large-scaled logs requires experienced software maintainer, which is expensive. Besides, the trained template generation model relies on patterns existing in training set with potential low generalization ability to unseen patterns. For examples, if logs in training set only cover the template `[ADDRESS] Byte flume reports host available`, the trained model may falsely infer the template of log `[ADDRESS] Byte flume reports host available again` while an important keyword `again` is missed due to it is unseen in training set. One intuitive solution to generalization problem is to re-train or update the neural network model based on the logs [55, 81]. However, it is also costly to re-train the transformer model [33, 49, 74]. As high cost for human-resources and low generalization problem for both heuristic-based approaches and neural network-based approaches, LLM-based template generation methods [74] have been proposed.

2.2.3 LLM-based Approaches. Large language model pre-trained on massive corpora has obtained extraordinary natural language processing ability [35]. Motivated by success of LLM, researchers have investigated LLM-based log template generation that feed the log and candidate words labels to LLM with instructing LLM to label each word. However, as logs contain various domain-specific words which is excluded in open-sourced corpora, it is difficult for LLM to understand these words [74]. Based on recent studies, performance of pre-trained LLM can be training-freely augmented by prompting LLM with several examples with ground truth templates as contexts, calling ICL [8, 16, 28, 37, 39, 45, 74].

Researchers have proposed a multi-round annotation framework for LLMs [45, 58, 79, 85] with an annotation component and an ICL component. As for annotation, they firstly compute the cosine similarity score between embedding of logs. Based on the score, annotation process finds a representative subset as labeled logs within a budget k_a. In ICL component, for each unlabeled log, they selects top-k_c similar labeled logs which offer contextual knowledge for LLM to comprehend the correlation between each word and labels. AdaICL [45] uses each unlabeled log to represent its top-k_a similar logs and proposes selecting k_a unlabeled logs to represent as many logs as possible. IDEAL [79] employs an information diffusion process [79] to select the top-k_a most informative unlabeled logs. DivLog [74] selects the top-k_a most diverse and informative unlabeled logs. However, the current works achieve sub-optimal performance due to following three issues: 1) The embedding of logs overlooks the importance of keywords but emphasizes several useless words thus cannot identify the most informative contexts. 2) The annotation ignores that logs are unconfident for LLM also worthy for annotation. 3) Fixed top-k_c similar demonstrations misguide LLM by irrelevant/scarced contexts.

3 METHOD

In this section, we introduce our LLM-driven multi-round annotation and adaptive in-context learning framework in Figure 2.

3.1 Framework Overview

Step 1: Log Similarity and Annotation. This component aims to construct and update the annotated log set L at each round r by selecting the most representative and challenging unlabeled logs from the dataset U. At the r-th round, we compute an edit-distance-based metric SED between logs, which emphasizes important keywords to identify representative logs. Based on SED, the framework selects a subset L_r of size B_r for human annotation by jointly considering two factors: representativeness and LLM prediction confidence. More details can refer to Section 3.2 and Section 3.3.
Step 2. Adaptive Demonstration Selection. This component dynamically selects a minimum number of labeled logs from the labeled log set L to serve as context for each unlabeled log $s_i \in U$ during LLM inference. This dynamic selection ensures that all words in the input log s_i are covered while avoiding irrelevant or redundant context, which could otherwise degrade the quality of template generation. More details can refer to Section 3.4.
Step 3. LLM-Driven Template Generation. After constructing the adaptive context for each unlabeled log s_i, we generate the final template prediction. This process begins with prompt construction, which includes three key elements: an instruction for template generation, examples of labeled logs with their templates (retrieved from the context), and the input log s_i with its identified words. Once the prompt is constructed, it is fed into the LLM for inference, resulting in the predicted template $\hat{t}_i$ for the input log s_i.

3.2 Log Similarity

As mentioned in Section 2.2.3, existing works compute the similarity [69] between two logs s_i and s_j using the cosine similarity of their embeddings [46, 67, 72], which can be defined as

$$sim(s_i, s_j) = cosine(\mathbf{s}_i, \mathbf{s}_j), \tag{1}$$

where $\mathbf{s}_i = \frac{\sum_{k=1}^{|s_i|} \mathbf{w}_k^s}{|s_i|}$ is the embedding of log s_i, and $\mathbf{w}_k^s$ represents the embedding of the k-th word in log s_i. However, the cosine similarity for log embedding is not suitable for representing log similarity in template generation, as it overlooks the importance of several keywords and favors irrelevant words with a large number of letters, such as timestamps or IP addresses. We aim to find similar template logs as contexts for the target log in the ICL phase, where each contextual log should provide enough information to guide the LLM on how to process each word.

Suppose we have an unlabeled log `com.cse.ust.hk:8080 POST`, and two labeled logs, `proxy.cse.cuhk.edu.hk:5070 POST` and `com.cse.ust.hk:8080 GET`. Intuitively, we should pick the first one to demonstrate how LLM converts the IP address to `[IP]` and tags `POST` with `<POST>`. However, the cosine similarity in Equation (1) emphasizes the longer words, such as `com.cse.ust.hk:8080`. Therefore, the cosine similarity metric selects `com.cse.ust.hk:8080 GET` as the context, which has no information about the keyword `POST`. Such a context may falsely guide the LLM to generate templates with an irrelevant word, `GET`. To emphasize the important words, we can build a bipartite graph to model the similarity via connections between logs and the words they contain.

Definition 1 (Log-Word Bipartite Graph). *We build a bipartite graph $G = (S, W)$ where $S = \{s_i\}_{i=1}^{|S|}$ is the set of logs and $W = \cup_{s_i \in S} \cup_{w \in s_i} w$ is the set of words contained in S. There is an edge connecting log s_i and word w if $w \in s_i$.*

In the log-word bipartite graph, multiple logs can be linked through their shared words. This implies that a log s_i can assist an LLM in understanding a similar log s_j if they share similar or identical words. Consequently, the LLM can generate the correct template for the words in log s_j. Here, we propose a novel semantic-based edit-distance in real sequence (SED) to determine the similarity among two logs, instead of using sentence embedding-based cosine similarity in Equation (1).

The core idea of our proposed metric $SED(s_i, s_j)$, is to compute the minimal number of operations (including deletion, insertion, and replacement) required to transform log s_i into log s_j, while incorporating word semantics into the calculation. Formally, given two logs $s_i = (w_1^{s_i}, \cdots, w_{|s_i|}^{s_i})$ and $s_j = (w_1^{s_j}, \cdots, w_{|s_j|}^{s_j})$ and all the labeled logs $L = \{(s_k, t_k)\}_{k=1}^{|L|}$, the semantic-based EDR similarity score $SED(s_i, s_j)$ between s_i and s_j is defined as follows:

$$SED(s_i, s_j) = \begin{cases} |s_i^r|, if |s_j^r| = 0 \\ |s_j^r|, if |s_i^r| = 0 \\ min \begin{cases} SED(R(s_i^r), R(s_j^r)) + c(s_i^r[0], s_j^r[0]) \\ SED(R(s_i^r), s_j^r) + 1 \\ SED(s_i^r, R(s_j^r)) + 1 \end{cases} \end{cases} \tag{2}$$

$$c(w_1, w_2) = \begin{cases} 1, & cosine(\mathbf{w}_1, \mathbf{w}_2) \geq 0 \\ 0, & cosine(\mathbf{w}_1, \mathbf{w}_2) < 0 \end{cases} \tag{3}$$

where $s_i^r = \{w_i \mid w_i \in s_i \wedge w_i \notin s_k, \forall s_k \in L\}$ represents the remaining words in s_i that are not identified in the labeled logs. Also, $R(s_i^r)$ represents the sub-sequence of words in s_i^r with the current first word removed, Equation (2) recursively enumerates all words in s_i and s_j. More specifically, as SED is designed to measure the minimal distance between two logs, we use $min(\cdot)$

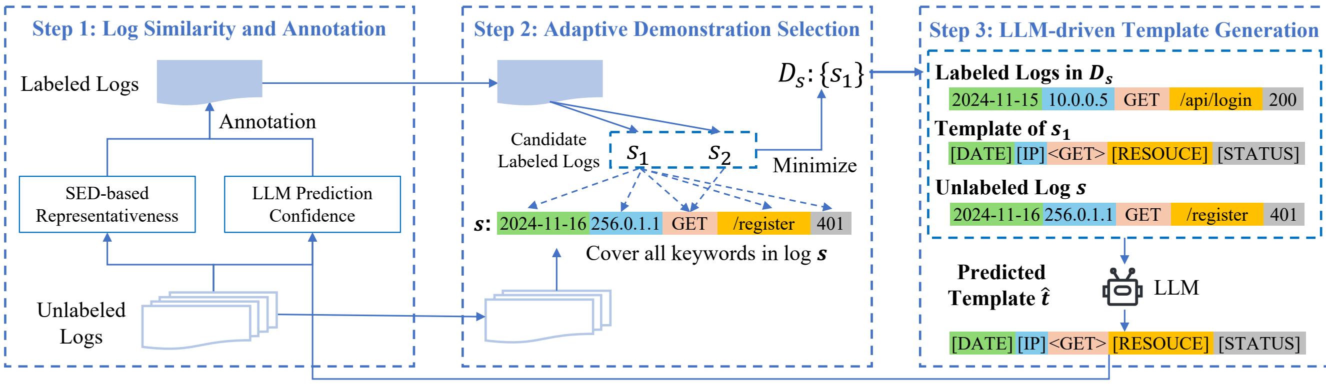


Figure 2: Framework overview of LLMLog consists of three key components: (1) Log similarity and annotation: LLMLog employs a semantic-based edit-distance (SED) metric to assess log similarity. It selects a subset of logs for human annotation by identifying the most representative and challenging ones through a greedy algorithm across multiple rounds. (2) Adaptive demonstration selection: LLMLog adaptively selects a minimal set of labeled logs that comprehensively cover all relevant words for each input log. This ensures that the contextual information provided to the LLM is both relevant and concise. (3) LLM-driven template generation: LLMLog constructs tailored prompts by incorporating the adaptive contexts and unlabeled logs. These prompts are then input into the LLM, which generates accurate templates for the unlabeled logs.

instead of $avg(\cdot)$ or $max(\cdot)$ to compute the minimal operation of replace/insertion/deletion. Instead of only computing the minimal operation for the first word in s_i and s_j, the minimum distance for the two sequences is obtained by recursively iterating $R(s_i^r)$ and $R(s_j^r)$ in a dynamic programming manner. The function $c(w_1, w_2)$ computes the cosine similarity between the two words based on cosine similarity and $\mathbf{w}_1$ is the embedding of word w_1 [5]. We define $c(w_1, w_2) = 1$ if the word similarity between w_1 and is greater than 0 otherwise the value $c(w_1, w_2) = 0$.

