LLM meets ML: Data-efficient Anomaly Detection on Unstable Logs

FATEMEH HADADI, University of Ottawa, Canada
QINGHUA XU, Lero, University of Limerick, Ireland
DOMENICO BIANCULLI, University of Luxembourg, Luxembourg
LIONEL BRIAND, University of Ottawa, Canada and Lero, University of Limerick, Ireland

Most log-based anomaly detectors assume logs are stable, though in reality they are often unstable due to software or environmental changes. Anomaly detection on unstable logs (ULAD) is therefore a more realistic, yet under-investigated challenge. Current approaches predominantly employ machine learning (ML) models, which often require extensive labeled data for training. To mitigate data insufficiency, we propose FlexLog, a novel hybrid approach for ULAD that combines ML models — decision tree, k-nearest neighbors, and a feedforward neural network — with a Large Language Model (Mistral) through ensemble learning. FlexLog also incorporates a cache and retrieval-augmented generation (RAG) to further enhance efficiency and effectiveness. To evaluate FlexLog, we configured four datasets for ULAD, namely ADFA-U, LOGEVOL-U, SynHDFS-U, and SYNEVOL-U. FlexLog outperforms all baselines by at least 1.2 percentage points (pp) in F1 score while using much less labeled data (62.87 pp reduction). When trained on the same amount of data as the baselines, FlexLog achieves up to a 13 pp increase in F1 score on ADFA-U across varying training dataset sizes. Additionally, FlexLog maintains inference time under one second per log sequence, making it suitable for most applications, except latency-sensitive systems. Further analysis reveals the positive impact of FlexLog's key components: cache, RAG and ensemble learning.

Additional Key Words and Phrases: unstable logs, anomaly detection, data efficiency, ensemble learning, large language models

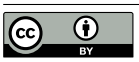
*This is the author's version of the work. The definitive Version of Record was published in **ACM Transactions on Software Engineering and Methodology (TOSEM)**, http://dx.doi.org/10.1145/3771283.*

1 INTRODUCTION

Various software-intensive systems, such as online service systems and Big Data systems, have permeated every aspect of people's daily lives. As the prevalence of such systems continues to grow, the potential impact of software failures has become increasingly significant. A critical software failure can result in service interruptions, financial losses and, in severe cases, poses threats to human safety [49].

Log-based anomaly detection has emerged as a promising approach to enhancing the dependability of software-intensive systems. An anomaly detector aims to discern anomalous patterns within system logs, which serve as vital indicators of the system's operational state. Early research predominantly employed classical machine learning techniques, such as Principal Component Analysis [90], Isolation Forest [58], and one-class SVM [34] for automated anomaly detection. However, these methods overlook the contextual information of the logs and, as a result, exhibit less effectiveness on more challenging cases [39, 51, 102].

Authors' addresses: Fatemeh Hadadi, University of Ottawa, Ottawa, Canada, fhada072@uottawa.ca; Qinghua Xu, Lero, University of Limerick, Limerick, Ireland, et.qinghua@gmail.com; Domenico Bianculli, University of Luxembourg, Luxembourg, Luxembourg, domenico.bianculli@uni.lu; Lionel Briand, University of Ottawa, Ottawa, Canada and Lero, University of Limerick, Limerick, Ireland, lbriand@uottawa.ca.



In recent years, deep learning-based (DL) methods have gained significant traction in anomaly detection. Unlike classical machine learning methods, DL methods typically consist of a large number of trainable parameters, enabling them to model long contextual dependencies and complex semantic patterns in logs. In particular, log-based anomaly detection has significantly benefited from sequential DL models such as LSTM and transformers, achieving high predictive performance on multiple benchmark datasets [21, 25, 65, 85]. Despite the success of DL-based methods, we highlight three key challenges prevalent in current practices of log-based anomaly detection:

C1 Existing approaches often assume a stable data distribution, which is unrealistic in real-world scenarios. In practice, unlike current benchmark datasets, where log structures and contents remain stable, logs can be *unstable* due to software evolution or environment changes. The majority of anomaly detection methods [21, 25, 51, 57, 62, 65, 91, 93] have been proposed for and evaluated on stable log datasets. In contrast, only a few studies [39, 56, 102] have investigated anomaly detection on unstable logs, mainly due to a lack of public benchmarks. Earlier works leverage private or synthetic data. However, more recently, Huo et al. [39] have proposed two public datasets for unstable logs.

C2 Machine learning-based (ML) anomaly detection, especially when based on DL methods, often relies on substantial labeled data, which is costly to obtain. The most effective methods in anomaly detection—particularly those based on DL—often rely on an extensive amount of labeled data for their training, due to their substantial number of trainable parameters. Collecting such data requires intensive labor and significant domain knowledge in practice. More recently, the study of Yu et al. [96] has demonstrated that simpler methods, such as Decision Trees (DT), exhibit greater effectiveness. However, they only evaluated their methods on stable logs.

C3 The reported effectiveness of ML-based anomaly detection might be inflated due to data leakage issues. Yu et al. [97] found such issues in several benchmark datasets such as HDFS [89] and BGL [70], where the testing data contains training instances. This leakage can potentially boost the effectiveness of ML methods, especially DL methods, as their large parameter set allows them to memorize the training data. To address this issue, Yu et al. [96] removed instances in testing data that were already seen in the training data, which led to a significant drop in anomaly detection effectiveness.

In light of these challenges, we identify the next frontier of log-based anomaly detection as addressing a more realistic and demanding task: *anomaly detection on unstable logs with limited labeled data (ULAD)*. Unlike *anomaly detection on stable logs (SLAD)*, ULAD reflects the practical reality where logs evolve due to software updates or environmental factors, resulting in instability (C1). This evolution results in changes such as the addition, removal, or modification of log messages, as well as shifts in their order. Furthermore, real-world constraints often limit the availability of labeled data, which is costly to collect and requires domain knowledge (C2). To ensure a realistic assessment of ULAD solutions, test instances already seen in the training data should be removed from testing data, addressing issues of data leakage (C3).

The literature on anomaly detection in the presence of unseen log templates [25, 65, 93] is related to the ULAD challenge. However, ULAD is more challenging since, although unstable logs are test instances not present in training data, they further follow a different log distribution than historical data. Moreover, these works do not address the C2 and C3 challenges.

A promising approach to tackling these challenges of ULAD lies in leveraging large language models (LLMs). Recently, LLMs have gained significant attention for their ability to mitigate the data insufficiency problem faced by ML-based methods. By pretraining on vast, diverse datasets, LLMs can excel in tasks with limited labeled data. Several researchers have investigated various prompts to instruct pretrained LLMs such as GPT to perform anomaly detection directly (i.e., in-context

learning) [29, 59, 101]. An alternative to in-context learning is fine-tuning, where extra training on domain-specific data is required. While in-context learning has been more widely studied because it does not require additional training and can be applied directly with prompts, Mosbach et al. [69] highlighted its poor generalizability on challenging tasks. Drawing from this observation, fine-tuning may be a more suitable strategy for ULAD when using LLMs.

Most recent works [29, 43, 52, 87, 101, 104] in log analysis focused on using closed-source LLMs from OpenAI, due to their user-friendly environment and effective performance. However, these LLMs induce a significant financial cost and show unpredictable latency during training and inference [41]. On the other hand, open-source LLMs are free to use, and we have some degree of control in terms of fine-tuning algorithms and inference time.

Though a fine-tuned LLM can address challenges C1, C2, and C3 faced by ML methods, they are inherently designed for textual understanding and generation rather than the detection of anomalous patterns in logs. Conversely, existing anomaly detectors using ML models such as Decision Tree (DT) and Single-layer Feedforward Network (SLFN) have proven to be effective in the SLAD task when abundant data is available for training [97], demonstrating their capacity to detect anomalous patterns. This indicates that combining ML methods with an LLM would leverage the strengths of both approaches, ML models' ability to detect anomalous patterns and fine-tuned LLM's capacity to handle scarce labeled data effectively.

To this end, we propose *FlexLog*, a novel approach that requires significantly less labeled data for ULAD compared to ML methods. FlexLog integrates ML-based anomaly detectors and an LLM, combining their strengths to enhance effectiveness and data efficiency. We summarize our contributions as follows.

- **Dataset configuration for ULAD.** Most existing benchmark datasets contain stable logs, used for the SLAD task. In this paper, we selected three of these datasets—HDFS, LOGEVOL, and ADFA-LD (referred to as ADFA for brevity hereafter)—and configured four unstable datasets for ULAD, namely SynHDFS-U, LOGEVOL-U, SYNEVOL-U, and ADFA-U, by deliberately introducing disparities between the training and testing datasets. To eliminate the influence of data leakage (C3) [97] and further increase instability, we performed de-duplication on each dataset, ensuring that any data samples included in the testing datasets were excluded from the training datasets.
- **FlexLog, a novel approach for ULAD equipped with practical strategies to boost effectiveness and efficiency.** FlexLog uses average-based ensemble learning to combine the predictive strengths of a fine-tuned LLM and ML methods, capturing intricate anomalous patterns with only limited labeled data for training. Specifically, to address C2, FlexLog employs parameter-efficient fine-tuning (PEFT) of a pretrained LLM, leveraging its vast embedded knowledge to mitigate the constraints of scarce labeled data. To tackle C1, FlexLog integrates retrieval-augmented generation (RAG) to dynamically incorporate external knowledge and enhance the model's adaptability to unstable log distributions. Additionally, a cache mechanism improves computational efficiency by eliminating redundant operations.
- **State-of-the-art effectiveness and data efficiency for ULAD.** To evaluate FlexLog, we first compare it against baselines trained on full datasets, even though FlexLog itself is trained on significantly smaller datasets. This comparison is conducted on two real-world datasets (ADFA-U and LOGEVOL-U) and two synthesized datasets (SynHDFS-U and SYNEVOL-U). Experimental results show that FlexLog achieves state-of-the-art effectiveness, outperforming the top baseline by at least 1.2 percentage points (pp) in terms of F1 score, while reducing the usage of labeled data by more than 62.87 pp. Further, we assess

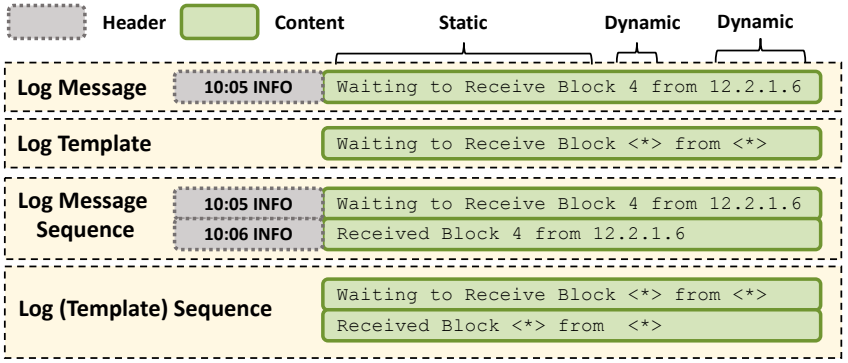


Fig. 1. Examples of log Message, Log Message Sequence, Log Template, and Log Template Sequence.

the data efficiency of FlexLog by comparing it with baselines when trained on the same datasets. Experiment results on ADFA-U show that FlexLog consistently outperforms all baselines across varying training dataset sizes, except the extreme data scarcity scenario (the training dataset size is 50), where all methods exhibit poor performance due to insufficient labeled data. FlexLog achieves a maximum gain of 13 pp in F1 score when the training dataset size is 500. This confirms FlexLog is the most effective choice when only limited labeled data is available.