Compared to cosine similarity in Equation (1), SED is better at identifying logs with the same templates by emphasizing keywords. By definition in Section 2.1, logs under the same template share keywords. Thus, we can use edit distance to capture the common parts. The effect of other words can be further reduced by using an adaptive word set. For example, consider the previous case where cosine similarity produces a false positive, identifying `com.cse.ust.hk:8080 GET` as a match for the unlabeled log `com.cse.ust.hk:8080 POST`, while ignoring the true positive log `proxy.cse.cuhk.edu.hk:5070 POST`. As the adaptive word set in the SED metric removes irrelevant IP addresses as shown in labeled logs, the edit distance between input log and `proxy.cse.cuhk.edu.hk:5070 POST` becomes 0, which is smaller than the distance between it and `com.cse.ust.hk:8080 GET`. Therefore, SED is more suitable for measuring the similarity between logs in the similar templates. In particular, SED can be computed in $O(|s_i| * |s_j|)$, where $|s_i|$ denotes the number of words in s_i.

3.3 Multiple Round Log Annotation

In this subsection, we introduce our LLM-driven multi-round annotation approach. Specifically, our framework for multi-round labeled log annotation follows a common setup where the total number of rounds is n, and each round r is allocated a budget of B_r. In the r-th round, given the unlabeled logs $U = \{s_i\}_{i=1}^{|U|}$, the objective is to annotate up to B_r logs, which are then added to the labeled log set $L = \{(s_j, t_j)\}_{j=1}^{|L|}$. This labeled log set L is subsequently utilized as demonstration candidates for unlabeled logs $U \setminus L$ during LLM inference, improving its ability to generate accurate templates.

To achieve effective annotation, we propose two complementary metrics to guide the selection of the most representative and challenging unlabeled logs for annotation in each round. The first metric is the representative score, which evaluates how representative an unlabeled log is in relation to other unlabeled logs. The second metric is LLM confidence, which measures the LLM's confidence in generating a correct template for each unlabeled log. By combining these two metrics, our approach ensures that the most representative and challenging logs are strategically selected for annotation in each round. The details of these two metrics are discussed in the following sections.

3.3.1 Representative Score. We firstly present the metric of the reprensentativeness of labeled logs. As introduced in Section 3.2, similar words tend to have the same types, and logs with low SED scores are likely to share similar templates. Therefore, we aim to annotate logs that share similar words with many other logs, maximizing the representativeness of the labeled set. This approach ensures that the labeled logs capture a diverse and meaningful range of patterns present in the data. As a result, we can select representative logs for labeling because they provide more contextual information for other logs during in-context learning (ICL). Formally, given two logs s_i and s_j, we say that s_j is represented by s_i if $SED(s_i, s_j) \leq \delta * min(len(s_i), len(s_j))$. We define the representative set for a log s_i as:

$$I_{s_i} = \{s_j | SED(s_i, s_j) \leq \delta * min(len(s_i), len(s_j))\}, \delta \in (0, 1] \quad (4)$$

The threshold is set to $\delta * min(len(s_i), len(s_j))$, implying that the represented log s_j must have at least one common word to s_i. In general, the size of the set I_{s_i} reflects how representative s_i is among all unlabeled logs. δ is hyper-parameter to control the size of representative group. Intuitively, δ should not be too large as it will weaken the representativeness of each log, thus provide less informative contexts for unlabeled logs. We give a parameter sensitivity experiment in Section 4.4 to investigate the effects of δ.

3.3.2 LLM Prediction Confidence Score. We introduce the confidence score of each unlabeled log based on the prediction of LLMs. Intuitively, if the LLM exhibits low confidence in generating an accurate template for a log, that log is prioritized for annotation. By focusing on these low-confidence logs, the annotation process targets the most challenging cases, thereby improving the LLM's overall performance on similar logs, which is a common practice in LLM annotation works [45, 58, 79].

In general, given an unlabeled log $s_i = (w_1^{s_i}, \dots, w_{|s_i|}^{s_i})$, the generated log template by a LLM can be denoted as $\hat{t}_i = \{\hat{w}_1^{t_i}, \cdots, \hat{w}_{|t_i|}^{t_i}\}$. Firstly, we can use the average predicted word-probability to estimate the confidence by LLM, defined as follows.

$$P(s_i, \hat{t}_i) = \frac{1}{|\hat{t}_i|} * \sum_{k=1}^{|\hat{t}_i|} p(w_k^{\hat{t}_i}) \tag{5}$$

where $p(w_k^{\hat{t}_i})$ is the predicted probability of LLM for each word in the predicted template $\hat{t}_i$, i.e, $w_k^{\hat{t}_i} \in \hat{t}_i$. If the average probability is low, it indicates the LLM's low confidence in its predictions, suggesting that the log is challenging to interpret and needs to be prioritized for annotation. Suppose there is a log `com.cse.ust.hk:8080 DELETE` where the keyword `DELETE` does not exist in labeled set. LLM is less confident for `DELETE` than that other identified keywords, like `POST`. Therefore, the predicted probability of the log tends to be less than other logs. The average probability metric selects this log for annotation.

However, a high predicted word-probability $P(s_i, \hat{t}_i)$ cannot guarantee that the generated word type $w_j^{\hat{t}_i} \in \hat{t}_i$ really corresponds to $w_j^{s_i} \in s_i$. The reason is that LLMs generate output words in a regressive manner, which cannot guarantee that the generated template word $w_j^{\hat{t}_i} \in \hat{t}_i$ corresponds accurately to $w_j^{s_i} \in s_i$. Additionally, current researchers [34, 40, 83] have demonstrated that LLMs suffer from the hallucination problem, where the model may misunderstand the context and generate irrelevant or incorrect information. For example, the correct template of a log `com.cse.ust.hk:8080 POST` is `[IP] <POST>`. However, the LLM might predict the template of this log as `[IP] [STATUS]` with high confidence, as it may mistakenly interpret `[IP] <POST>` as `[IP] [STATUS]` due to the hallucination problem. Thus, the LLM produces an incorrect result even exhibiting high confidence in the predicted template.

Therefore, except for average predicted word-probability metric, inspired by the template generation requirements in Section 2.1, we can derive a word-consistency metric for template generation task. Specifically, in addition to the predicted template $\hat{t}_i$ of s_i generated by the LLM, we enable the LLM to generate the corresponding words $\hat{s}_i = (w_1^{\hat{s}_i}, \cdots, w_{|\hat{s}_i|}^{\hat{s}_i})$ as well. We then compare the consistency between the words in the input log $s_i = (w_1^{s_i}, \cdots, w_{|s_i|}^{s_i})$ and the generated words $\hat{s}_i = (w_1^{\hat{s}_i}, \cdots, w_{|\hat{s}_i|}^{\hat{s}_i})$, with consistency indicator $\mathbb{I}(s_i, \hat{s}_i)$ defined as follows.

$$\mathbb{I}(s_i, \hat{s}_i) = \begin{cases} 0, s_i = \hat{s}_i \\ 1, s_i \neq \hat{s}_i \end{cases} \tag{6}$$

The word-consistency indicator function evaluates whether the words in the log $\hat{s}_i$ generated by the LLM are consistent with the original words in the input log s_i. Specifically, it ensures that no new words or non-candidate labels are falsely generated by the LLM. Additionally, it verifies whether the word count $|\hat{s}_i|$ matches the word count $|s_i|$, ensuring that each word is correctly labeled. Any templates that fail to meet these two conditions are considered incorrect and can be used to identify error cases. To assess the LLM's performance in the log template generation task, we combine the average word-probability and word-consistency metrics. Formally, given a log s_i and the predicted log template $\hat{t}_i$ associated with the predicted words $\hat{s}_i$, we define the confidence score as $C(s_i, \hat{t}_i, \hat{s}_i)$ as follows. into a weighted sum as follows.

$$C(s_i, \hat{t}_i, \hat{s}_i) = a * (1 - P(s_i, \hat{t}_i)) + (1 - a) * \mathbb{I}(s_i, \hat{s}_i) \tag{7}$$

where a is a weight to balance the average probability $P(s_i, \hat{t}_i)$ and consistency score $\mathbb{I}(s_i, \hat{s}_i)$, where we use $1 - P(s_i, \hat{t}_i)$ to select logs with low confidence in the predicted template $\hat{t}_i$. A larger $C(s_i, \hat{t}_i, \hat{s}_i)$ indicates that the LLM has low confidence and potential inconsistency between $\hat{s}_i$ and s_i, suggesting that s_i is challenging.

3.3.3 LLM-driven Log Annotation Problem. We introduce our LLM-driven log annotation problem by incorporating the proposed representative score in Section 3.3.1 and the LLM prediction confidence score in Section 3.3.2. The formal definition is given as follows.

Definition 1 (LLM-driven Log Annotation Problem). *Given a budget B_r and an unlabeled log set U at round r, where the LLM predicts a template $\hat{t}_i$ along with the corresponding log $\hat{s}_i$ for each unlabeled log $s_i \in U$, the objective is to select a subset of informative unlabeled logs $L_r \subseteq U$ for annotation, such that $|L_r| \leq B_r$. The selection aims to maximize the following objective:*

$$IS(L_r) = \max \sum_{s_i \in L_r} (1 - \lambda) \frac{|\bigcup_{i=0}^{|L_r|} I_{s_i}|}{|U|} + \lambda C(s_i, \hat{t}_i, \hat{s}_i) \tag{8}$$

$$s.t. \quad |L_r| \leq B_r, L_r \subseteq U \tag{9}$$

where λ is a trade-off parameter, I_{s_i} is the unlabeled logs represented by s_i defined in Equation (4)*, and $C(s_i, \hat{t}_i, \hat{s}_i)$ is the confidence score defined in Equation* (7)*.*

As shown in Equation (8), in each round r, the first term, $\frac{|\bigcup_{i=0}^{|L_r|} I_{s_i}|}{|U|}$, prioritizes selecting representative logs that are likely to impact a larger number of unlabeled logs and guide the LLM to process more logs effectively. The second term, $C(s_i, \hat{t}_i, \hat{s}_i)$, ensures the selection of logs with the lowest LLM prediction confidence. By combining these two scores, we can effectively select B_r informative logs, $L_r \subseteq U$, for annotation.

Theorem 1. *The LLM-driven log annotation problem is NP-hard.*

Proof Sketch. We prove Theorem 3 by reduction from the Max Coverage problem [32] to our problem. Due to space limits, we put full proof in technique report [65] Appendix 6.1. □

3.3.4 Algorithm for Annotation Log Selection. As demonstrated in Theorem 1, the LLM-driven log annotation problem under the budget B_r is *NP-hard*, indicating that it is unlikely to be solved optimally in polynomial time. To address this, we propose a greedy algorithm with a theoretical guarantee. The basic idea is to greedily select the unlabeled log that can bring the maximum information gain into the selected annotation set until exceeding the log budget B_r. Specifically, given the unlabeled set U, we first define the marginal information gain of s for the selected set L_r as follows:

$$\triangle IS(s|L_r) = IS(L_r \cup \{s\}) - IS(L_r) \tag{10}$$

The details are provided in Algorithm 1. Specifically, we first initialize the selected log set L_r as $\emptyset$ (line 1). Then, for each unlabeled log $s \in U$, we update $SED(s, s_i)$ among the other unlabeled logs $s_i \in U \setminus s$ (line 3). Also, we obtain the representative log set I_s for each unlabeled log s (line 4) and compute the confidence score $C(s, \hat{t}, \hat{s})$ based on the LLM model f_θ (line 5). Next, we compute the informative score $IS(L_r \cup s)$ by incorporating each unlabeled log $s \in U$ into the selected log set L_r. We then select the log s^* with the maximum $\triangle IS(s|L_r)$, as defined in Equation (10) (lines 7–10). Afterward, we add s^* to S_r and remove it from the set of unlabeled logs U (lines 11–12). This selection procedure is repeated until B_r logs have been selected (lines 6–13).

Time complexity Loop in line 2 enumerates U to compute the representative score and LLM confidence score. For each $s \in U$, let s_{max} be the log with maximum number of words, it takes at most $O(|s_{max}|^2)$ time to compute SED while scores in line 4 and line 5 can be computed in constant time, where the loop costs $O(|U||s_{max}|^2)$. As for while loop starting in line 6, it requires to enumerate each instance in U in inner for loop at line 7 to select one log with maximized $\triangle IS(s|L_r)$, which roughly takes time complexity $O(B_r * |U|)$. Thus, the overall time complexity is $O((B_r + |s_{max}|^2) * |U|)$.

Theorem 2. *Algorithm 1 has an approximation ratio of* $1 - \frac{1}{e}$.

Proof Sketch. Let $IS(L_r^*)$ denotes the optimal value of objective in Equation (8) within budget B_r. We first prove the $\triangle IS(s|D_s)$ in Equation (10) is monotone increasing and submodular. Then we prove $(1 - (\frac{1}{B_r})^{B_r}) IS(L_r^*) \leq IS(L_r)$. Due to space limits, we put the full proof in technique report [65] Appendix 6.2. □

3.3.5 Algorithm for Multiple Round Log Annotation. In the multi-round framework, annotation selection can be performed iteratively using Algorithm 1 under a total budget B. The overall performance of LLM will converge once B is large enough to cover all the words. We give a parameter sensitivity experiment in Section 4.4 to investigate the effect of B. To execute Algorithm 1, a strategy is required to determine the budget B_r for each individual round. However, since the LLM operates as a black box, it is not possible to predict the performance improvement of the LLM as the labeled log set is augmented. Alternatively, we design an adaptive strategy base on the number of identified words.

$$B_r = B_{r-1} * (1 - \frac{\triangle W_{r-1}}{W_{r-1}}) \tag{11}$$

$$\triangle W_{r-1} = W_{r-1} - W_{r-2} \tag{12}$$

Instead of manually setting a hyper-parameter for the total number of annotation rounds, Equation (11) adaptively computes B_r based on the previous round budget B_{r-1}. Specifically, in the $r-1$-th round, the number of identified words is denoted as W_{r-1}. $\triangle W_{r-1}$ denotes the increment of words in the $r-1$-th round. By this definition, if the number of words in the $r-1$-th round increases compared to its previous round, LLMLog will annotate fewer logs in the current round. On the contrary, if the number of words in the $r-1$-th round decreases, LLMLog will annotate more logs in the current round. Since the adaptive budget strategy requires annotation results from the previous two rounds, we intuitively set the annotation budget for the first two rounds.

Algorithm 1: Annotation selection at the r-th round

Input: Annotation budget B_r, unlabelled logs U, previous LLM prediction $\hat{T}_{r-1}^U$ and LLM f_θ
Output: Selected logs L_r for annotation
1 $L_r \leftarrow \emptyset$
2 **for** $s \in U$ **do**
3     $\forall s_i \in U \setminus s$, $SED(s, s_i) \leftarrow$Equation (2)
4     $I_s \leftarrow$ Equation (4)
5     $C(s, \hat{t}, \hat{s}) \leftarrow$ Equation (7)
6 **while** $|L_r| < B_r$ **do**
7   **for** $s \in U$ **do**
8         $IS(L_r \cup \{s\}) \leftarrow$Equation (8)
9         $\triangle IS(s|L_r) = IS(L_r \cup \{s\}) - IS(L_r)$
10     $s^* = argmax_{s \in U} \triangle IS(s|L_r)$
11     $L_r = L_r \cup s^*$
12     $U = U \setminus s^*$
13 **return** L_r

As illustrated in Algorithm 2, in line 1, we initialize the annotated labeled log set using the Determinantal Point Process (DPP)[10, 74], following the approach in Divlog[74] to select the most diverse logs. After the initialization process, we iteratively perform the following steps: First, in lines 5 to 7, we execute Algorithm 3 to retrieve the demonstration set D_s for each $s \in U$. Then, we feed D_s and s into f_θ to obtain the predicted template $\hat{t}$. After obtaining the prediction result, we determine the annotation budget B_r for the next round based on Equation 11 (line 10-11). Once the annotation budget B_r is determined, we proceed to a new round of annotation at line 12. The labeled log set L and unlabeled log set U are updated after the annotation process (lines 13-14).

3.4 Adaptive Demonstration Selection

After annotating the selected logs $L_r = \{(s_i, t_i)\}_{i=1}^{B_r}$ in the r-th round, we obtain all labeled logs from the first round to the r-th round as $L = \cup_{i=1}^{r} L_i$. For each unlabeled log $s \in U$, we select a set of demonstrations $D_s \subseteq L$ to provide contextual information for s. These demonstrations D_s will help the LLM understand the words in each unlabeled log s and generate the correct log template t. To improve efficiency and prevent irrelevant information, it is important to limit the size of the demonstration set D_s. As mentioned in Section 2.2, existing works [45, 58, 74, 79] commonly define a fixed number k and select k demonstration logs for each s. However, a fixed number k is not ideal for every s, as the number of words and the difficulty of each unlabeled log can vary significantly.

Algorithm 2: Multiple Round Log Annotation

Input: Annotation budget B, LLM f_θ, unlabelled logs U, total rounds n
Output: Annotated logs L
Initialize L_1, $\mathcal{K}$, $\mathcal{V}$ by DPP
$r \leftarrow 2$
while $B \geq 0$ **do**
  $\hat{T}_r^U \leftarrow \emptyset$
  for $s \in U$ **do**
    $D_s \leftarrow AdaptiveDemonSelection(s, L)$
    $\hat{t} \leftarrow f_\theta(s, D_s)$
    $\hat{T}_r^U \leftarrow \hat{T}_r^U \cup \hat{t}_s$
  $r \leftarrow r + 1$
  $B_r \leftarrow$ Equation (11)
  $B_r = min(B_r, B)$, $B = B - B_r$
  $L_r \leftarrow AnnotationSelection(B_r, U, \hat{T}_r^U, f_\theta)$
  $L \leftarrow L \cup L_r$
  $U \leftarrow U \setminus L_r$
return L

In this paper, we propose an adaptive demonstration selection approach that does not rely on a fixed number k. Instead, the size of the demonstration set is dynamically adjusted based on the characteristics of each unlabeled log s. Formally, we define the adaptive log demonstration selection problem as follows.

Problem 1 (Adaptive Log Demonstration Selection Problem). *Given an input log $s = (w_1^s, \cdots, w_{|s|}^s)$ and the annotated log set $L = \{(s_i, t_i)\}_{i=1}^{|L|}$, the goal is to select a demonstration set $D_s \subseteq L$ from the annotated log set L by minimizing the following objective.*

$$\min |D_s| \tag{13}$$

$$s.t. \forall\, w_i^s \in s, \exists\, w_k^{s_j} \in s_j \wedge s_j \in D_s,$$
$$\text{such that } cosine(\mathbf{w}_i^s, \mathbf{w}_k^{s_j}) \geq 0 \tag{14}$$

where $\mathbf{w}_i^s \in \mathbb{R}^{d_w}$ is the word embedding of the word w_i^s and $cosine(\cdot)$ is the cosine similarity measurement.

Theorem 3. *The adaptive log demonstration selection is NP-hard.*

Proof Sketch. We prove Theorem 3 by reducing our problem to the Set-cover problem [13]. Due to space limits, we provide the full proof in the technical report [65] Appendix 6.3. □

3.4.1 Adaptive Context Selection. As demonstrated in Theorem 3, the adaptive log demonstration selection problem is *NP-hard*. To address this, we propose a greedy algorithm with a theoretical guarantee. As shown in Algorithm 3, the *Adaptive Demonstration Selection Algorithm* aims to select a subset of labeled logs $D_s \subseteq L$ from a pool of labeled logs L that are most relevant to a given unlabeled log s. The algorithm begins by initializing an empty set of selected logs D_s and an empty set of union words $UW(D_s)$ (line 1-2). It iteratively selects the most relevant labeled log from L to include in D_s until no additional contribution from labeled logs L.

Algorithm 3: Adaptive Demonstration Selection

Input: Unlabeled log s and all labeled logs L.
Output: Selected demonstrated logs $D_s \subseteq L$ for s
1 $D_s = \emptyset$
2 $UW(D_s) = \emptyset$
3 **while** ***True*** **do**
4   **for** $s_i \in L$ **do**
5     $UW(s_i|s) = \emptyset$
6   **for** $w_j^s \in s$ **do**
7       $w_k^{s_i*} = \arg\max_{w_k^{s_i} \in s_i} cosine(\mathbf{w}_k^{s_i}, \mathbf{w}_j^s)$
8   **if** $cosine(\mathbf{w}_k^{s_i*}, \mathbf{w}_j^s) \geq 0$ **then**
9         $UW(s_i|s) = UW(s_i|s) \cup \{w_k^{s_i*}\}$
10     $UW(D_s \cup s_i) = UW(D_s) \cup UW(s_i|s)$
11     $\triangle UW(s_i|D_s) = UW(D_s \cup s_i) - UW(D_s)$
12   **if** $|\triangle UW(s_i|D_s)|$=*0* **then**
13   **break**
14   **if** $\forall s_i \in L$, $|\triangle UW(s_i|D_s)|$=*0* **then**
15   **break**
16   $s_i^* = argmax_{s_i \in L} \triangle UW(s_i|D_s)$
17   $D_s \leftarrow D_s \cup s_i^*$
18   $L \leftarrow L \setminus s_i^*$
19 **return** D_s

Specifically, at each iteration, the algorithm evaluates every labeled log $s_i \in L$. For each word w_j^s in the unlabeled log s, it identifies the most similar token $w_k^{s_i*}$ in the labeled log $s_i \in L$ using cosine similarity (line 6-7). Words are considered similar if their cosine similarity score is greater than or equal to 0 (line 8). If the word in s_i meets this threshold, it is added to the set of matched word set $UW(s_i|s)$ (line 9). After processing all words in s, the algorithm merges current similar words $UW(D_s)$ in selected logs D_s with the words in $UW(s_i|s)$ (line 10). The contribution of s_i to s regarding the selected log D_s as $\triangle UW(s_i|D_s) = UW(D_s \cup s_i) - UW(D_s)$ (line 11). If the larges contribution among logs $s_i \in L$ (i.e., $|\triangle UW(s_i|D_s)|$ is zero), the algorithm stops processing that log, as it adds no new information (line 12-13). If all remaining labeled logs $s_i \in L$ contributes no information to s, the algorithm terminates (line 14-15). Otherwise, we add s_i^* to the selected set D_s, and remove it from the pool of labeled logs L (line 16-18).