The rest of the paper is organized as follows. Section 2 presents the basic definitions and concepts that will be used throughout the paper. Section 3 describes our data-efficient anomaly detection approach, FlexLog. Section 4 presents our experimental design. Section 5 outlines our results, discusses the implications, and describes the threats to the validity of our study. Section 6 presents related works and finally, Section 7 concludes and suggests future directions for research and improvement.

2 BACKGROUND

2.1 Logs

Logs are semi-structured or unstructured text generated by logging statements (e.g., *"logging.info ('Received Block %d from %s',id, ip)"*) in source code [31]. The main concepts related to logs are *log message*, *log message sequence*, *log template*, and *log template sequence*, which we explain below and exemplify in Figure 1.

A *log message* contains two main components: the *header* (e.g., timestamp or log level), and the *content*, depicted as grey-dotted and green boxes in Figure 1, respectively. The content of a log message can be further divided into static and dynamic parts. The static parts refer to the fixed text written by developers in the logging statement, e.g.,"`Waiting to Receive Block from`" and "`Received Block from`"; the dynamic parts are expressions evaluated at runtime, such as the actual block id "`4`" and IP address "`12.2.1.6`".

A *log template*, also called *event template* or *log key*, is usually obtained through log parsing [100], which masks the dynamic parts of the log message content with a special symbol, such as "<*>". Compared to log messages, log templates eliminate the influence of specific values in the dynamic parts, enabling downstream tasks (e.g., anomaly detection [50], log summarization [60]) to focus on analyzing patterns within logs, without being confused by variations in concrete values.

A *log message sequence* is a log fragment consisting of multiple log messages, which typically records the execution flow of a specific job or process. A log message sequence consists of log messages that either pertain to a specific task (i.e., session-based partitioning) or are grouped within

a fixed-size window (i.e., sliding/fixed window partitioning). Session-based partitioning groups log messages based on their session IDs, thereby creating log sequences that encapsulate activities within sessions. On the other hand, sliding/fixed window partitioning employs a fixed-size window to group log messages, generating log sequences that capture a snapshot of system activity over time.

Parsing a log message sequence yields a *log template sequence* (as shown at the bottom of Figure 1). We remark that "log template sequence" is often referred to simply as "log sequence" in the literature; therefore, for brevity, we adopt the term *log sequence* throughout this paper.

2.2 Anomaly Detection on Logs

Anomalies in logs refer to logs that do not conform to the normal behavior of a system [31]. Log-based anomaly detection represents a binary classification task to identify anomalies from logs. Depending on their distributions, logs can be divided into two categories: *stable logs* (Definition 2.1) and *unstable logs* (Definition 2.2).

Definition 2.1 (Stable Logs)**.** Logs drawn from a single underlying distribution, i.e., their structure and semantics remain consistent in all the logs.

Definition 2.2 (Unstable Logs)**.** Logs drawn from more than one underlying distribution.

Stable logs are typically generated from systems whose logging behaviors and operating environment remain unchanged over time, resulting in consistent structure and semantics of logs. In contrast, unstable logs originate from multiple distributions caused by system or environmental changes. *System evolution* refers to internal changes within a software system, such as version upgrades. Developers often modify source code, including logging statements, resulting in changes to logs. As Kabinna et al. [45] reported, around 24 %–40 % of log statements change during their lifetime. Taking the public dataset LOGEVOL as an example, 24 % of logging statements were modified during the system upgrade from Spark version 2 to 3 [39]. As a result, 14 % of logs collected in Spark 3 contain new log templates induced by system evolution. This figure represents a conservative estimate of the percentage of unstable logs, as other causes of instability (e.g., reordering of logging statements during execution) are not accounted for due to the lack of a mapping between execution paths and their log distributions. Nonetheless, these results clearly highlight that unstable logs are common in practice, underscoring the importance of handling instability caused by system evolution. *Environmental evolution*, on the other hand, represents the changes of external factors, such as a shift of user distribution and the emergence of unseen attack types. These changes affect both normal and abnormal patterns in the logs by altering the structure, content, or frequency of log messages. For example, shifts in user distributions—such as changes in user geographic regions—may introduce new log sequences or alter the frequency of existing ones due to differences in usage patterns, device configurations, or regional preferences. Similarly, novel or previously unseen attacks may generate anomalous logs with sequences or templates that have not been observed in the earlier log distribution.

Built on the definitions of stable and unstable logs, we define two corresponding anomaly detection tasks, namely *Anomaly Detection on Stable Logs (SLAD)* and *Anomaly Detection on Unstable Logs (ULAD)* as defined in in Definition 2.3 and 2.4, respectively.

Definition 2.3 (Anomaly Detection on Stable Logs)**.** SLAD is a binary classification task that aims to predict anomalies in stable logs, i.e., the training data and testing data follow the same distribution.

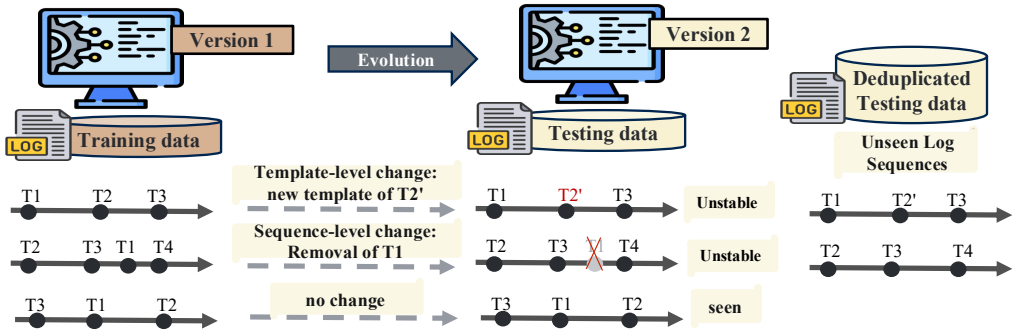


Fig. 2. Examples of Unstable Logs Resulting from Log Evolution.

Definition 2.4 (Anomaly Detection on Unstable Logs)**.** ULAD is a binary classification task that aims to predict anomalies in unstable logs, i.e., the training data and testing data are drawn from different distributions.

SLAD is the predominant configuration in the literature [53, 102], largely because existing benchmark datasets often assume a stable software system. In this configuration, the training dataset $D^{train} = \{ls_1, ls_2, ..., ls_n\}$ and the testing dataset $D_S^{test} = \{ls_{n+1}, ls_{n+2}, ..., ls_{n+m}\}$ are sampled from the same distribution. Despite its wide adoption, SLAD does not reflect challenges faced by real-world applications, e.g., evolving systems or operating environments. In contrast, ULAD (Definition 2.4) considers a more challenging yet realistic scenario. Specifically, the training dataset D^{train} consists of stable logs collected under consistent conditions, while the test dataset D_U^{test} contains unstable logs resulting from system or environmental evolution.

Evaluation on De-duplicated Datasets. As demonstrated in previous work [97], data leakage is prevalent in existing benchmark datasets, artificially inflating the effectiveness of anomaly detectors. Data leakage entails an overlap between testing and training data, i.e., some log sequences in the testing dataset have already been seen in the training dataset. This phenomenon affects both the SLAD and ULAD tasks. To eliminate the risk of data leakage, we remove *seen* testing log sequences that are already present in the training dataset D^{train}, yielding a new testing dataset for ULAD and SLAD, denoted as $D_{U\dagger}^{test}$ and $D_{S\dagger}^{test}$ respectively, defined in Equations 1 and 2.

$$D_{U\dagger}^{test} = D_U^{test} \setminus D^{train} \tag{1}$$

$$D_{S\dagger}^{test} = D_S^{test} \setminus D^{train} \tag{2}$$

$D_{U\dagger}^{test}$ and $D_{S\dagger}^{test}$ consist of only *unseen log sequences*. These sequences differ from the ones in the training dataset at two possible levels: 1) template level (when there is an *unseen log template* in the sequence), 2) sequence level (when all the log templates are already mentioned in the training data but their order is new). Unseen log sequences can be either stable or unstable, depending on their underlying log distributions. As mentioned by Yu et al. [96], after de-duplication, the effectiveness of anomaly detection models on testing data drops, making it a more challenging task in this realistic scenario.

Illustrative Examples. Figure 2 provides an overview of the de-duplicated ULAD with example log sequences. After system evolution from version 1 to version 2, the first two sequences at the

top undergo changes at different levels. In the first sequence, $T1 \rightarrow T2 \rightarrow T3$, template $T2$ is updated to a new template $T2'$, representing a change at the template level. The second sequence, $T2 \rightarrow T3 \rightarrow T1 \rightarrow T4$, experiences a change at the sequence level, where template $T1$ is no longer present. The last sequence shown, $T3 \rightarrow T1 \rightarrow T2$, is regenerated in the later version without any change, and is therefore marked as a "seen" sequence relative to the training data. During de-duplication, the seen sequence is removed from the test set. The remaining sequences are referred to as unseen log sequences, as they contain no overlapping sequences.

2.3 Task Adaptation Strategies for Large Language Models

LLMs typically consist of substantial parameters pretrained on vast and diverse datasets, possessing knowledge across various domains. However, how to effectively adapt pretrained LLMs to domain-specific tasks remains an open problem. Two predominant strategies for task adaptation are in-context learning (ICL) and fine-tuning (FT).

ICL operates without altering the weights of the LLMs [3]. Instead, it leverages prompts—structured textual inputs—to guide the model's behavior. These prompts typically include task instructions and, in some cases, a series of demonstrations in a conversation between the user and the assistant. In a classification task, e.g., ULAD, each demonstration consists of an input x paired with its corresponding ground-truth label y. When no demonstrations are provided, the approach is referred to as zero-shot ICL, whereas the inclusion of a few demonstrations constitutes a few-shot ICL.

Although ICL is relatively easy to implement, it faces several challenges and limitations, including issues with efficiency, scalability, generalizability, and high financial cost when using closed-source LLMs [20]. As an alternative, FT alleviates these issues by training pre-trained LLMs with domain-specific data. In practice, there are two main types of fine-tuning, namely API -based FT and Custom FT.

API-based FT refers to fine-tuning performed through dedicated APIs made available by the LLM provider, e.g., OpenAI [71]. This is typically the case for closed-source LLMs,

such as GPT-3.5 [73] and GPT-4 [61], for which neither full nor selective fine-tuning is allowed without accessing their APIs. These APIs support fine-tuning a set of prompt-completion pairs or conversations depending on whether LLMs are used in purely generative or conversational settings.

Custom FT, on the other hand, is applicable to open-source LLMs, such as LLama [79] and Mistral [42]. Common custom FT techniques include full fine-tuning and parameter-efficient fine-tuning (PEFT). Let the trainable parameter set of an LLM be denoted as W, the task-specific dataset as D, and its associated label set as L.

- *Full Fine-Tuning*: This approach utilizes gradient descent-based optimizer to update W to W^f, thereby adapting the LLM to a specific task. Specifically, prompts are constructed using D and fed into an LLM. The model's output distribution $\hat{y}$ is then compared with the corresponding label distribution y, using a distribution-level loss, e.g., cross-entropy loss. This loss guides weight updates from W to W^f through backpropagation.
- *Parameter-Efficient Fine-Tuning:* PEFT preserves original LLM weights while training only a small number of task-specific adapter layers and parameters. There are several types of PEFT methods, including additive, selective, reparameterized, and hybrid PEFT [28]. The predominant PEFT techniques are LoRa [36] and its derivative techniques such as QLoRa [17]. Essentially, LoRa uses a low-rank decomposition to reduce computational cost while maintaining performance similar to full fine-tuning as in Equation 3.

$$\begin{aligned} W^f &= W + \Delta W \\ &= W + AB \end{aligned} \tag{3}$$

where $A \in \mathbb{R}$ and $B \in \mathbb{R}$ are lower rank matrices compared to W, with dramatically fewer trainable parameters. Such techniques significantly reduce the computational cost while maintaining comparable performance to fully fine-tuned LLMs.

3 METHODOLOGY

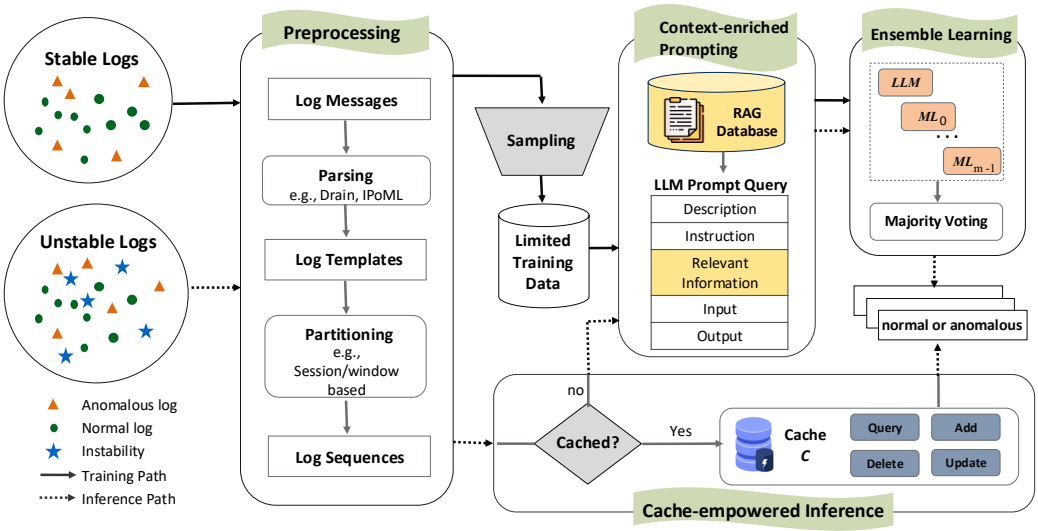


Fig. 3. Architecture of FlexLog.

In this paper, we propose a novel approach, namely FlexLog, to tackle ULAD by synergizing the capabilities of ML methods and LLMs via ensemble learning. Specifically, FlexLog combines the predictions from trained ML methods and a fine-tuned LLM to make a final decision. This approach leverages the strengths of both paradigms: ML methods excel at capturing anomalous patterns within logs, while LLM brings broad prior knowledge from pretraining, allowing them to adapt to novel log patterns even with limited labeled data. Also, we tackle the three key challenges in ULAD— unstable log distribution (C1), data insufficiency (C2), and data leakage (C3) — by employing ensemble learning of ML and LLM, PEFT, and de-duplication in the testing data, respectively. Additionally, we further tackle unstable log distributions (C1) by using RAG in prompting when relevant external information is available regarding log sequences.

Figure 3 illustrates the architecture of FlexLog, which comprises four main components: preprocessing (§ 3.1), cache-empowered inference (§ 3.2), context-enriched prompting (§ 3.3), and ensemble learning (§ 3.4). FlexLog is designed to predict whether unstable logs—generated by software systems that have undergone software or environmental evolution—are anomalous. Specifically, the *preprocessing component* converts raw stable and unstable logs into log sequences by extracting log templates and grouping them into log sequences using either window-based or session-based partitioning. For illustrative purposes, consider a log sequence ls_i as an example. The *cache-empowered inference component* first checks if ls_i matches an existing entry in the cache. If a match is found, FlexLog retrieves the stored prediction as the output label directly. Otherwise, the *context-enriched prompting component* uses ls_i in a structured prompt enriched with contextual information, such as log event descriptions and Linux system call names. This prompt includes key fields such as description, instructions, relevant information (optional), input, and output. It serves as input both for fine-tuning of and for inference with an LLM, which acts as one of FlexLog's

base models. To construct the ensemble, the *ensemble learning component* fine-tunes the LLM-based model (e.g., Mistral or Llama) and trains the ML-based models (e.g., DT and KNN [96]) on a limited dataset sampled from the preprocessed stable logs to maintain data efficiency and reduce training overhead. During inference, if no matching log sequence is found in the cache, binary anomaly predictions from these base models are aggregated using majority voting to produce the final decision. This prediction is then stored in the cache to optimize future queries. In the following sections, we provide a detailed explanation of each component.

3.1 Preprocessing

As illustrated in the first box of Figure 3, the *preprocessing component* transforms raw log messages to log sequences via two primary processes, namely parsing and partitioning. Given a raw log message (e.g., "`12:03 INFO Sent Block 12`"), we leverage a log parser (e.g., Drain [30]) to identify the static parts (e.g., "`Sent Block`") and dynamic parts (e.g., "`12`") and replace the latter with the symbol "`<*>`".

The parsing process identifies unique log templates and assigns the same identifier to all their occurrences, enabling the *cache-empowered inference* component to track processed log sequences and the *context-enriched prompting* component to incorporate relevant template information during prompt construction. The partitioning process aggregates log templates into log sequences based on their session IDs or a fixed-size window (as described in § 2.1).

3.2 Cache-empowered Inference

FlexLog maintains a cache C as illustrated at the bottom of Figure 3. C stores previously seen log sequences along with their predicted labels. Given a log sequence ls_i under detection, FlexLog first queries cache C for a matching entry. If an identical log sequence is found, the corresponding label l_i is retrieved and used directly, bypassing the need for additional computation by RAG and ensemble learning. Conversely, if no match is found in C, FlexLog performs context-enriched prompting (§ 3.3) and leverages ensemble learning (§ 3.4) to predict the label for ls_i and subsequently adds the new log sequence and its label to the cache. To reduce storage overhead and remain independent of template length, each log sequence is represented as an ordered sequence of log template IDs rather than raw log messages.

The maintenance of C involves four core functions, namely *query*, *add*, *update*, and *delete*. The *query* function compares the input log sequence against the entries in C and returns the identical entry along with its associated label if a match is found. The *add* function inserts a new log sequence and its predicted label into C if it is not present. The *delete* function removes a specific entry from C, allowing FlexLog to manage cache size or discard outdated information. The *update* function modifies an existing entry's label, incorporating human corrections to improve future predictions. By leveraging the delete and update functions, the cache mechanism enables flexibility to adjust to memory constraints and future updates of the sequences' labels, respectively.

3.3 Context-enriched Prompting

This component processes the log sequences that are not stored in the cache, preparing them for the LLMs used in FlexLog. Log sequences are inherently challenging for LLMs to understand because these models are primarily designed for natural language processing (NLP) tasks and are better suited to processing and reasoning over textual data. To bridge this gap, we place log sequences with their log templates into semi-structured prompts that resemble natural language, enabling LLMs to leverage their NLP capabilities effectively. The RAG component inside the prompt aims to integrate relevant external information with log sequences acquired from preprocessing, formulating semi-structured prompts.

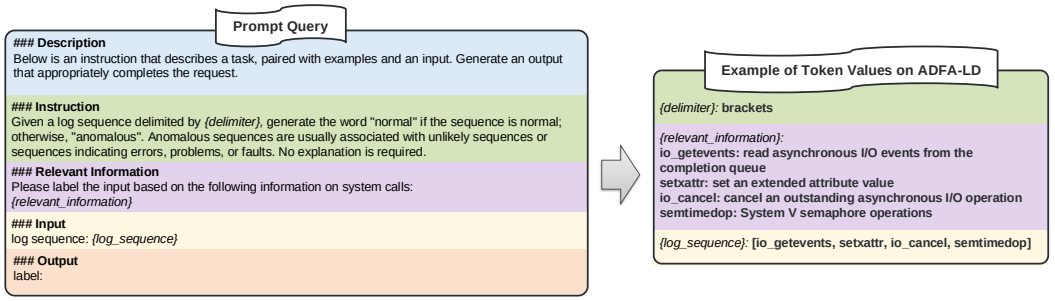


Fig. 4. FlexLog's Prompt Design for LLM Fine-tuning and Inference.

To devise an effective prompt structure for ULAD, we adopt the tactics reported by Winteringham [86]. Concretely, as illustrated in Figure 4, each prompt comprises five parts, namely, description, instruction, relevant information, input, and output. Notably, the retrieved context is included only when pertinent information is available. We provide the details of each part next.

Description. This part sets the overall context of the task for the LLM. It provides a high-level overview of what the model is expected to accomplish. For instance, as shown in Figure 4, the description explains that the LLM should generate an output that appropriately completes the task request. While this part does not specify the input or output format explicitly, it prepares the LLM by providing a concise summary of the task objective.

Instruction. The instruction part formally introduces the task by describing its goal, input format, and output expectations. It specifies how the LLM should process the input and produce the desired label. After experimenting with multiple instruction formats, we present the most effective formulation in Figure 4. In this part, we describe the delimiter for log sequences (i.e., "brackets") and add "No explanation is required", instructing the LLM to output only the label.

Relevant Information (Optional). This part includes additional information related to the log sequence, which enhances the LLM's ability to interpret the input. When available, contextual data is retrieved and presented in this part to provide background information. This information can include descriptions of specific log templates, system calls, or other contextual elements relevant to the log sequence under analysis. For example, the right-hand side of Figure 4 includes system call descriptions, such as `setxattr` or `semtimedop`, which help clarify the function and purpose of the operations within the log sequence. Including such information allows the LLM to better understand relationships between the components of the log sequence, improving its ability to generate accurate predictions. However, if no relevant external context is available, this part of the prompt is omitted.

Input. This part presents the log sequence to be analyzed in the format *"log sequence: {log_sequence}"*, where *{log_sequence}* is a placeholder dynamically replaced with different log sequences during fine-tuning and inference. For example, the right-hand side of Figure 4 shows the replacement of the placeholder with the log sequence *"[io_getevents, setxattr, io_cancel, semtimedop]"* from the ADFA-U dataset.

Output. The output section guides an LLM in predicting the label of the input log sequence. It provides a formatted prompt that ends with "label:", prompting the LLM to generate the next token as either "normal" or "anomalous", based on its analysis of the log sequence.

3.4 Ensemble Learning

The goal of this component is to maximize the utility of limited labeled data by integrating different base models, each offering a unique perspective on performing ULAD effectively. To train the base

models, we leverage the stable logs collected from software systems before undergoing the software or environment evolution. A salient feature of FlexLog is employing both ML- and LLM-based models as base models as introduced in § 1. Due to LLMs' pretraining on diverse corpora, they can be effectively fine-tuned with only limited data. Consequently, we sample only a subset from the stable logs to create the training dataset. Using this training dataset, FlexLog fine-tunes an *LLM* and fits m ML models $\{ML_0, \ldots, ML_{m-1}\}$.