Theorem 4. *Algorithm 3 has an approximation ratio of* $1 + ln(n)$.

Proof Sketch. We prove that $\triangle UW(s_i|D_s)$ is monotone increasing and submodular. Then, according to [21],the approximation ratio is $1 + \ln(n)$. We provide the full proof in [65] Appendix 6.4. □

4 EXPERIMENTS

4.1 Experiment Setting

4.1.1 Datasets. We use the widely-used log template benchmark over 16 domains provided by Log-PAI [87] with their statistics summarized in Table 2. In each domain, there are 2,000 logs labeled with ground-truth templates and a unique ID [74, 87].

4.1.2 Metrics. We use three metrics to evaluate the effectiveness of template generation from logs, message level accuracy (MLA), precision template accuracy (PTA) and recall template accuracy (RTA). MLA is to measure the effectiveness of template generation in log level while PTA and RTA evaluate it at template levels.

- **Message Level Accuracy (MLA). MLA** is defined as the ratio of logs whose templates are *correctly generated* to the total number of logs [31]. Formally, given the unlabeled logs S, **MLA** is defined as $\frac{\sum_{s_i \in S} \mathbb{I}(\hat{t}_i = t_i)}{|S|}$, where $\mathbb{I}(\hat{t}_i = t_i) = 1$ if the predicted template $\hat{t}_i$ of each unlabeled log $s_i \in S$ is the same as its ground truth t_i.
- **Precision Template Accuracy (PTA). PTA** is the ratio of *correctly generated* templates to all generated templates, where *correctly generated* refers to the template whose corresponding logs are all correctly predicted. Formally, given the generated templates $\hat{T}$, **PTA** is $\frac{\sum_{\hat{t} \in \hat{T}} f\left(\bigwedge_{s_i \in \text{logs}(\hat{t})} \mathbb{I}(\hat{t}_i = \hat{t})\right)}{|\hat{T}|}$, where $f(\cdot) = 1$ when the predicted template $\hat{t}_i$ of each log $s_i \in \text{logs}(\hat{t})$ is correctly predicted as $\hat{t}$.
- **Recall Template Accuracy (RTA). RTA** is the ratio of ground truth templates for which all corresponding logs are correctly predicted to the total number of ground truth templates. Formally, given ground truth templates T, **RTA** is $\frac{\sum_{t \in T} f\left(\bigwedge_{s_i \in \text{logs}(t)} \mathbb{I}(\hat{t}_i = t)\right)}{|T|}$, where $f(\cdot) = 1$ when the predicted template $\hat{t}_i$ of each log $s_i \in \text{logs}(t)$ is correctly predicted as t.

Accuracy in template level is tighter than MLA as they require all corresponding logs are correctly generated, which are suitable to evaluate the effectiveness for large-scaled system logs [74].

4.1.3 Baselines. We select Drain [23] and LogPPT [33] as representative for heuristic-based methods and NN-based methods respectively. We also include the LLM-based method, Divlog [74], which is the SOTA approach on template generation from log. Besides existing template generation methods, we adopt existing multiple-round annotation algorithm, AdaICL [45] to Divlog, forming a new baseline namely AdaICL. For *apple-to-apple* comparison, we select GPT-4o [51] as the base LLM for all LLM-based baselines and our proposed framework.

4.1.4 Implementation and Hyperparameter Setting. In our experiments, our proposed LLMLog and all baselines are implemented in Python 3.9. For LLMs, we use GPT-4o [51] and Qwen2.5-7B-Instruct [64] as our LLM backbones to conduct experiments due to their superior capabilities. Specifically, for our proposed model LLMLog and all ICL-based baselines, including Divlog [74] and AdaICL [45], we set the total budget $B = 50$ on five datasets, including HDFS, Proxifier, Apache, HPC and Windows, since they have fewer templates and words as shown in Table 2. For other datasets with more templates and words, we set the total budget $B = 200$ instead. Specifically, for single-round annotation method Divlog, we perform DPP [10] algorithm to select 50 or 200 labeled logs following [74]. For multiple-round with fixed budget method AdaICL, the budget per round is 10 for $B = 50$ and 40 for $B = 200$, respectively [45]. In terms of LLMLog, we need manually set startup two rounds for adaptive budget which is $B_0 = 10$, $B_1 = 10$ for $B = 50$ and $B_0 = 50$, $B_1 = 25$ for $B = 200$ respectively. Also, for baselines [23, 33, 74], we maintain the default settings following [74] with labeled demonstration log number $k_c = 5$. For our LLMLog, we set $\lambda = 0.5$ in Equation (8) and $\delta = 0.5$ for all dataset.

Table 2: Statistics for sixteen log datasets.

Dataset	Templates#	Logs#	Words#
Android	165	437	857
BGL	120	1367	2008
Hadoop	114	734	979
HDFS	14	2000	2960
Linux	118	290	667
Mac	341	1185	3136
Thunderbird	149	339	676
Zookeeper	50	693	959
HealthApp	75	1179	1682
Spark	36	1699	1360
Windows	50	963	1185
OpenSSH	27	729	692
OpenStack	43	1548	1484
Proxifier	8	1056	2284
HPC	46	381	485
Apache	5	886	907

All experiments are conducted on CentOS 7 with a 20-core Intel(R) Xeon(R) Silver4210 CPU@2.20GHz, 8 NVIDIA GeForce RTX 2080 Ti GPUs (11G), and 92G of RAM.

4.2 Main Results

For main experiments, we evaluate the effectiveness by reporting MLA, PTA and RTA over 16 datasets on GPT-4o in Table 3. In terms of efficiency, we report the average template generation time for unlabeled logs and API cost for LLM-based approaches. Under the same dataset, the bold number indicates the best performance.

4.2.1 Effectiveness Evaluation. We compare our LLMLog with state-of-the-art baselines using three metrics: MLA, PTA, and RTA on sixteen datasets, as shown in Table 3. Regarding MLA, LLMLog and AdaICL outperform DivLog by leveraging the benefits of multi-round annotation, enabling more effective and accurate log processing. Our LLMLog outperforms AdaICL in terms of MLA on all datasets. The reason is that, for HDFS and Proxifier with fewer templates, SED in LLMLog outperforms traditional cosine similarity in AdaICL by generating a higher-quality labeled log set for annotation. These annotated logs serve as effective demonstrations for a large number of unlabeled logs. Additionally, the adaptive demonstration strategy in LLMLog ensures that each word in the unlabeled logs is accurately processed, achieving higher accuracy even with a limited budget. For datasets with diverse templates and words, such as Mac, LLMLog reduces redundant contexts, providing clear and sufficient context for each unlabeled log compared to fixed top-k_c demonstrations.

In terms of template-level accuracy, the PTA and RTA metrics tend to be lower than MLA since they require all logs belonging to a template to be correctly predicted. Specifically, if more words are mistakenly generated in predicted templates, PTA will drop because the total number of predicted templates increases. Therefore, the lower PTA of DivLog and AdaICL compared to our proposed LLMLog reflects the false generation of word types, indicating that they

Table 3: Effectiveness (accuracy) over 16 log datasets on GPT-4o. The bold number indicates the best performance.

Dataset	Drain			LogPPT			DivLog			AdaICL			LLMLog (Ours)		
	MLA	PTA	RTA	MLA	PTA	RTA	MLA	PTA	RTA	MLA	PTA	RTA	MLA	PTA	RTA
Android	73.0	56.6	62.0	76.7	58.4	68.4	63.8	58.9	68.4	97.8	89.4	92.1	**99.6**	**94.6**	**96.4**
BGL	44.4	33.9	30.8	97.0	68.6	78.3	94.0	68.4	77.5	99.4	93.5	95.8	**99.9**	**95.1**	**98.3**
Hadoop	43.9	36.8	34.2	89.5	54.0	58.8	89.0	69.3	85.1	99.4	92.2	97.4	**100.0**	**100.0**	**100.0**
HDFS	95.9	81.3	92.9	90.2	85.7	85.7	**100.0**	**100.0**	**100.0**	99.9	86.7	92.9	**100.0**	**100.0**	**100.0**
Linux	19.4	43.4	42.2	94.9	47.5	49.1	97.3	92.4	93.2	99.7	96.6	96.6	**99.8**	**96.6**	**98.3**
Mac	27.2	21.2	24.9	67.3	43.6	53.4	62.4	48.3	64.5	93.2	74.4	82.1	**96.0**	**77.1**	**85.9**
Thunderbird	19.1	29.9	36.9	92.6	50.6	59.1	88.9	86.8	92.6	98.9	83.3	90.6	**99.9**	**93.3**	**98.7**
Zookeeper	49.8	39.1	36.0	99.0	74.1	86.0	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
HealthApp	24.1	8.3	34.7	78.9	85.3	85.3	99.9	98.7	98.7	99.9	98.7	98.7	**100.0**	**100.0**	**100.0**
Spark	37.6	50.0	41.7	99.1	60.0	58.3	82.1	48.3	77.8	99.9	97.2	97.2	**100.0**	**100.0**	**100.0**
Windows	69.6	46.3	50.0	98.3	55.4	72.0	97.6	55.9	76.0	99.9	92.3	96.0	**100.0**	**100.0**	**100.0**
OpenSSH	53.4	52.0	50.0	97.6	48.9	84.6	99.9	96.3	96.3	99.9	96.3	96.3	**100.0**	**100.0**	**100.0**
OpenStack	18.0	5.5	39.5	90.7	84.4	88.4	96.9	74.0	88.4	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
Proxifier	52.7	26.9	87.5	**100.0**	**100.0**	**100.0**	96.5	14.3	75.0	99.9	77.8	87.5	**100.0**	**100.0**	**100.0**
HPC	67.2	38.8	41.3	94.7	73.6	84.8	97.5	42.6	87.0	98.6	62.5	97.8	**100.0**	**100.0**	**100.0**
Apache	**100.0**	**100.0**	**100.0**	99.4	83.3	83.3	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**

provide low-quality contexts to unlabeled logs. LLMLog ensures higher-quality contexts through both the SED-based representative score and the adaptive demonstration strategy. Thus, LLMLog performs better than the baselines on the PTA metric. As for RTA, it evaluates how many templates in a dataset are correctly predicted. Though Drain and LogPPT achieve 100 percent accuracy on Apache and Proxifier, respectively, their low RTA on other datasets illustrates their low generalization abilities. In summary, LLMLog outperforms AdaICL on PTA and RTA, proving its suitability for large-scale log datasets.