For a given LLM *LLM*, FlexLog adopts (see Equation 4) API-based fine-tuning for closed-source LLMs (e.g., GPT 4o) and LoRa for open-source LLMs (e.g., Llama and Mistral); details about API-based fine-tuning and LoRa are provided in Section 2. We denote the fine-tuned LLM as LLM^f.

$$LLM^f = \begin{cases} API\text{-}based\_FT(LLM, data = S_{train}, label = L) & LLM \in \text{Closed-source LLMs} \\ LoRa(LLM, data = S_{train}, label = L) & LLM \in \text{Open Source LLMs} \end{cases} \tag{4}$$

For a given ML model *ML*, gradient descent optimization is applied for neural network-based models, while model-specific fitting methods are used for non-parametric models such as DT (Equation 5). Note that K-Nearest Neighbors (KNN) does not involve a traditional training or fitting process but instead relies on distance-based comparison during inference. The resulting learned ML model is denoted as ML^f.

$$ML^f = \begin{cases} gradient\_descent(ML, data = S_{train}, label = L) & ML \in \text{Neural Networks} \\ fit\_dt(ML, data = S_{train}, label = L) & ML = \text{DT} \\ distance\_based\_comparison & ML = \text{KNN} \end{cases} \tag{5}$$

After training individual models, the next step is to combine their outputs using *majority voting*, a commonly used, simple yet effective ensemble learning technique in the literature [75, 103, 106]. Formally, let $\mathcal{M} = \{M_0, M_2, \ldots, M_{N-1}\}$ be the set of N learned base models, with $\mathcal{M}_i \in \mathcal{M}$ representing either a learned LLM LLM^f or an ML-based method ML^f. For a given log sequence ls_i, each base model $\mathcal{M}_i$ predicts the label y_i of x, equals 1 if anomalous, and 0 if normal. The final label is determined by a majority voting function $MV(\cdot)$ among all base models, as shown in Equation 6.

$$\phi_i = MV(ls_i) = \begin{cases} 1, & \text{if } \sum_{i=0}^{N-1} y_i > \frac{N}{2} \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

Here, ϕ_i represents the final prediction for ls_i. In case of a tie (when exactly half of the votes are normal), the sequence is classified as *normal*, following our assumption that anomalies are rare.

4 EXPERIMENTAL DESIGN

4.1 Research Questions

We investigate the following research questions:

RQ1 (**effectiveness**) How effective is FlexLog for ULAD compared to the baselines?
- **RQ1.1** Can FlexLog trained on limited labeled data achieve comparable effectiveness to baselines trained on full datasets?
- **RQ1.2** What impact does the level of log instability have on FlexLog and the baselines?

RQ2 (**data efficiency**) How does the amount of labeled training data impact FlexLog's effectiveness, and can it maintain robust effectiveness under varying degrees of data scarcity?

RQ3 (**time and memory efficiency**) What is the performance of FlexLog in terms of time efficiency during training and inference, and how much memory overhead does the cache incur during inference?

RQ4 (**configuration impact**) How does the performance of FlexLog vary under different configurations, including ablations of base models, RAG, and the cache, as well as alternative LLM choices?

RQ1 investigates the overall effectiveness of FlexLog for the ULAD, comprising two sub-RQs. With RQ1.1, we aim to demonstrate the strengths of FlexLog when only limited labeled data is available. Specifically, we compare baselines trained with full datasets against FlexLog trained with much smaller subsets. With RQ1.2, we aim to highlight the distinct advantage of FlexLog in handling gradually increasing instabilities in ULAD. To this end, we assess the effectiveness of FlexLog and baselines on SLAD and under the influence of varying ratios of instability in both log templates and sequence levels. RQ2 focuses on data efficiency by training FlexLog and baselines on progressively larger subsets of ADFA-U (e.g., containing 50, 500, 1000, 1500, and 2000 training samples). Due to computational constraints, we cannot perform data efficiency analysis on all datasets. Hence, we prioritize our most challenging dataset ADFA-U for this analysis. RQ3 investigates the time efficiency of FlexLog during training and inference. While employing LLM-based approaches (e.g., FlexLog) may enhance effectiveness, it often comes at the cost of increased training and inference time. This question aims to evaluate the trade-offs between effectiveness and time efficiency, providing practical insights for those considering the use of LLMs in similar tasks. Additionally, RQ3 examines the memory efficiency of FlexLog 's cache mechanism during inference, demonstrating its scalability in resource-constrained environments. Lastly, RQ4 involves exploring the impact of different configurations of FlexLog. Specifically, we assess how the exclusion of the cache C, the exclusion of RAG in context-enriched prompting, and the choices of base models (e.g., removing some base models from the ensemble or replacing Mistral with other LLMs), affect the overall effectiveness or efficiency of FlexLog.

4.2 Experiment Setup

Table 1. Overview of Datasets

Name	Sys	#Log Messages	#Anomalous Messages	#Sessions	#Log Templates	Session Length		
						avg	min	max
ADFA	Linux	2,747,550	317,388 (11.5%)	5,951	175	461.69	75	4,474
LOGEVOL	Hadoop 2	2,120,739	35,072 (1.6%)	333,699	319	6.35	1	1,963
	Hadoop 3	2,050,488	30,309 (1.4%)	343,013	313	5.97	1	1,818
	Spark 2	931,960	1,702 (0.1%)	13,892	130	67.08	1	1125
	Spark 3	1,600,273	2,430 (0.1%)	21,232	134	75.37	1	1977
HDFS	Hadoop	11,110,850	284,818 (2.9%)	575,061	48	19.32	2	30

4.2.1 Datasets. We configured four datasets for ULAD from three public datasets, namely ADFA [13], LOGEVOL [39], and HDFS [89]. We exclude other popular benchmarks (e.g., BGL [70], Thunderbird [70], and Spirit [70]) because they contain only stable logs, and manually injecting instability is not feasible due to the lack of information about their annotation strategies.

Table 1 presents relevant statistics for these three datasets; column "Sys" indicates the system from which the logs were collected. "#Log Messages", "#Anomalous Messages", "#Sessions", and "#Log Templates" indicate the number of log messages, anomalous log messages, sessions, and unique log templates in each dataset, respectively. Column "Session Length" indicates the average, minimum, and maximum number of log messages in each session. We elaborate on each dataset next.

ADFA. Creech and Hu [13] created the Australian Defense Force Academy Linux Dataset by collecting Linux server operation logs and applying contemporary web attacks. ADFA comprises 2,747,550 log messages, i.e., Linux system calls in this context, of which 317,388 are anomalous (11.5 %). The attacks applied to the system include the exploitation of a TIKI WIKI vulnerability using a Java-based Meterpreter ("java"), password brute-forcing with the Hydra tool ("hydra"), deploying a Linux Meterpreter payload via a poisoned executable ("meter"), leveraging a remote file inclusion vulnerability to deploy a C100 webshell ("web") and creating privilege escalation by adding a superuser account with a poisoned executable ("adduser").

LOGEVOL. Huo et al. [39] introduced the LOGEVOL dataset captured from the real-world operations of Hadoop 2, Hadoop 3, Spark 2 and Spark 3 systems[1]. All datasets were generated using HiBench [40] during the operation of 22 cloud computing tasks, such as sorting and classification [33, 57]. To capture real-world anomaly scenarios into logs, they injected 18 fault types into the system, including network fault, process suspension, process killing, and resource occupation. The Hadoop 2 dataset consists of 2,120,739 log messages (including 1.6 % anomalous) while the Hadoop 3 dataset is made up by 2,050,488 log messages (including 1.4 % anomalous). Notably, 104 out of 303 (33.22 %) log templates from the Hadoop 3 dataset are novel and absent from the Hadoop 2 dataset, reflecting its instability and log template evolution. The Spark 2 dataset involves 931,960 log messages, and the Spark 3 dataset is made up of 1,600,273 log messages. Compared to the Hadoop 2 and Hadoop 3 datasets, the proportion of anomalous logs in the Spark 2 and Spark 3 datasets is significantly lower, with both datasets having an anomaly rate of only 0.1 %.

HDFS. Hadoop Distributed File System (HDFS) logs [89] were produced by running MapReduce jobs on Amazon EC2 nodes, consisting of 11,197,954 log messages, of which 284,818 (2.9 %) are anomalous. The average number of log messages in a sequence is 19.32. The total number of unique log templates is 48. This dataset includes 11 types of anomalies, such as the deletion of a block that no longer exists or receiving a block that does not belong to any file. For a comprehensive description of the anomalies, we refer readers to the original paper [89].

4.2.2 ULAD and SLAD Configuration. Our experiments involve the evaluation of FlexLog on both the ULAD and SLAD, with ULAD being the primary focus of this paper and SLAD serving as a baseline in RQ1 and RQ2. As mentioned in Section 2, ULAD is characterized by the disparity between the training and testing datasets, whereas SLAD involves training and testing data drawn from the same distribution. Each dataset described in § 4.2.1 is configured to be used for both SLAD and ULAD, consisting of a training dataset $\mathcal{D}^{train}$ and a testing dataset $\mathcal{D}^{test}$. Additionally, a small, curated subset $\tilde{\mathcal{D}}^{train}$ is sampled from each full training dataset for the training of FlexLog. We provide details of the SLAD and ULAD configurations on each dataset next, followed by configurations of $\tilde{\mathcal{D}}^{train}$ for FlexLog.

ULAD Configuration. As discussed in Section 2.2, unstable logs result from system or environmental evolution. To simulate these scenarios, we configured the unstable LOGEVOL dataset LOGEVOL-U for system evolution and the unstable ADFA dataset ADFA-U for environmental evolution. To further investigate the influence of different levels of instability, we include two synthesized datasets in our experiments, namely SynHDFS-U and SYNEVOL-U. These datasets are created by injecting different levels of instability into the HDFS and LOGEVOL datasets, respectively. Table 2 summarizes the ULAD configuration for each dataset and presents their statistics, including the full training dataset size ($\mathcal{D}_{\#}^{train}$), FlexLog's training dataset size ($\tilde{\mathcal{D}}_{\#}^{train}$), and the testing dataset size ($\mathcal{D}_{\#}^{test}$). We also report the duplication ratio, which quantifies data leakage, i.e., log

[1]Hadoop versions 2.10.2 and 3.3.3 are referred to as Hadoop 2 and 3, respectively, and Spark versions 2.4.0 and 3.0.3 are denoted as Spark 2 and Spark 3, respectively.

sequences appearing in both training and testing datasets. As discussed in § 1, data leakage allows anomaly detectors to memorize the training data, resulting in artificially inflated effectiveness on the testing data. Hence, we addressed the data leakage issues identified by Yu et al. [97] through de-duplication in each configuration. Specifically, we removed log sequences from the testing dataset if they were already included in the training dataset.