4.2.2 *Efficiency Evaluation.* We compare the template generation efficiency for LLMLog with baseline LLM-based methods. Template generation efficiency is measured by average LLM prediction time for single log on each dataset, summarized in Table 4. The average LLM prediction time for LLMLog is less than 0.8 seconds on all 16 datasets while the time for DivLog and AdaICL is larger than 1 second. As all three methods are measured under the same API, the less time consumption reflects the fewer number of input tokens. The adaptive demonstration selection minimizes the number of example logs based on word coverage, which reduces the number of input tokens in contextual demonstration compared to fixed top-k_c logs in DivLog and AdaICL. On datasets with fewer words and templates like Apache, Windows and HPC, adaptive demonstration can cover all the words within 1 or 2 log. Compared to $k_c = 5$ setting in DivLog and AdaICL, the generation time of our proposed LLMLog is significantly decreased. On datasets with more words and templates like mac, it requires more example logs to cover the words. The difference of generation time between AdaICL and LLMLog is less than that on simpler datasets. API Cost experiments also prove the claim.

4.2.3 *API Cost Evaluation.* We compare the total API monetary costs for LLMLog and the state-of-the-art baselines, DivLog [74] and AdaICL [45], as summarized in Table 4. The cost of GPT-4o is approximately $3.6 USD per one million tokens. In Table 4, the cost

Table 4: Template generation time (time with seconds) and API Cost (in USD) across 16 log datasets. The bold number indicates the most efficient and lower API cost results.

Dataset	Generation Time (s)			API Cost (USD)		
	DivLog	AdaICL	LLMLog	DivLog	AdaICL	LLMLog
Android	1.1	1.1	**0.7**	3.5	3.5	**2.1**
BGL	1.4	1.1	**0.8**	3.8	3.7	**2.8**
Hadoop	1.1	1.0	**0.6**	3.7	3.7	**2.0**
HDFS	1.0	1.0	**0.7**	7.6	7.6	**5.1**
Linux	1.2	1.1	**0.7**	3.1	3.1	**2.0**
Mac	1.1	1.1	**0.8**	5.5	5.4	**5.0**
Thunderbird	1.1	1.0	**0.6**	3.3	3.2	**2.7**
Zookeeper	1.1	1.1	**0.3**	3.1	3.1	**2.1**
HealthApp	1.8	1.4	**0.8**	4.4	3.7	**2.3**
Spark	2.1	1.5	**1.4**	6.5	5.7	**3.3**
Windows	2.2	**0.7**	**0.7**	5.3	5.3	**2.6**
OpenSSH	1.1	1.1	**0.7**	7.8	7.8	**2.5**
OpenStack	1.8	1.9	**1.0**	9.8	10.0	**4.2**
Proxifier	0.9	0.9	**0.8**	5.7	5.7	**3.5**
HPC	1.6	0.5	**0.3**	2.6	1.8	**1.3**
Apache	1.8	0.5	**0.4**	2.7	2.4	**1.6**

of our LLMLog ranges from $1 to $5 USD per dataset, where the cost of processing each log is only $0.0025-$0.005 USD. Therefore, the cost of LLMLog is both cheap and practical. Moreover, LLMLog incurs less API cost than the state-of-the-art baselines, as it adaptively selects the number of labeled logs to use as demonstrations for each unlabeled log, whereas AdaICL relies on a fixed number of demonstrations. By eliminating unnecessary log demonstrations, LLMLog significantly reduces the input token length for LLMs, further lowering the computational cost. LLMLog can effectively reduce costs for simple datasets like HDFS and Proxifier since the words can be adequately covered by one or two logs. As for complex datasets like Mac, the amount of cost savings is smaller while still better than the baselines, as each unlabeled log requires more example logs to cover the words.

Table 5: Ablation study on GPT-4o.

Model / Dataset	Mac			BGL			Hadoop			Proxifier		
	MLA	PTA	RTA	MLA	PTA	RTA	MLA	PTA	RTA	MLA	PTA	RTA
LLMLog\SED	$93.7_{(-3.3)}$	$65.6_{(-15.7)}$	$77.1_{(-9.1)}$	$87.2_{(-12.7)}$	$35.3_{(-61.4)}$	$52.5_{(-45.8)}$	$95.5_{(-4.5)}$	$85.4_{(-14.6)}$	$92.1_{(-7.9)}$	$75.7_{(-24.3)}$	$37.5_{(-62.5)}$	$37.5_{(-62.5)}$
LLMLog\RS	$93.9_{(-3.1)}$	$66.5_{(-14.8)}$	$77.7_{(-8.5)}$	$94.5_{(-5.4)}$	$83.0_{(-13.7)}$	$89.2_{(-9.1)}$	$94.1_{(-5.9)}$	$82.9_{(-17.1)}$	$97.4_{(-2.6)}$	$99.4_{(-0.6)}$	$37.5_{(-62.5)}$	$37.5_{(-62.5)}$
LLMLog\PC	$96.0_{(-1.0)}$	$77.10_{(-4.2)}$	$85.9_{(-0.3)}$	$99.0_{(-0.9)}$	$93.5_{(-3.2)}$	$95.8_{(-2.5)}$	$99.5_{(-0.5)}$	$98.3_{(-1.7)}$	$99.1_{(-0.9)}$	$100.0_{(-0.0)}$	$100.0_{(-0.0)}$	$100.0_{(-0.0)}$
LLMLog\AD	$93.1_{(-3.9)}$	$67.6_{(-13.7)}$	$76.3_{(-9.9)}$	$97.3_{(-2.6)}$	$93.5_{(-3.2)}$	$96.7_{(-1.6)}$	$99.5_{(-0.5)}$	$92.6_{(-7.4)}$	$98.2_{(-1.8)}$	$100.0_{(-0.0)}$	$100.0_{(-0.0)}$	$100.0_{(-0.0)}$
LLMLog\AB	$96.9_{(-0.1)}$	$80.2_{(-1.1)}$	$86.2_{(-0.0)}$	$97.6_{(-3.2)}$	$90.6_{(-6.1)}$	$96.7_{(-1.6)}$	$100.0_{(-0.0)}$	$100.0_{(-0.0)}$	$100.0_{(-0.0)}$	$100.0_{(-0.0)}$	$100.0_{(-0.0)}$	$100.0_{(-0.0)}$
LLMLog	**97.0**	**81.3**	**86.2**	**99.9**	**96.7**	**98.3**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**

4.3 Ablation Study

This subsection analyzes the impact of components in LLMLog. For the log annotation, it introduces the SED metric to calculate the representative score of logs. Logs are selected for annotation by optimizing a weighted combination of the LLM's prediction confidence and the representative score. In the adaptive demonstration selection component, Algorithm 3 is proposed to adaptively determine suitable contexts for each unlabeled log. Additionally, an adaptive budget strategy (Equation (11)) dynamically allocates the annotation budget for each round. To verify these designs, we denote LLMLog with SED replaced by cosine similarity as **LLMLog\SED**, without the representative score as **LLMLog\RS**, without LLM prediction confidence as **LLMLog\PC**, with Algorithm 3 replaced by a fixed top-k_c strategy as **LLMLog\AD**, and with Equation (11) replaced by a fixed budget for each round as **LLMLog\AB**. For **LLMLog\AB**, we set the budget for each round to 10 for Proxifier and 40 for Mac, BGL, and Hadoop. We conduct experiments on 4 datasets with different template distributions, including Mac, BGL, Hadoop, and Proxifier.

As shown in Table 5, **LLMLog\SED** suffers from a significant accuracy drop because SED eliminates redundant words in logs, treating logs with the same template as similar. Regarding the representative score, the performance decrease of **LLMLog\RS** shows that logs similar to the majority provide useful contexts for generating templates. On the other hand, **LLMLog\PC** performs slightly better by selecting challenging logs for the LLM, but these logs are less representative, making their overall impact smaller. Regarding the adaptive demonstration strategy, **LLMLog\AD** shows reduced accuracy as template complexity increases. While a fixed top-k_c strategy works well for simpler datasets, it struggles on complex datasets like Mac, where insufficient context leads to irregular processing and more generated templates. This results in a larger accuracy drop compared to simpler datasets like PTA. For the adaptive budget strategy, **LLMLog\AB** shows that both fixed and adaptive strategies perform well on simpler datasets. However, in complex datasets with more templates and words, the adaptive strategy limits annotations per round, selecting labeled logs that are more diverse in word count and template variety.

4.4 Parameter Sensitivity

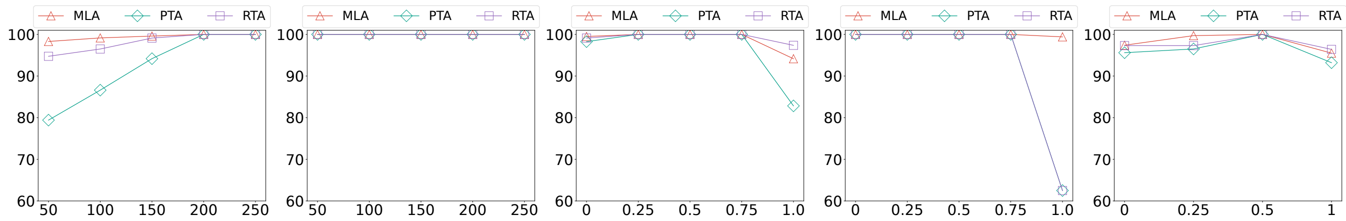
We evaluate the effectiveness of different hyper-parameter settings of LLMLog over two datasets, Hadoop and Proxifier.

4.4.1 Annotation Budget B in Algorithm 3 . The annotation budget B represents the total budget for human annotation. We vary B within $\{50, 100, 150, 200, 250\}$. On the Hadoop dataset in Figure 3 (a), the three accuracy metrics increase first and then stabilize at $B = 200$, indicating $B = 200$ is sufficient to cover most templates on Hadoop. Besides, the increment of PTA is larger than other two metrics due to the LLM falsely generating incorrect word types under insufficient B. As Proxifier is a much simpler dataset containing only 4 templates in Figure 3 (b), all three metrics stabilize at $B = 50$ which can provide sufficient contextual information for each word in unlabled logs. To sum up, the performance stabilizes after B is large enough, rather than peaking, making tuning easier.