Table 2. ULAD Configurations

Dataset	Configuration		Duplication Ratio	#Log Sequences		
	train	test		$\mathcal{D}_{\#}^{train}$	$\tilde{\mathcal{D}}_{\#}^{train}$	$\mathcal{D}_{\#}^{test}$
ADFA-U	$ADFA_{w/o\ java}$	$ADFA_{w/\ java}$	0.32	4786	1000	1165
	$ADFA_{w/o\ hydraSSH}$	$ADFA_{w/\ hydraSSH}$	0.31	4734	1000	1217
	$ADFA_{w/o\ hydraFTP}$	$ADFA_{w/\ hydraFTP}$	0.31	4748	1000	1203
	$ADFA_{w/o\ meter}$	$ADFA_{w/\ meter}$	0.34	4835	1000	1116
	$ADFA_{w/o\ web}$	$ADFA_{w/\ web}$	0.33	4792	1000	1159
	$ADFA_{w/o\ adduser}$	$ADFA_{w/\ adduser}$	0.33	4819	1000	1132
LOGEVOL-U	Hadoop 2	Hadoop 3	0.84	302312	8558	34495
	Spark 2	Spark 3	0.50	11114	1134	4246
SYNEVOL-U	Spark 2	$Spark\ 2_{5\%\_sequence}$	0.6	11114	1134	2778
		$Spark\ 2_{10\%\_sequence}$	0.55			
		$Spark\ 2_{15\%\_sequence}$	0.50			
		$Spark\ 2_{20\%\_sequence}$	0.44			
		$Spark\ 2_{25\%\_sequence}$	0.37			
		$Spark\ 2_{30\%\_sequence}$	0.32			
		$Spark\ 2_{5\%\_template}$	0.54	11114	1134	2778
		$Spark\ 2_{10\%\_template}$	0.45			
		$Spark\ 2_{15\%\_template}$	0.36			
		$Spark\ 2_{20\%\_template}$	0.28			
		$Spark\ 2_{25\%\_template}$	0.22			
		$Spark\ 2_{30\%\_template}$	0.18			
SynHDFS-U	HDFS	$SynHDFS_{5\%\_sequence}$	0.93	460048	5772	51000
		$SynHDFS_{10\%\_sequence}$	0.88			
		$SynHDFS_{20\%\_sequence}$	0.78			
		$SynHDFS_{30\%\_sequence}$	0.69			

ADFA-U. We derived six ULAD configurations by splitting ADFA based on attack types. Specifically, for each configuration, five out of six attack types are used for training (e.g., $ADFA_{w/o\ java}$, in column "train" in Table 2, represents training data containing all attack types except the Java-based Meterpreter attack) and the remaining one for testing (e.g., $ADFA_{w/\ java}$, in the "test" column of Table 2, represents a testing dataset with only the Java-based Meterpreter attack), simulating external changes in real-world scenarios where novel attack types emerge during operation. Consequently, we obtain six ULAD configurations for ADFA-U involving $ADFA_{w/o\ java} \rightarrow ADFA_{w/\ java}$, $ADFA_{w/o\ hydraSSH} \rightarrow ADFA_{w/\ hydraSSH}$, $ADFA_{w/o\ hydraFTP} \rightarrow ADFA_{w/\ hydraFTP}$, $ADFA_{w/o\ meter} \rightarrow ADFA_{w/\ meter}$, $ADFA_{w/o\ web} \rightarrow ADFA_{w/\ web}$, $ADFA_{w/o\ adduser} \rightarrow ADFA_{w/\ adduser}$, denoted as "java", "hydraSSH", "hydraFTP", "meter", "web", and

"adduser" hereafter for brevity, respectively. As reported in Table 2, the duplication ratio ranges from 0.31 to 0.34 in different training and testing dataset pairs, indicating that, without de-duplication, approximately 31 % to 34 % log sequences in the testing datasets are already included in the training datasets.

LOGEVOL-U. The LOGEVOL dataset naturally captures software evolution, namely the transition from Hadoop 2 to Hadoop 3, as well as from Spark 2 to Spark 3. These transitions result in internal changes at both template and sequence levels (defined in § 2.1). Hence, we use the Hadoop 2 and Spark 2 datasets for training and, correspondingly, the Hadoop 3 and Spark 3 datasets for testing. The duplication ratios for LOGEVOL Hadoop and LOGEVOL-Spark, as shown in Table 2, are 0.84 and 0.5, respectively; these values indicate that, without de-duplication, 84% of Hadoop and 50% of Spark log sequences in the testing dataset are already included in the training dataset.

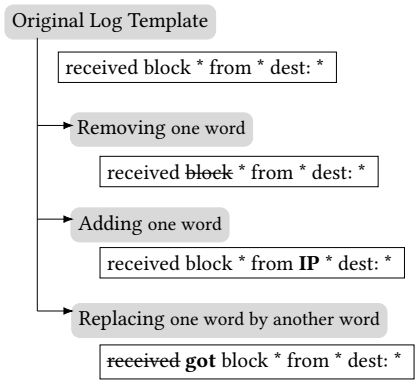


Fig. 5. Examples of Template-level Changes

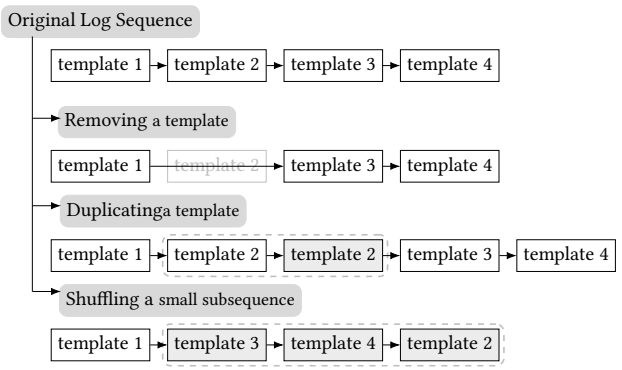


Fig. 6. Examples of Sequence-level Changes

SYNEVOL-U. The ULAD configurations in this dataset aim to simulate different levels of instability by applying internal changes of varying percentages to the LOGEVOL dataset. Huo et al. [39] injected log template/sequence-level changes into the LOGEVOL Spark 2 dataset, with varying injection ratio of 5 %, 10 %, 15 %, 20 %, 25 %, and 30 %. Figure 6 demonstrates the three types of log sequence-level changes injected, firstly introduced by Zhang et al. [102], involving removing or duplicating a log template and shuffling a small subsequence. As shown in Figure 5, we introduce three types of log template-level changes, including adding, removing, or replacing a word in a log template. Huo et al. [39] injected changes in the sequences in a way that sequence labels do not flip. The duplication ratio decreases from 0.6 to 0.18 as the testing set becomes more unstable.

SynHDFS-U. Similar to SYNEVOL-U, we created four ULAD configurations for the HDFS dataset, namely $\text{SynHDFS}_{5\%}$, $\text{SynHDFS}_{10\%}$, $\text{SynHDFS}_{20\%}$, and $\text{SynHDFS}_{30\%}$ by changing 5%, 10%, 20%, and 30% of log sequences in the HDFS dataset, respectively. The injection ratios are determined by following common practices in the literature [102]. Similar to SYNEVOL-U, we are aware that such changes in log sequences can induce changes in their labels. We only apply sequence-level changes, excluding template-level changes due to the lack of implementation details reported by previous studies [38, 56, 102]. At the sequence-level, to obviate the need for re-labeling, we applied changes only to log templates that are less likely to flip the labels of the entire log sequence. These log templates are identified by a strategy proposed by Xu et al. [91], which combines building a decision tree and manual examination. To reduce the cost of manual examination, we sampled and applied changes to a subset of the HDFS dataset instead of the full dataset. Concretely, we

Table 3. Overview of Baselines

Learning Method	Approach	Parser	Log Representation	ML Method	Base Model
Unsupervised	PCA	Yes	Template ID	Traditional ML	PCA
	LogCluster	Yes	Template ID	Traditional ML	Clustering
	DeepLog	Yes	Template ID	Deep Learning	LSTM
	LogAnomaly	Yes	Template2Vec	Deep Learning	LSTM
Semi-supervised	PLELog	Yes	FastText and TF-IDF	Deep Learning	GRU
Supervised	LogRobust	Yes	FastText and TF-IDF	Deep Learning	BiLSTM
	CNN	Yes	Logkey2vec	Deep Learning	CNN
	NeuralLog	No	BERT	Deep Learning	Transformer
	LightAD	Yes	TemplateID	Traditional ML & Deep Learning	KNN, DT, SLFN

randomly selected 50,000 normal and 1,000 anomalous log sequences, following the study by Zhang et al. [102], to keep the anomaly percentage (2 %) close to that of the original HDFS dataset. The duplication ratio decreases from 0.93 to 0.69 as the testing set becomes more unstable.

SLAD Configuration. For the ADFA dataset, we drew the training and testing data from the full dataset, containing all six types of anomalies. For LOGEVOL Hadoop 2, Spark 2, and HDFS, we adopt the same training datasets as in their ULAD configurations, whereas the testing datasets differ. While ULAD employs unstable testing data, SLAD uses stable testing data collected from the same system as the training data, specifically from Hadoop 2, Spark 2, and HDFS operations, respectively. Notably, the testing dataset for HDFS SLAD configuration is the same as the one used in its ULAD configuration, a subset sampled from the original testing dataset, but without instability injection. This ensures consistency and a fair comparison between the ULAD and SLAD configuration of HDFS. Also, we did not configure LOGEVOL Hadoop 3 and Spark 3 for SLAD as ULAD because the evolution information from Hadoop 3 and Spark 3 to other versions was not available.

Training Dataset Configuration for FlexLog. In RQ1, we compare FlexLog and the baselines with their respective optimal settings. Baselines are trained on the full training datasets $\mathcal{D}^{train}$, following implementations in their original papers, whereas FlexLog is trained on small subsets $\tilde{\mathcal{D}}^{train}$ randomly sampled from $\mathcal{D}^{train}$. As reported in the second-to-last column ("$\tilde{\mathcal{D}}^{train}_{\#}$") of Table 2, their data sizes are determined empirically for each dataset to achieve the optimal performance of FlexLog. For small datasets with low duplication ratios such as ADFA-U, we randomly selected 1,000 log sequences from their full training dataset. For larger datasets with high duplication ratios such as LOGEVOL-U Hadoop, LOGEVOL-U Spark, SYNEVOL-U, and SynHDFS-U, unique anomalous log sequences are rare, accounting for only 0.2 % to 2 % of the full datasets, respectively. To maximize the use of these rare anomalous log sequences, we included all of them in the fine-tuning datasets and sampled 20 % unique normal log sequences from all unique normal log sequences, preventing excessive duplication.

4.2.3 Baselines. We considered nine ML methods as baselines in this paper, including four unsupervised, one semi-supervised, and four supervised. Among these methods, LightAD [96] achieves the best performance on the SLAD task. However, the leading approach for ULAD remains undetermined as different evaluation datasets are used in reported studies. Our choice of baselines is also determined by source code availability to ensure the reliability of the implementation. Consequently, we had to exclude models such as SwissLog [56], HitAnomaly [38], EvLog [39], and LLMeLog [29]. Our implementations are based on the code provided by Yu et al. [96], Le and Zhang [53], and He

et al. [32]. We have also not included LogPrompt [59] in our evaluation since it relies on anomaly detection at the message level, ignoring sequential characteristics such as temporal dependencies, whereas our datasets are labeled at the sequence level.

Table 3 shows the main characteristics of the baselines; we provide a brief description in the following. *Principal Component Analysis (PCA)* [91], a dimensionality reduction method, converts logs into count vectors [16] and then uses the PCA algorithm to detect the label of log sequences by assigning them to either the normal or anomalous space. In this paper, by PCA we refer to the PCA-based model introduced by Xu et al. [91] as an anomaly detector. *LogCluster* [57] clusters log sequences by computing the similarity of log representations to the centroid of normal logs. *DeepLog* [21] applies two layers of long short-term memories (LSTMs) in their network [35] to predict the next event from a given log sequence and labels sequences as anomalous if the predicted log is different than the actual log template. *LogAnomaly* [65] has an architecture similar to DeepLog, but is further improved by adopting semantic embeddings for log templates and adding an attention layer between LSTM layers. *PLELog* [93] is a semi-supervised strategy that uses normal data as well as a small subset of unlabeled data to train. First, it adopts a clustering method (HDBSCAN [64]) to probabilistically predict the labels of unlabeled data and then uses them to train an attention-based GRU [9] to detect anomalies.