4.4.2 The weight λ in Equation (8). λ is a trade-off parameter between the representative score and the LLM prediction confidence. We vary $\lambda \in \{0, 0.25, 0.5, 0.75, 1\}$. As demonstrated in Figure 3(c), MLA, PTA, and RTA initially increase and subsequently decrease over Hadoop, peaking within 0.25 to 0.75. Small value of λ undervalues the impact of LLM confidence, resulting in the selection of annotation logs with redundant information. In contrast, large λ prioritizes logs with low template generation confidence. In Figure 3 (d), most logs are easily identified using representative contextual information, achieving 100% accuracy even when $\lambda = 0$ over Proxifier. However, relying solely on LLM prediction confidence ($\lambda = 1$) causes LLMLog to focus only on low-confidence logs which is not appropriate even over simple datasets.

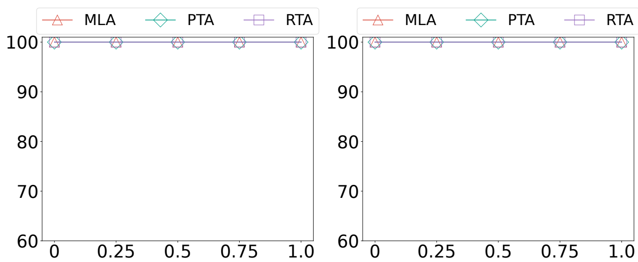
4.4.3 The threshold δ in Equation (4). δ controls the threshold of representative score. We vary it within $\{0, 0.25, 0.5, 1.0\}$. As shown in Figure 3 (e), MLA, PTA, and RTA exhibit a trend of rising and then falling, roughly peaking at $\delta = 0.5$. A low threshold underestimates the informativeness of logs. The annotation focuses excessively on LLM confidence. On the contrary, a high threshold causes only a subset of unlabeled logs obtaining enough context.

4.4.4 Cosine Similarity Threshold in Equation (3) *and* (13). Cosine similarity $cosine(\cdot)$ measures word similarity. Two words are considered similar if their embedding cosine similarity is greater than the threshold 0. We vary it within $\{0, 0.25, 0.5, 0.75, 1\}$. As shown in Figure 4 (a) and (b), the effect of varying threshold of cosine similarity has converged over two datasets. This implies that LLMLog is robust to different settings of word similarity under a certain budget. Since the total number of distinct words is relatively small in system events, it is sufficient to distinguish words by 0.



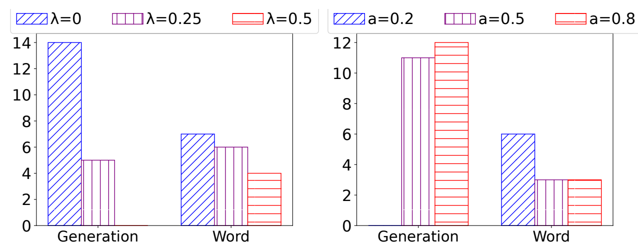
(a) B over Hadoop (b) B over Proxifier (c) λ over Hadoop (d) λ over Proxifier (e) δ over Hadoop

Figure 3: Parameter sensitivity evaluations



(a) Word similarity over Hadoop (b) Word similarity over Proxifier

Figure 4: Parameter sensitivity for word similarity



(a) Error cases with λ (b) Error cases with a

Figure 5: Confidence score on hallucination errors

4.5 Case Study

Hallucination is that LLM generates outputs without following the prompts, which is a common problem in LLM-related tasks [28, 37, 45, 74]. There are mainly two types of error caused by hallucination. The first is **generation error** that LLM falsely generates or deletes words in input logs. For instance, input log is `rts: kernel terminated for reason 1004` with the ground truth `rts: kernel terminated for reason [CODE]`. However, LLM may predict `rts: kernel terminated` where `for reason 1004` are falsely deleted. The second case is **word error** that even the type of a target word is included in prompt, LLM still makes wrong predictions. For example, input log is `rts: kernel terminated for reason 1004` and the prompt has instructed to replace `1004` to word type `[CODE]`, LLM still mistakenly remains `1004` in predicted template.

To investigate how confidence score in Equation (7) help alleviate the hallucination issue, we vary two hyperparameters related to the confidence score. One is λ in Equation (8), the trade-off parameter for prediction confidence. The other is a in Equation (7), the trade-off parameter for the effect of token probability in the prediction confidence score. First, we vary $\lambda \in \{0, 0.25, 0.5\}$. As shown in Figure 5 (a), as λ increases, both generation error and word error are reduced, implying that the confidence score can effectively select several "hard" logs, allowing human annotation to replace the LLM's hallucinated output. On the other hand, we vary $a \in \{0.2, 0.5, 0.8\}$. As shown in Figure 5 (b), as a increases, logs with low prediction probability are included in the labeled log set. Thus, the labeled log set becomes effective at preventing word errors associated with low prediction probabilities. However, generation errors increase because several word-inconsistent error logs cannot be selected due to the decreasing weight of word consistency.

5 CONCLUSION

In this paper, we present LLMLog, an LLM-driven multi-round annotation framework with adaptive in-context learning for log template generation. Firstly, we propose a distance metric to measure the log similarity, along with a confidence metric to assess the difficulty faced by LLM. Based on the two metrics, we identify the most valuable unlabeled logs for human annotation in each round. Second, we introduce an adaptive approach for selecting demonstrative contexts of each log to generate more accurate templates by LLM. Experimental results demonstrate that LLMLog achieves superior performance compared to the state-of-the-art baselines.

ACKNOWLEDGMENTS

Lei Chen's work is partially supported by National Key Research and Development Program of China Grant No. 2023YFF0725100, National Science Foundation of China (NSFC) under Grant No. U22B2060, Guangdong-Hong Kong Technology Innovation Joint Funding Scheme Project No. 2024A0505040012, the Hong Kong RGC GRF Project 16213620, RIF Project R6020-19, AOE Project AoE/E-603/18, Theme-based project TRS T41-603/20R, CRF Project C2004-21G, Key Areas Special Project of Guangdong Provincial Universities 2024ZDZX1006, Guangdong Province Science and Technology Plan Project 2023A0505030011, Guangzhou municipality big data intelligence key lab, 2023A03J0012, Hong Kong ITC ITF grants MHX/078/21 and PRP/004/22FX, Zhujiang scholar program 2021JC02X170, Microsoft Research Asia Collaborative Research Grant, HKUST-Webank joint research lab and 2023 HKUST Shenzhen-Hong Kong Collaborative Innovation Institute Green Sustainability Special Fund, from Shui On Xintiandi and the InnoSpace GBA.

REFERENCES

[1] Anunay Amar and Peter C. Rigby. 2019. Mining historical test logs to predict bugs and localize faults in the test logs. In *Proceedings of the 41st International Conference on Software Engineering* (Montreal, Quebec, Canada) *(ICSE '19)*. IEEE Press, 140–151. https://doi.org/10.1109/ICSE.2019.00031

[2] Apache Software Foundation. [n.d.]. *Hadoop*. https://hadoop.apache.org

[3] Anna Arpaci-Dusseau, Zixiang Zhou, and Xuhao Chen. 2025. Accurate and Fast Approximate Graph Pattern Mining at Scale. *Proc. VLDB Endow.* 18, 2 (Feb. 2025), 93–107. https://doi.org/10.14778/3705829.3705831

[4] Tanveer I. Bagban and Prakash J. Kulkarni. 2020. Template Based Clustering of Web Documents Using Locality Sensitive Hashing (LSH). In *Computing in Engineering and Technology*, Brijesh Iyer, P. S. Deshpande, S. C. Sharma, and Ulhas Shiurkar (Eds.). Springer Singapore, Singapore, 567–584.

[5] Parishad BehnamGhader, Vaibhav Adlakha, Marius Mosbach, Dzmitry Bahdanau, Nicolas Chapados, and Siva Reddy. 2024. LLM2Vec: Large Language Models Are Secretly Powerful Text Encoders. In *First Conference on Language Modeling*. https://openreview.net/forum?id=IW1PR7vEBf

[6] Satadisha Saha Bhowmick, Eduard C. Dragut, and Weiyi Meng. 2023. Globally Aware Contextual Embeddings for Named Entity Recognition in Social Media Streams. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. 1544–1557. https://doi.org/10.1109/ICDE55515.2023.00122

[7] Angela Bonifati, Wim Martens, and Thomas Timm. 2017. An analytical study of large SPARQL query logs. *Proc. VLDB Endow.* 11, 2 (Oct. 2017), 149–161. https://doi.org/10.14778/3149193.3149196

[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901. https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[9] Chia-Hui Chang, M. Kayed, M.R. Girgis, and K.F. Shaalan. 2006. A Survey of Web Information Extraction Systems. *IEEE Transactions on Knowledge and Data Engineering* 18, 10 (2006), 1411–1428. https://doi.org/10.1109/TKDE.2006.152

[10] Laming Chen, Guoxin Zhang, and Hanning Zhou. 2018. Fast greedy MAP inference for determinantal point process to improve recommendation diversity. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (Montréal, Canada) *(NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 5627–5638.

[11] Yurong Cheng, Zhaohe Liao, Xiaosong Huang, Yi Yang, Xiangmin Zhou, Ye Yuan, and Guoren Wang. 2024. Cross Online Ride-Sharing for Multiple-Platform Cooperations in Spatial Crowdsourcing. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. 4140–4152. https://doi.org/10.1109/ICDE60146.2024.00317

[12] Robert Christensen and Feifei Li. 2013. Adaptive log compression for massive log data. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data* (New York, New York, USA) *(SIGMOD '13)*. Association for Computing Machinery, New York, NY, USA, 1283–1284. https://doi.org/10.1145/2463676.2465341

[13] V. Chvatal. 1979. A Greedy Heuristic for the Set-Covering Problem. *Math. Oper. Res.* 4, 3 (Aug. 1979), 233–235. https://doi.org/10.1287/moor.4.3.233

[14] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 1285–1298. https://doi.org/10.1145/3133956.3134015

[15] Ju Fan, Guoliang Li, and Lizhu Zhou. 2011. Interactive SQL query suggestion: Making databases user-friendly. In *2011 IEEE 27th International Conference on Data Engineering*. 351–362. https://doi.org/10.1109/ICDE.2011.5767843

[16] Meihao Fan, Xiaoyue Han, Ju Fan, Chengliang Chai, Nan Tang, Guoliang Li, and Xiaoyong Du. 2024. Cost-Effective In-Context Learning for Entity Resolution: A Design Space Exploration . In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, Los Alamitos, CA, USA, 3696–3709. https://doi.org/10.1109/ICDE60146.2024.00284

[17] Jieming Feng, Zhanhuai Li, and Qun Chen. 2024. Towards Exploratory Query Optimization for Template-Based SQL Workloads. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. 151–164. https://doi.org/10.1109/ICDE60146.2024.00019

[18] Raul Castro Fernandez, Aaron J. Elmore, Michael J. Franklin, Sanjay Krishnan, and Chenhao Tan. 2023. How Large Language Models Will Disrupt Data Management. *Proc. VLDB Endow.* 16, 11 (July 2023), 3302–3309. https://doi.org/10.14778/3611479.3611527

[19] Benjamin Feuer, Yurong Liu, Chinmay Hegde, and Juliana Freire. 2024. ArcheType: A Novel Framework for Open-Source Column Type Annotation Using Large Language Models. *Proc. VLDB Endow.* 17, 9 (Aug. 2024), 2279–2292. https://doi.org/10.14778/3665844.3665857

[20] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *Proc. VLDB Endow.* 17, 5 (May 2024), 1132–1145. https://doi.org/10.14778/3641204.3641221

[21] Daniel Golovin and Andreas Krause. 2011. Adaptive submodularity: theory and applications in active learning and stochastic optimization. *J. Artif. Int. Res.* 42, 1 (Sept. 2011), 427–486.

[22] Hossein Hamooni, Biplob Debnath, Jianwu Xu, Hui Zhang, Guofei Jiang, and Abdullah Mueen. 2016. LogMine: Fast Pattern Recognition for Log Analytics. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management* (Indianapolis, Indiana, USA) *(CIKM '16)*. Association for Computing Machinery, New York, NY, USA, 1573–1582. https://doi.org/10.1145/2983323.2983358

[23] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R. Lyu. 2017. Drain: An Online Log Parsing Approach with Fixed Depth Tree. In *2017 IEEE International Conference on Web Services (ICWS)*. 33–40. https://doi.org/10.1109/ICWS.2017.13

[24] Shilin He, Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, Michael R. Lyu, and Dongmei Zhang. 2018. Identifying impactful service system problems via log analysis. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Lake Buena Vista, FL, USA) *(ESEC/FSE 2018)*. Association for Computing Machinery, New York, NY, USA, 60–70. https://doi.org/10.1145/3236024.3236083

[25] Shilin He, Jieming Zhu, Pinjia He, and Michael R. Lyu. 2016. Experience Report: System Log Analysis for Anomaly Detection. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. 207–218. https://doi.org/10.1109/ISSRE.2016.21

[26] Yuncheng Huang, Qianyu He, Jiaqing Liang, Sihang Jiang, Yanghua Xiao, and Yunwen Chen. 2024. Enhancing Quantitative Reasoning Skills of Large Language Models through Dimension Perception . In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, Los Alamitos, CA, USA, 789–802. https://doi.org/10.1109/ICDE60146.2024.00066

[27] Peng Jia, Pinghui Wang, Junzhou Zhao, Ye Yuan, Jing Tao, and Xiaohong Guan. 2021. LogLog Filter: Filtering Cold Items within a Large Range over High Speed Data Streams. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. 804–815. https://doi.org/10.1109/ICDE51399.2021.00075

[28] Wenqi Jiang, Marco Zeller, Roger Waleffe, Torsten Hoefler, and Gustavo Alonso. 2024. Chameleon: a Heterogeneous and Disaggregated Accelerator System for Retrieval-Augmented Language Models. *Proc. VLDB Endow.* 18, 1 (2024), 42–52. https://www.vldb.org/pvldb/vol18/p42-jiang.pdf

[29] Ryan Johnson, Ippokratis Pandis, Radu Stoica, Manos Athanassoulis, and Anastasia Ailamaki. 2010. Aether: a scalable approach to logging. *Proc. VLDB Endow.* 3, 1–2 (Sept. 2010), 681–692. https://doi.org/10.14778/1920841.1920928

[30] Hung-Yu Kao, Shian-Hua Lin, Jan-Ming Ho, and Ming-Syan Chen. 2004. Mining Web informative structures and contents based on entropy analysis. *IEEE Transactions on Knowledge and Data Engineering* 16, 1 (2004), 41–55. https://doi.org/10.1109/TKDE.2004.1264821

[31] Zanis Ali Khan, Donghwan Shin, Domenico Bianculli, and Lionel Briand. 2022. Guidelines for assessing the accuracy of log message template identification techniques. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 1095–1106. https://doi.org/10.1145/3510003.3510101

[32] Samir Khuller, Anna Moss, and Joseph (Seffi) Naor. 1999. The budgeted maximum coverage problem. *Inf. Process. Lett.* 70, 1 (April 1999), 39–45. https://doi.org/10.1016/S0020-0190(99)00031-9

[33] Van-Hoang Le and Hongyu Zhang. 2023. Log Parsing with Prompt-Based Few-Shot Learning. In *Proceedings of the 45th International Conference on Software Engineering* (Melbourne, Victoria, Australia) *(ICSE '23)*. IEEE Press, 2438–2449. https://doi.org/10.1109/ICSE48619.2023.00204

[34] Guoliang Li, Xuanhe Zhou, and Xinyang Zhao. 2024. LLM for Data Management. *Proceedings of the VLDB Endowment* 17, 12 (2024), 4213–4216.

[35] Haoyang Li, Yiming Li, Anxin Tian, Tianhao Tang, Zhanchao Xu, Xuejia Chen, Nicole Hu, Wei Dong, Qing Li, and Lei Chen. 2024. A survey on large language model acceleration based on kv cache management. *arXiv preprint arXiv:2412.19442* (2024).

[36] Xiaoyun Li, Pengfei Chen, Linxiao Jing, Zilong He, and Guangba Yu. 2020. SwissLog: Robust and Unified Deep Learning Based Log Anomaly Detection for Diverse Faults. 92–103. https://doi.org/10.1109/ISSRE5003.2020.00018

[37] Zhaodonghui Li, Haitao Yuan, Huiming Wang, Gao Cong, and Lidong Bing. 2024. LLM-R2: A Large Language Model Enhanced Rule-based Rewrite System for Boosting Query Efficiency. *Proc. VLDB Endow.* 18, 1 (2024), 53–65. https://www.vldb.org/pvldb/vol18/p53-yuan.pdf

[38] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. 2016. Log clustering based problem identification for online service systems. In

Proceedings of the 38th International Conference on Software Engineering Companion (Austin, Texas) *(ICSE '16)*. Association for Computing Machinery, New York, NY, USA, 102–111. https://doi.org/10.1145/2889160.2889232

[39] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. *ACM Comput. Surv.* 55, 9, Article 195 (Jan. 2023), 35 pages. https://doi.org/10.1145/3560815

[40] Xinfu Liu, Yirui Wu, Yuting Zhou, Junyang Chen, Huan Wang, Ye Liu, and Shaohua Wan. 2024. Enhancing Large Language Models with Multimodality and Knowledge Graphs for Hallucination-free Open-set Object Recognition. *Proceedings of the VLDB Endowment. ISSN* 2150 (2024), 8097.

[41] Yudong Liu, Xu Zhang, Shilin He, Hongyu Zhang, Liqun Li, Yu Kang, Yong Xu, Minghua Ma, Qingwei Lin, Yingnong Dang, S. Rajmohan, and Dongmei Zhang. 2022. UniParser: A Unified Log Parser for Heterogeneous Log Data. *Proceedings of the ACM Web Conference 2022* (2022). https://api.semanticscholar.org/CorpusID:246822534

[42] Adetokunbo Makanju, A. Nur Zincir-Heywood, and Evangelos E. Milios. 2012. A Lightweight Algorithm for Message Type Extraction in System Application Logs. *IEEE Transactions on Knowledge and Data Engineering* 24, 11 (2012), 1921–1936. https://doi.org/10.1109/TKDE.2011.138

[43] Markos Markakis, Brit Youngmann, Trinity Gao, Ziyu Zhang, Rana Shahout, Peter Baile Chen, Chunwei Liu, Ibrahim Sabek, and Michael Cafarella. 2025. From Logs to Causal Inference: Diagnosing Large Systems. *Proc. VLDB Endow.* 18, 2 (Feb. 2025), 158–172. https://doi.org/10.14778/3705829.3705836

[44] Wim Martens, Matthias Niewerth, Tina Popp, Carlos Rojas, Stijn Vansummeren, and Domagoj Vrgoč. 2023. Representing Paths in Graph Database Pattern Matching. *Proc. VLDB Endow.* 16, 7 (March 2023), 1790–1803. https://doi.org/10.14778/3587136.3587151

[45] Costas Mavromatis, Balasubramaniam Srinivasan, Zhengyuan Shen, Jiani Zhang, Huzefa Rangwala, Christos Faloutsos, and George Karypis. 2023. Which Examples to Annotate for In-Context Learning? Towards Effective and Efficient Selection. arXiv:2310.20046 [cs.CL] https://arxiv.org/abs/2310.20046

[46] Lang Mei, Jiaxin Mao, and Ji-Rong Wen. 2024. Optimizing Probabilistic Box Embeddings with Distance Measures . In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, Los Alamitos, CA, USA, 5088–5100. https://doi.org/10.1109/ICDE60146.2024.00106

[47] Masayoshi Mizutani. 2013. Incremental Mining of System Log Format. In *2013 IEEE International Conference on Services Computing*. 595–602. https://doi.org/10.1109/SCC.2013.73

[48] Animesh Nandi, Atri Mandal, Shubham Atreja, Gargi B. Dasgupta, and Subhrajit Bhattacharya. 2016. Anomaly Detection Using Program Control Flow Graph Mining From Execution Logs. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 215–224. https://doi.org/10.1145/2939672.2939712

[49] Sasho Nedelkoski, Jasmin Bogatinovski, Alexander Acker, Jorge Cardoso, and Odej Kao. 2021. Self-supervised Log Parsing. In *Machine Learning and Knowledge Discovery in Databases: Applied Data Science Track*, Yuxiao Dong, Dunja Mladenić, and Craig Saunders (Eds.). Springer International Publishing, Cham, 122–138.

[50] Reham Omar, Ishika Dhall, Panos Kalnis, and Essam Mansour. 2023. A Universal Question-Answering Platform for Knowledge Graphs. *Proceedings of the ACM on Management of Data* 1 (2023), 1 – 25. https://api.semanticscholar.org/CorpusID:257254920

[51] OpenAI. 2024. Introducing gpt-4o: our fastest and most affordable flagship model. https://platform.openai.com/docs/guides/vision 2024-11-07.