LogRobust [102] uses a pre-trained word vectorizer (FastText [44]) to extract semantic information from log templates and utilizes an attention-based BiLSTM model [76] to detect anomalous log sequences. *CNN* [62] transforms an input log sequence into a trainable matrix and uses this matrix as input to train a Convolutional Neural Network [48, 55] for log-based anomaly detection. *NeuralLog* [51] extracts the semantic meaning of raw log messages and represents them as semantic vectors, which are then used to detect anomalies through a transformer-based classification model [82]. *LightAD* [96] employes Bayesian method to select the most effective model from a heterogeneous pool of ML/DL algorithms—including KNN [22], DT [7], and SLFN [37]—while simultaneously optimizing hyperparameters for the SLAD task. To ensure fair comparisons, we adopted the same model pool and employed a small, held-out validation dataset to identify the optimal model for each dataset as instructed in Yu et al. [96]. The performance of the optimal model, evaluated on test data, serves as LightAD's reported effectiveness.

4.2.4 Evaluation Metrics and Statistical Testing. To provide a comprehensive evaluation, we assess FlexLog in terms of effectiveness, data efficiency, and time efficiency. Further, we investigate the statistical significance of differences (from a point of view of effectiveness and time efficiency) on each dataset.

Effectiveness To measure the effectiveness of FlexLog, we use Precision, Recall, and F1-score as metrics. We consider TP (true positive) as the number of anomalies that are correctly detected by the model, FP (false positive) as the number of normal log sequences that are labeled as anomalous by the model, and FN (false negative) as the number of anomalous log sequences that the model fails to identify. Precision (P) is calculated by $\frac{TP}{TP+FP}$ as the percentage of true anomalies among all anomalies detected by the model. Recall (R) is the proportion of actual anomalies detected, computed by $\frac{TP}{TP+FN}$. F1 score (F1) is the harmonic mean of Precision and Recall, i.e., $2 * \frac{P*R}{P+R}$.

Data Efficiency We define data efficiency to be the ability of a method to achieve accurate results while minimizing the use of labeled data for training. Considering a training dataset with $\mathcal{D}_{\#}$ log sequences, we quantify the usage of labeled data using $\mathcal{U}_{\#}$, which represents the number of unique log sequences. Each unique log sequence corresponds to a distinct pattern that requires annotation, meaning that a higher $\mathcal{U}_{\#}$ reflects greater labeling effort.

A data-efficient method, such as FlexLog, requires only a small subset of the full dataset for training, reducing the overall usage of labeled data. To compare data efficiency across different methods, we introduce the relative metric $\mathcal{U}_{\%}$ as in Equation 7, which measures the percentage of unique log sequences in the subset relative to the total unique log sequences in the full dataset (denoted by $\mathcal{U}_{\#}^{full}$).

$$\mathcal{U}_{\%} = \frac{\mathcal{U}_{\#}}{\mathcal{U}_{\#}^{full}} \tag{7}$$

To quantify the reduction in labeled data achieved by data-efficient methods compared to methods trained on full datasets, we define the labeled data usage reduction $\Delta\mathcal{U}_{\%}$ as in Equation 8. A higher $\Delta\mathcal{U}_{\%}$ indicates a greater reduction, demonstrating the superior data efficiency of the method under evaluation, and vice versa.

$$\Delta\mathcal{U}_{\%} = 1 - \mathcal{U}_{\%} \tag{8}$$

Time Efficiency We evaluate the time efficiency in terms of training and inference time for each model. For training time, we calculate the total training time taken for a model. For inference time, we calculate the average inference time for one input sequence in the testing set.

Memory Efficiency We evaluate memory efficiency based on the additional memory required by the cache component during inference. Specifically, we measure the memory overhead introduced by the in-memory cache, which stores predictions for all previously seen unique sequences. To approximate this overhead, we compute the memory consumed by the cache in the structure of a dictionary after processing the entire test set. Since the cache grows incrementally with the number of unique sequences, this measurement represents its maximum size at the end of inference.

Statistical Testing To mitigate the potential influence of randomness on our results, we repeat each experiment on each configuration 5 times and report the average performance across the runs. This ensures our analysis is robust and not unduly influenced by any single random sampling or stochastic training and fine-tuning, providing a reliable evaluation. We further perform Mann-Whitney U test as recommended in [1] on each dataset, resulting in test group sizes of 30 for ADFA-U (6 configurations), 10 for LOGEVOL-U (2 configurations), 20 for SynHDFS-U (4 configurations), and 30 for SYNEVOL-U (6 configurations).

The Mann-Whitney U test is a non-parametric statistical test that compares two methods, A and B, without any assumption of the data distribution. It computes a p-value, which indicates whether the observed performance difference is statistically significant. The null hypothesis presumes no significant distinction between performance A and B. If the p-value falls below the commonly used threshold of 0.05, we reject the null hypothesis and conclude that the difference is statistically significant. Conversely, if the p-value is greater than or equal to 0.05, the difference is considered non-significant, meaning the observed difference could be due to randomness.

4.2.5 Other Settings. We conducted all experiments with on a cloud computing environment containing 28 CPU cores for computation, 2 × Nvidia L40S GPU devices, and 256 GB RAM.

4.3 Implementation

In this section, we introduce the implementation details of preprocessing, FlexLog, and the baselines.

4.3.1 Preprocessing. Two primary steps of preprocessing are parsing and partitioning, as described in Section 3.

Log parsing is used to provide structured context for FlexLog (as explained in § 3.1) and log-parsing-based baselines such as LogRobust and CNN. For ADFA-U, parsing is not required since each log message is a one-word system call. For datasets with evolving templates—ADFA-U and SYNEVOL-U —we follow their original authors' practice and use the Prefix Graph parser [10]. This parser does not require a training set and is more flexible in handling varying template lengths and substructures compared to fixed-depth approaches such as Drain [30]. For SynHDFS-U, we use Drain following the common practice for this specific synthetic dataset [38, 102][2].SynHDFS-U exhibits instability at the sequence level, but its log templates remain stable; hence, the limitations of Drain in handling evolving templates do not apply in this case. Since each log message in ADFA consists of a one-word system call, the subsequent RAG can easily associate a log message with its relevant information, such as the description of that system call.

Applying a smaller window over long sequences facilitates the localization of anomalies when logs are labeled at the message level. Hence, we applied sliding window-based partitioning with a window size of 50 on LOGEVOL-Hadoop. In contrast, sliding window partitioning is not an option for long sequences in the ADFA, LOGEVOL-Spark, SYNEVOL-U, and HDFS datasets due to the absence of message-level labels. For HDFS, most sessions are short, with only 3.5 % sessions exceeding 30 templates. We followed the implementation of Le and Zhang [53] and truncated these long sessions to ensure that all sessions were within the 30-template limit.

4.3.2 FlexLog. The implementation of FlexLog mainly involves two key aspects, namely, the selection of base models for ensemble learning and the fine-tuning of the LLM base model.

Base Model Selection FlexLog combines multiple heterogeneous base models through ensemble learning, including ML/DL models and LLMs. Specifically, the ensemble in FlexLog comprises three ML base models (KNN, DT, and SLFN) with one LLM base model (Mistral [67]). We selected KNN, DT, and SLFN due to their high effectiveness on SLAD task as reported by Yu et al. [96]. We selected the LLM base model (Mistral 22B) through the empirical evaluation of multiple open-source and closed-source LLMs. To elaborate, closed-source LLMs like GPT are considered state-of-the-art LLMs in various domains, albeit at a high cost [95]. We experimented with a major version— GPT-4o (`GPT-4o-turbo` version) — based on OpenAI's recommendation in terms of performance [71, 72]. In contrast, open-source LLMs incur no cost and offer more flexibility regarding fine-tuning and inference. Within the limit of our computing resources, we explored two open-source LLMs that have shown competitive performance to closed-source LLMs [23, 94]: LLama 3.1 8B and Mistral 22B.

Fine-tuning LLMs For open-source LLMs, we utilized 4-bits QLora [17] for fine-tuning, with the following configurations based on our preliminary experiments: rank=16, alpha=16, and batch=1. We fine-tuned Llama 3.1 8B and Mistral Small 22B using the Unsloth library due to its high efficiency [81]. The number of steps is empirically tuned for unique pairs of LLM and dataset separately, using grid-search and cross-validation; values range from 500 to 2500 in steps of 500. For closed-source LLM of GPT-4o, fine-tuning was accomplished through OpenAI API, which incurred an associated cost. Hyperparameters such as the number of epochs and batch size for fine-tuning GPT were optimized and automatically determined by OpenAI fine-tuning APIs on each dataset.

LLMs, even after fine-tuning, tend to generate non-deterministic output, which threatens the reliability of FlexLog for ULAD. To ensure reliable anomaly detection, we instruct the LLMs to generate the response with minimum temperature (e.g., 0.1 for Mistral). In case the responses deviate from explicit labels (e.g., "normal", "anomalous", "0" or "1"), ambiguous responses trigger

[2]We acknowledge that on the original HDFS dataset, more recent log parsers [43, 63] demonstrated higher parsing effectiveness than Drain. However, as a recent study [47] demonstrated, there is no correlation between parsing accuracy and anomaly detection accuracy.

up to five regeneration attempts with progressively higher temperatures from 0.2 to 1, in steps of 0.2, to diversify outputs. If no valid label is parsed after all attempts, we classify the sequence as "normal" to reduce false positives, which might trigger alert fatigue and operational disruption unnecessarily.

Hyper-parameters settings of ML base models in FlexLog For the DT and SLFN base models, we use the default values provided by Scikit-learn Library across all datasets. Our preliminary experiments suggest that tuning these hyper-parameters with limited data often leads to overfitting and reduced effectiveness compared to the default settings. While further tuning could potentially improve FlexLog's performance, we leave this for future work. Specifically for DT, we set criterion to "gini", max_depth to "None", and min_samples_split to 2. For SLFN, we set hidden_layer_sizes to 100, activation to "relu", solver="adam", and batch_size to "auto". For KNN, since the number of neighbors plays an important role in handling imbalanced datasets [24], we empirically tuned it with grid search and cross-validation on limited training data. We set the number of neighbors to 2 for ADFA and LOGEVOL Hadoop, and 1 for HDFS[3]. For the extremely imbalanced datasets—LOGEVOL Spark and SYNEVOL-U, which share the same training set—KNN performs poorly on the validation set, exhibiting significantly lower effectiveness compared to DT and SLFN. This aligns with known limitations of KNN on highly imbalanced datasets, where the majority class tends to dominate the predictions [99]. Therefore, we excluded KNN from FlexLog for LOGEVOL Spark and SYNEVOL-U.

Lastly, for the ensemble strategy, we implemented a majority voting algorithm in which, in the event of a tie, the sequence is labeled as normal, as described in § 3.4. Our preliminary results across all four datasets indicate that this strategy remains the most effective and straightforward compared to alternative ensemble learning approaches, including SNAIL [66] and MetaFormer [98] (see Appendix A.1).

4.3.3 Baselines . For baselines, we set hyper-parameters as reported in their original papers or suggested by their implementation packages. When hyper-parameters were not available in either of them, particularly for datasets such as ADFA, we empirically tuned the parameters by grid search with a cross-validation approach. For LightAD, we fine-tuned hyper-parameters using Bayesian optimization, available in their implementation code for KNN, DT, and SLFN. LogAnomaly's representation model (template2vec) requires domain-specific antonyms and synonyms for training. This information is unavailable in its original paper and, thus, similar to the method previously adopted [53], we used a pre-trained FastText model [44] to compute the semantic vectors. As LOGEVOL Hadoop and SynHDFS training sets are too large to process on NeuralLog, we used a subset containing the first 200,000 log messages, following the methodology adopted in prior work [53].

4.4 Data availability.

The replication package, including our synthesized datasets, additional experiment results, and source code, is publicly available [26].

Table 4. Statistics of training data for FlexLog and baselines used in RQ1 on ADFA-U, LOGEVOL, SYNEVOL-U and SynHDFS-U.

Dataset	$\mathcal{D}_{\#}$		$\mathcal{U}_{\#}$		$\mathcal{U}_{\%}$		$\Delta\mathcal{U}_{\%}$
	Baselines	FlexLog	Baselines	FlexLog	Baselines	FlexLog	
ADFA-U	4,785	1,000	3,181	686	100 %	21.57%	78.43 pp
LOGEVOL-U	313,426	9,692	29,809	9,692	100 %	32.51%	67.49 pp
SYNEVOL-U	11,114	1,134	4,939	1,134	100 %	22.96%	77.04 pp
SynHDFS-U	460,048	5,772	15,545	5,772	100 %	37.13%	62.87 pp

Table 5. Effectiveness of FlexLog and baselines for ULAD and SLAD on the ADFA dataset

Data	Unstable	M	limited data	full training set								
				Supervised				Semi-S	Unsupervised			
			FlexLog	LightAD	NeuralLog	LogRobust	CNN	PLELog	LogAnomaly	DeepLog	LogCluster	PCA
ADFA	No	P	0.708	0.820	0.538	0.718	0.666	0.736	0.357	0.334	0.255	0.196
		R	0.894	0.814	0.602	0.708	0.842	0.216	0.430	0.523	0.907	0.139
		F1	0.791	**0.817**	0.568	0.713	0.744	0.334	0.390	0.408	0.398	0.162
adduser	Yes	P	0.619	0.673	0.303	0.711	0.547	0.507	0.208	0.198	0.217	0.139
		R	0.857	0.786	0.651	0.415	0.775	0.281	0.483	0.562	0.725	0.202
		F1	0.718	**0.725**	0.412	0.524	0.641	0.361	0.291	0.292	0.334	0.165
hydraFTP	Yes	P	0.686	0.780	0.532	0.612	0.882	0.247	0.345	0.411	0.268	0.139
		R	0.913	0.731	0.306	0.306	0.653	0.897	0.650	0.581	0.950	0.133
		F1	**0.784**	0.754	0.388	0.408	0.750	0.388	0.451	0.481	0.418	0.144
hydraSSH	Yes	P	0.657	0.833	0.298	0.656	0.882	0.522	0.408	0.390	0.299	0.121
		R	0.804	0.635	0.479	0.262	0.653	0.433	0.583	0.554	0.999	0.121
		F1	**0.723**	0.666	0.368	0.374	0.750	0.473	0.480	0.458	0.461	0.140
java	Yes	P	0.634	0.699	0.366	0.695	0.752	0.371	0.357	0.269	0.213	0.169
		R	0.650	0.590	0.849	0.457	0.549	0.513	0.516	0.459	0.959	0.157
		F1	0.642	0.639	0.511	0.558	0.635	0.430	0.422	0.339	0.348	0.158
web	Yes	P	0.639	0.788	0.330	0.583	0.786	0.799	0.254	0.335	0.186	0.204
		R	0.709	0.487	0.741	0.395	0.513	0.136	0.530	0.373	0.829	0.191
		F1	**0.672**	0.602	0.457	0.449	0.621	0.233	0.343	0.353	0.304	0.197
meter	Yes	P	0.564	0.710	0.312	0.745	0.642	0.569	0.147	0.178	0.235	0.210
		R	0.859	0.732	0.889	0.577	0.799	0.163	0.425	0.436	0.943	0.139
		F1	**0.682**	0.679	0.461	0.651	0.711	0.253	0.218	0.253	0.175	0.241
average	Yes	P	0.633	0.747	0.357	0.667	0.748	0.503	0.286	0.297	0.236	0.164
		R	0.799	0.660	0.652	0.402	0.657	0.404	0.531	0.494	0.901	0.157
		F1	**0.704**	0.677	0.433*	0.494*	0.685	0.356*	0.368*	0.363*	0.340*	0.174*

* FlexLog yields a significant higher F1-score than compared baseline.

5 RESULTS

5.1 RQ1: Overall Effectiveness

RQ1 investigates the effectiveness of FlexLog trained with limited data and compares it to baselines trained with full datasets on ULAD and SLAD. This comparison between FlexLog and baselines is

[3]Overall, due to the limited training data, we recommend using default parameters and tuning only those that are highly sensitive to imbalanced data, as anomalies are often rare in real-world datasets, using cross-validation. In the future, we plan to conduct more experiments to explore the correlation between the percentage of anomalous data and the optimal number of neighbors.

Table 6. Effectiveness of FlexLog and baselines for ULAD and SLAD on the LOGEVOL dataset

Data	Unstable	M	limited data	full training set								
				supervised				Semi-S	Unsupervised			
			FlexLog	LightAD	NeuralLog	LogRobust	CNN	PLELog	LogAnomaly	DeepLog	LogCluster	PCA
$Hadoop_{2\to2}$	No	P	0.999	0.999	0.997	0.984	0.997	0.648	0.263	0.384	0.952	0.267
		R	0.986	0.994	0.986	0.976	0.997	0.888	0.616	0.352	0.320	0.867
		F1	0.993	**0.997**	0.992	0.980	**0.997**	0.749	0.368	0.367	0.479	0.408
$Spark_{2\to2}$	No	P	0.999	0.999	0.999	0.941	0.999	0.172	0.501	0.512	0.771	0.072
		R	0.969	0.939	0.636	0.969	0.878	0.129	0.393	0.443	0.818	0.471
		F1	**0.984**	0.968	0.777	0.952	0.935	0.243	0.441	0.475	0.794	0.125
$Hadoop_{2\to3}$	Yes	P	0.998	0.998	0.914	0.905	0.992	0.626	0.221	0.384	0.510	0.225
		R	0.965	0.963	0.984	0.950	0.968	0.819	0.522	0.352	0.371	0.898
		F1	**0.982**	0.980	0.948	0.927	0.980	0.709	0.310	0.367	0.430	0.360
$Spark_{2\to3}$	Yes	P	0.999	0.981	0.916	0.696	0.992	0.105	0.141	0.08	0.347	0.061
		R	0.805	0.708	0.766	0.832	0.736	0.377	0.458	0.606	0.805	0.484
		F1	**0.892**	0.829	0.834	0.757	0.840	0.165	0.216	0.122	0.485	0.108
Average ($Spark_{2\to3}$, $Hadoop_{2\to3}$)	Yes	P	0.998	0.99	0.915	0.786	0.992	0.366	0.181	0.232	0.428	0.143
		R	0.871	0.83	0.875	0.863	0.852	0.887	0.49	0.479	0.588	0.691
		F1	**0.928**	0.898*	0.891*	0.833*	0.910*	0.437*	0.263*	0.244*	0.458*	0.234*

* FlexLog yields a significant higher F1-score than baseline.

particularly less advantageous for FlexLog, as it is trained with less data than the baselines[4]. To prevent inflated effectiveness caused by data leakage, test data is de-duplicated from the training data, as explained in § 3.4; a comparison of model performance evaluated on the test set with and without de-duplication is also provided in Appendix A.2.

Table 4 reports the configurations of training datasets for FlexLog and baselines, including the training dataset size ($\mathcal{D}_{\#}$), the number of unique log sequences ($\mathcal{U}_{\#}$), the percentage of unique log sequences relative to the total unique log sequences in the full datasets ($\mathcal{U}_{\%}$) and the reduction in percentage points in labeled data achieved by FlexLog ($\Delta\mathcal{U}_{\%}$); please refer to § 4.2.4 for details of these metrics. In the rest of this section, we first report the evaluation of the overall effectiveness of FlexLog and baselines on two real-world unstable datasets, LOGEVOL-U and ADFA-U (§ 5.1.1). To further analyze the influence of instability, we conducted controlled experiments on two synthesized datasets, SYNEVOL-U and SynHDFS-U, where varying levels of instability are systematically introduced (§ 5.1.2).

5.1.1 Effectiveness on Real-world Datasets. Table 5 and Table 6 report the effectiveness of FlexLog on two real-world datasets, LOGEVOL and ADFA, respectively. Column *"Data"* specifies different dataset configurations, including SLAD alongside with various ULAD settings. For instance, $Hadoop_{2\to3}$ represents that logs produced from Hadoop 2 and Hadoop 3 are used as training and testing datasets, respectively. Column "Unstable" indicates whether the data configuration is unstable or not. Column *"M"* reports the effectiveness metrics introduced in § 4.2.4. We recall in this table that FlexLog is trained with only *"limited data"* while all baselines are all trained with the *"full training dataset"*, including *"supervised"*, *"semi-supervised"* and *"unsupervised"* baselines.

As shown in Table 5, in the SLAD of the ADFA dataset, FlexLog achieves an F1 score of 0.791, second only to LightAD (0.817) among all methods. However, in the ULAD, where log instability arises from environmental changes, i.e., the introduction of novel attack types, FlexLog outperforms all baselines on 5 out of 6 configurations. The only exception occurs in the *adduser configuration*, where LightAD surpasses FlexLog with a small margin of 0.7 pp (72.5% – 71.8%). Overall, FlexLog

[4]The effectiveness of baselines when trained on the same limited data as FlexLog is reported in Appendix A.3.

yields the highest average F1 score of 0.704 in ULAD configurations, followed by CNN (0.685) and LightAD (0.677). A Mann-Whitney U test indicates that the differences between the average F1 scores of FlexLog and LightAD, as well as the differences between FlexLog and CNN, are statistically insignificant, whereas FlexLog outperforms all other baselines significantly. Moreover, when moving from SLAD to ULAD, the most effective baseline in SLAD (LightAD) experiences an average F1 score decrease of 14 pp, whereas the decrease with FlexLog remains below 9 pp, alleviating the impact of unstable logs on anomaly detection effectiveness.

Notably, FlexLog achieves such high effectiveness on the ADFA dataset with only limited training data, while baselines like LightAD and CNN are trained with full datasets. As shown in Table 4, FlexLog's fine-tuning datasets contain only 21.57 % of unique log sequences relative to the total unique log sequences in the full ADFA datasets, each requiring a dedicated label. This translates to a reduction of 78.43 pp in usage of labeled data while achieving state-of-the-art effectiveness on the ADFA dataset.

We observe similar results on the LOGEVOL-U dataset, where instability is introduced due to system evolution, e.g., software system version updates. As shown in Table 6, we evaluated two SLAD configurations ($\text{Hadoop}_{2\to2}$ and $\text{Spark}_{2\to2}$) and two ULAD configurations ($\text{Hadoop}_{2\to3}$ and $\text{Spark}_{2\to3}$). In the $\text{Hadoop}_{2\to2}$ SLAD configuration, all supervised approaches achieve a high F1 score ($\geq$ 0.980), including FlexLog, LightAD, NeuralLog, LogRobust, and CNN. In the other SLAD configuration $\text{Spark}_{2\to2}$, FlexLog surpass baselines, reaching an F1 score of 0.984 by 1.4 pp from the top baseline, LightAD. In the $\text{Hadoop}_{2\to3}$ and $\text{Spark}_{2\to3}$ ULAD configurations, FlexLog yields F1 scores of 0.982 and 0.892, respectively, remaining the best approach compared to all the baselines. On average, FlexLog outperforms all the baselines in terms of F1 score, with a minimum margin of 1.8 pp (0.928 − 0.910). Additionally, when moving from SLAD to ULAD, the most effective baseline in SLAD (LightAD) experiences an average F1 score decrease of 10 pp, whereas the decrease with FlexLog remains below 7 pp, alleviating once again the impact of unstable logs on anomaly detection effectiveness.