[52] Abdelghny Orogat and Ahmed El-Roby. 2023. Maestro: Automatic Generation of Comprehensive Benchmarks for Question Answering Over Knowledge Graphs. *Proc. ACM Manag. Data* 1, 2, Article 177 (June 2023), 24 pages. https://doi.org/10.1145/3589322

[53] Zhencan Peng, Zhizhi Wang, and Dong Deng. 2023. Near-Duplicate Sequence Search at Scale for Large Language Model Memorization Evaluation. *Proc. ACM Manag. Data* 1, 2, Article 179 (June 2023), 18 pages. https://doi.org/10.1145/3589324

[54] Tonghui Ren, Yuankai Fan, Zhenying He, Ren Huang, Jiaqi Dai, Can Huang, Yinan Jing, Kai Zhang, Yifan Yang, and X. Sean Wang. 2024. PURPLE: Making a Large Language Model a Better SQL Writer . In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, Los Alamitos, CA, USA, 15–28. https://doi.org/10.1109/ICDE60146.2024.00009

[55] Cedric Renggli, Xiaozhe Yao, Luka Kolar, Luka Rimanic, Ana Klimovic, and Ce Zhang. 2022. SHiFT: an efficient, flexible search engine for transfer learning. *Proc. VLDB Endow.* 16, 2 (Oct. 2022), 304–316. https://doi.org/10.14778/3565816.3565831

[56] Shreya Shankar, Haotian Li, Parth Asawa, Madelon Hulsebos, Yiming Lin, J. D. Zamfirescu-Pereira, Harrison Chase, Will Fu-Hinthorn, Aditya G. Parameswaran, and Eugene Wu. 2024. spade: Synthesizing Data Quality Assertions for Large Language Model Pipelines. *Proc. VLDB Endow.* 17, 12 (Nov. 2024), 4173–4186. https://doi.org/10.14778/3685800.3685835

[57] Jie Song and Yeye He. 2021. Auto-Validate: Unsupervised Data Validation Using Data-Domain Patterns Inferred from Data Lakes. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 1678–1691. https://doi.org/10.1145/3448016.3457250

[58] Hongjin SU, Jungo Kasai, Chen Henry Wu, Weijia Shi, Tianlu Wang, Jiayi Xin, Rui Zhang, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. 2023. Selective Annotation Makes Language Models Better Few-Shot Learners. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=qY1hlv7gwg

[59] Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çağatay Demiralp, Chen Chen, and Wang-Chiew Tan. 2022. Annotating Columns with Pre-trained Language Models. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 1493–1503. https://doi.org/10.1145/3514221.3517906

[60] Yushi Sun, Wang Jiachuan, Peng Cheng, Libin Zheng, Lei Chen, and Jian Yin. 2024. Cross-Domain-Aware Worker Selection with Training for Crowdsourced Annotation. 249–262. https://doi.org/10.1109/ICDE60146.2024.00026

[61] Yushi Sun, Hao Xin, and Lei Chen. 2023. RECA: Related Tables Enhanced Column Semantic Type Annotation Framework. *Proc. VLDB Endow.* 16, 6 (Feb. 2023), 1319–1331. https://doi.org/10.14778/3583140.3583149

[62] Liang Tang, Tao Li, and Chang-Shing Perng. 2011. LogSig: generating system events from raw textual logs. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management* (Glasgow, Scotland, UK) *(CIKM '11)*. Association for Computing Machinery, New York, NY, USA, 785–794. https://doi.org/10.1145/2063576.2063690

[63] Yanni Tang, Zhuoxing Zhang, Kaiqi Zhao, Lanting Fang, Zhenhua Li, and Wu Chen. 2025. Substructure-Aware Log Anomaly Detection. *Proc. VLDB Endow.* 18, 2 (Feb. 2025), 213–225. https://doi.org/10.14778/3705829.3705840

[64] Qwen Team. 2024. Qwen2.5: A Party of Foundation Models. https://qwenlm.github.io/blog/qwen2.5/

[65] Fei Teng, Haoyang Li, and Lei Chen. 2025. LLMLog: Advanced Log Template Generation via LLM-driven Multi-Round Annotation. *Online* (2025). https://github.com/XinTT/LLMLog

[66] The Apache Software Foundation. 2024. *SparkR: R Front End for 'Apache Spark'*. https://www.apache.orghttps://spark.apache.org R package version 3.5.1https://www.apache.org https://spark.apache.org.

[67] Xiaobin Tian, Zequn Sun, and Wei Hu. 2024. Generating Explanations to Understand and Repair Embedding-Based Entity Alignment . In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, Los Alamitos, CA, USA, 2205–2217. https://doi.org/10.1109/ICDE60146.2024.00175

[68] Tianzheng Wang, Ryan Johnson, and Ippokratis Pandis. 2017. Query fresh: log shipping on steroids. *Proc. VLDB Endow.* 11, 4 (Dec. 2017), 406–419. https://doi.org/10.1145/3186728.3164137

[69] Yuxiang Wang, Arijit Khan, Tianxing Wu, Jiahui Jin, and Haijiang Yan. 2020. Semantic Guided and Response Times Bounded Top-k Similarity Search over Knowledge Graphs . In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, Los Alamitos, CA, USA, 445–456. https://doi.org/10.1109/ICDE48307.2020.00045

[70] Yubo Wang, Hao Xin, and Lei Chen. 2024. KGLink: A Column Type Annotation Method that Combines Knowledge Graph and Pre-Trained Language Model. *2024 IEEE 40th International Conference on Data Engineering (ICDE)* (2024), 1023–1035. https://api.semanticscholar.org/CorpusID:270214355

[71] Zhenyu Wen, Jiaxu Qian, Bin Qian, Qin Yuan, Jianbin Qin, Qi Xuan, and Ye Yuan. 2024. Across Images and Graphs for Question Answering. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. 1366–1379. https://doi.org/10.1109/ICDE60146.2024.00112

[72] Anbiao Wu, Ye Yuan, Changsheng Li, Yuliang Ma, and Hao Zhang. 2024. Attributed Network Embedding in Streaming Style. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. 3138–3150. https://doi.org/10.1109/ICDE60146.2024.00243

[73] Zeyu Xiong, Daizong Liu, Xiang Fang, Xiaoye Qu, Jianfeng Dong, Jiahao Zhu, Keke Tang, and Pan Zhou. 2024. Rethinking Video Sentence Grounding From a Tracking Perspective With Memory Network and Masked Attention. *IEEE Transactions on Multimedia* 26 (2024), 11204–11218. https://doi.org/10.1109/TMM.2024.3453062

[74] Junjielong Xu, Ruichun Yang, Yintong Huo, Chengyu Zhang, and Pinjia He. 2024. DivLog: Log Parsing with Prompt Enhanced In-Context Learning. 1–12. https://doi.org/10.1145/3597503.3639155

[75] Lyu Xu, Byron Choi, Yun Peng, Jianliang Xu, and Sourav S Bhowmick. 2023. A Framework for Privacy Preserving Localized Graph Pattern Query Processing. *Proc. ACM Manag. Data* 1, 2, Article 129 (June 2023), 27 pages. https://doi.org/10.1145/3589274

[76] Chengcheng Yang, Lisi Chen, Hao Wang, Shuo Shang, Rui Mao, and Xiangliang Zhang. 2023. Dynamic Set Similarity Join: An Update Log Based Approach. *IEEE Transactions on Knowledge and Data Engineering* 35, 4 (2023), 3727–3741. https://doi.org/10.1109/TKDE.2021.3126631

[77] Muzhi Yu, Zhaoxiang Lin, Jinan Sun, Runyun Zhou, Guoqiang Jiang, Hua Huang, and Shikun Zhang. 2022. TencentCLS: the cloud log service with high query performances. *Proc. VLDB Endow.* 15, 12 (Aug. 2022), 3472–3482. https://doi.org/10.14778/3554821.3554837

[78] Chen Zhang, Sen Zhang, Chen Lei, and Peiguang Lin. 2018. Burstiness in Query Log: Web Search Analysis by Combining Global and Local Evidences. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. 1388–1391. https://doi.org/10.1109/ICDE.2018.00157

[79] Shaokun Zhang, Xiaobo Xia, Zhaoqing Wang, Ling-Hao Chen, Jiale Liu, Qingyun Wu, and Tongliang Liu. 2024. IDEAL: Influence-Driven Selective Annotations Empower In-Context Learners in Large Language Models. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=Spp2i1hKwV

[80] Tianzhu Zhang, Han Qiu, Gabriele Castellano, Myriana Rifai, Chung Shue Chen, and Fabio Pianese. 2023. System Log Parsing: A Survey. *IEEE Transactions on Knowledge and Data Engineering* 35, 8 (2023), 8596–8614. https://doi.org/10.1109/TKDE.2022.3222417

[81] Xinyi Zhang, Hong Wu, Yang Li, Zhengju Tang, Jian Tan, Feifei Li, and Bin Cui. 2023. An Efficient Transfer Learning Based Configuration Adviser for Database Tuning. *Proc. VLDB Endow.* 17, 3 (2023), 539–552. https://doi.org/10.14778/3632093.3632114

[82] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, Junjie Chen, Xiaoting He, Randolph Yao, Jian-Guang Lou, Murali Chintalapati, Furao Shen, and Dongmei Zhang. 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* (Tallinn, Estonia) *(ESEC/FSE 2019)*. Association for Computing Machinery, New York, NY, USA, 807–817. https://doi.org/10.1145/3338906.3338931

[83] Xinyang Zhao, Xuanhe Zhou, and Guoliang Li. 2024. Chat2Data: An Interactive Data Analysis System with RAG, Vector Databases and LLMs. *Proceedings of the VLDB Endowment* 17, 12 (2024), 4481–4484.

[84] Weiguo Zheng, Lei Zou, Xiang Lian, Jeffrey Xu Yu, Shaoxu Song, and Dongyan Zhao. 2015. How to Build Templates for RDF Question/Answering: An Uncertain Graph Similarity Join Approach. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (Melbourne, Victoria, Australia) *(SIGMOD '15)*. Association for Computing Machinery, New York, NY, USA, 1809–1824. https://doi.org/10.1145/2723372.2747648

[85] Xiangmin Zhou, Chengkun He, Xi Chen, and Yanchun Zhang. 2024. HSAP: A Human-in-the-Loop Social Media-Based Situation Awareness Platform. *Proc. VLDB Endow.* 17, 12 (Aug. 2024), 4493–4496. https://doi.org/10.14778/3685800.3685908

[86] Erkang Zhu, Silu Huang, and Surajit Chaudhuri. 2023. High-Performance Row Pattern Recognition Using Joins. *Proc. VLDB Endow.* 16, 5 (Jan. 2023), 1181–1195. https://doi.org/10.14778/3579075.3579090

[87] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R. Lyu. 2019. Tools and benchmarks for automated log parsing. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice* (Montreal, Quebec, Canada) *(ICSE-SEIP '19)*. IEEE Press, 121–130. https://doi.org/10.1109/ICSE-SEIP.2019.00021

[88] Zhen Zhu, Yibo Wang, Shouqing Yang, Lin Long, Runze Wu, Xiu Tang, Junbo Zhao, and Haobo Wang. 2024. CORAL: Collaborative Automatic Labeling System Based on Large Language Models. *Proc. VLDB Endow.* 17, 12 (Nov. 2024), 4401–4404. https://doi.org/10.14778/3685800.3685885