Similar to the ADFA-U dataset, FlexLog achieves such high effectiveness on the LOGEVOL dataset by training on a relatively limited dataset instead of the entire LOGEVOL training dataset. Specifically, as depicted in Table 4, FlexLog's fine-tuning dataset contain only 32.51 % of unique log sequences relative to the total unique log sequences in the full LOGEVOL datasets, indicating a reduction in usage of labeled data of 67.49 pp.

5.1.2 Impact of Log Instability. As detailed in § 4.2.2, we conducted comprehensive experiments on two synthesized datasets, SynHDFS-U and SYNEVOL-U, to analyze the impact of instability at both the log sequence and log template levels. Specifically, SYNEVOL-U exhibits instability at both the sequence and template levels, whereas SynHDFS-U is characterized solely by sequence-level instability.

Instability at Log Sequence Level. Table 7 presents the performance of FlexLog and baseline approaches on SynHDFS-U with sequence-level instability. As the injection ratio of changes increases, FlexLog consistently achieves the highest F1 score across all unstable configurations, demonstrating its robustness to varying levels of instability. On average, FlexLog outperforms all baselines, exceeding the top-performing baseline, LightAD, by a margin of 1.2 pp in F1 score. A Mann-Whitney test reveals that the difference in F1 scores between FlexLog and LightAD is statistically insignificant, indicating comparable performance between the two. However, unlike all baselines—including LightAD—which are trained on the full dataset, FlexLog is trained on only a small subset comprising 37.13 % of unique log sequences (Table 4). This means that FlexLog achieves similar effectiveness and robustness while reducing the usage of labeled data by 62.87 pp.

Table 7. Effectiveness of FlexLog and baselines under different sequence-level injection ratios on SynHDFS-U.

Data	Unstable	M	limited data	full training set								
				supervised				Semi-S	Unsupervised			
			FlexLog	LightAD	NeuralLog	LogRobust	CNN	PLELog	LogAnomaly	DeepLog	LogCluster	PCA
0%	No	P	0.954	0.9763	0.949	0.957	0.933	0.630	0.243	0.725	0.999	0.924
		R	0.999	0.990	0.985	0.980	0.951	0.966	0.971	0.927	0.346	0.667
		F1	0.976	**0.983**	0.966	0.969	0.942	0.763	0.389	0.814	0.514	0.762
5%	Yes	P	0.953	0.957	0.944	0.995	0.932	0.602	0.236	0.678	0.986	0.976
		R	0.995	0.985	0.985	0.979	0.952	0.554	0.961	0.894	0.346	0.667
		F1	**0.974**	0.971	0.964	0.878	0.944	0.577	0.380	0.771	0.512	0.792
10%	Yes	P	0.954	0.966	0.914	0.692	0.933	0.404	0.239	0.649	0.999	0.225
		R	0.995	0.980	0.980	0.974	0.951	0.884	0.975	0.927	0.350	0.673
		F1	**0.974**	0.973	0.946	0.809	0.942	0.555	0.385	0.764	0.519	0.337
20%	Yes	P	0.949	0.943	0.906	0.560	0.933	0.345	0.482	0.644	0.949	0.158
		R	0.995	0.966	0.980	0.956	0.951	0.798	0.538	0.889	0.360	0.694
		F1	**0.971**	0.954	0.942	0.707	0.942	0.482	0.509	0.747	0.522	0.258
30%	Yes	P	0.940	0.925	0.896	0.489	0.929	0.243	0.464	0.548	0.915	0.137
		R	0.985	0.956	0.970	0.951	0.947	0.877	0.572	0.903	0.365	0.718
		F1	**0.964**	0.940	0.931	0.646	0.938	0.381	0.512	0.682	0.522	0.230
Average	Yes	P	0.949	0.948	0.915	0.684	0.932	0.398	0.355	0.63	0.962	0.374
		R	0.992	0.972	0.979	0.965	0.95	0.778	0.762	0.903	0.355	0.688
		F1	**0.971**	0.959	0.946*	0.760*	0.942*	0.499*	0.446*	0.741*	0.519*	0.404*

* FlexLog yields a significant higher F1-score than compared baseline.

The effectiveness of FlexLog and baselines on SYNEVOL-U under varying log sequence instability demonstrated similar results to SynHDFS-U that we have provided in Appendix A.4.

Instability at Log Template Level. Table 8 reports the effectiveness of FlexLog compared with baselines on SYNEVOL-U under varying template-level changes. Once again, FlexLog achieves the highest precision, recall and F1 score across all injection ratios, outperforming the top-performing baseline—LightAD—by 1.9 pp (96.3% − 94.4%) in terms of average F1 score while reducing usage of labeled data by 77.04 pp. A Mann-Whitney U test suggests this difference is statistically significant. Similar to sequence-level changes in SYNEVOL-U, increasing template-level instability negatively impacts the effectiveness of semi-supervised and unsupervised approaches, whereas supervised methods—FlexLog, LightAD, NeuralLog, LogRobust, and CNN—are relatively robust in terms of effectiveness across instability levels. This is likely because many template modifications in SYNEVOL-U involve minor textual variations, such as inserting, removing, or replacing short tokens, which have minimal impact on ULAD, especially in long log sequences with up to 1,818 templates. For instance, a log template stating "`SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(root); groups with view permissions: Set(); users with modify permissions: Set(root); ...`" was modified by inserting "`so`" before "`users with modify permissions`", a minor change that has limited impact on ULAD effectiveness.

The answer to RQ1 is that FlexLog achieves **state-of-the-art effectiveness** on both real-world and synthesized datasets while exhibiting **high data efficiency** (with a reduction in labeled data usage ranging between 62.87 pp and 78.43 pp). On synthesized datasets, FlexLog remains effective under up to 30 % sequence- and template-level instability.

Table 8. Effectiveness of FlexLog and baselines under different template-level injection ratios on SYNEVOL-U.

Data	Unstable	M	limited data	full training set								
				supervised				Semi-S	Unsupervised			
			FlexLog	LightAD	NeuralLog	LogRobust	CNN	PLELog	LogAnomaly	DeepLog	LogCluster	PCA
0%	No	P	0.999	0.999	0.999	0.941	0.999	0.172	0.501	0.512	0.771	0.072
		R	0.969	0.939	0.636	0.969	0.878	0.129	0.393	0.443	0.818	0.471
		F1	**0.984**	0.968	0.777	0.952	0.935	0.243	0.441	0.475	0.794	0.125
5%	Yes	P	0.999	0.999	0.999	0.943	0.999	0.127	0.351	0.282	0.651	0.062
		R	0.970	0.941	0.647	0.971	0.882	0.315	0.393	0.382	0.823	0.500
		F1	**0.985**	0.969	0.785	0.956	0.937	0.181	0.440	0.325	0.727	0.111
10%	Yes	P	0.999	0.999	0.999	0.921	0.937	0.120	0.260	0.181	0.622	0.054
		R	0.942	0.914	0.600	0.969	0.857	0.316	0.371	0.400	0.800	0.457
		F1	**0.970**	0.955	0.750	0.944	0.895	0.174	0.305	0.250	0.700	0.097
15%	Yes	P	0.999	0.999	0.999	0.944	0.916	0.117	0.265	0.180	0.604	0.054
		R	0.921	0.894	0.605	0.918	0.891	0.315	0.447	0.447	0.763	0.447
		F1	**0.958**	0.944	0.754	0.931	0.904	0.171	0.333	0.257	0.674	0.097
20%	Yes	P	0.999	0.999	0.916	0.943	0.906	0.112	0.180	0.095	0.457	0.047
		R	0.941	0.911	0.647	0.951	0.852	0.315	0.382	0.411	0.794	0.470
		F1	**0.969**	0.953	0.758	0.946	0.878	0.165	0.245	0.155	0.580	0.085
25%	Yes	P	0.999	0.999	0.999	0.908	0.916	0.127	0.188	0.100	0.507	0.051
		R	0.904	0.850	0.625	0.954	0.846	0.239	0.400	0.400	0.800	0.475
		F1	**0.950**	0.918	0.769	0.930	0.880	0.166	0.256	0.160	0.621	0.093
30%	Yes	P	0.973	0.999	0.931	0.923	0.947	0.121	0.180	0.096	0.438	0.053
		R	0.925	0.857	0.642	0.954	0.857	0.242	0.404	0.476	0.761	0.476
		F1	**0.948**	0.923	0.760	0.938	0.900	0.161	0.250	0.160	0.556	0.095
Average	Yes	P	0.995	0.999	0.974	0.93	0.937	0.121	0.237	0.156	0.547	0.053
		R	0.934	0.894	0.628	0.953	0.864	0.29	0.399	0.419	0.79	0.471
		F1	**0.963**	0.944*	0.763*	0.941*	0.899*	0.17*	0.305*	0.218*	0.643*	0.096*

* FlexLog yields a significant higher F1-score than compared baseline.

5.2 RQ2: Data Efficiency on the ADFA-U Dataset

Table 9. Statistics of the sampled subsets of ADFA-U

$\mathcal{D}_{\#}$	adduser			hydraFTP			hydraSSH			java			web			meter		
	$\mathcal{U}_{\%}$	$\mathcal{U}^{+}_{\%}$	$\mathcal{U}^{-}_{\%}$	$\mathcal{U}_{\#}$	$\mathcal{U}^{+}_{\%}$	$\mathcal{U}^{-}_{\%}$	$\mathcal{U}_{\#}$	$\mathcal{U}^{+}_{\%}$	$\mathcal{U}^{-}_{\%}$	$\mathcal{U}_{\#}$	$\mathcal{U}^{+}_{\%}$	$\mathcal{U}^{-}_{\%}$	$\mathcal{U}_{\#}$	$\mathcal{U}^{+}_{\%}$	$\mathcal{U}^{-}_{\%}$	$\mathcal{U}_{\#}$	$\mathcal{U}^{+}_{\%}$	$\mathcal{U}^{-}_{\%}$
50	1.43	3.88	0.81	1.46	4.37	0.81	1.37	4.46	0.70	1.44	4.08	0.81	1.34	4.05	0.70	1.30	3.77	0.66
500	12.23	32.34	7.19	12.50	36.18	7.23	12.62	36.96	7.30	11.95	32.84	6.98	12.42	34.19	7.19	12.46	31.11	7.66
1000	21.78	53.65	13.80	21.45	57.16	13.53	21.76	58.03	13.82	21.55	54.08	13.84	21.30	53.80	13.49	21.81	52.41	13.91
1500	29.10	67.49	19.47	29.15	70.97	19.82	29.23	72.32	19.79	29.17	70.09	19.47	29.49	69.85	19.79	28.77	67.22	18.80
2000	35.48	78.38	24.72	35.04	82.69	24.41	35.96	82.32	25.81	36.13	79.90	25.71	35.30	79.57	24.66	35.11	76.43	24.43

RQ2 focuses on the data efficiency analysis of FlexLog and baselines. As mentioned in § 4.1, computational constraints preclude us from investigating the data efficiency on all the datasets. Hence, we focus on ADFA-U in this RQ since it includes six diverse configurations, enabling a robust evaluation under different real-world ULAD scenarios. As shown in RQ1, these diverse configurations make it the most challenging dataset in our experiments in terms of detection

effectiveness. Specifically, for each configuration, we randomly sample five training subsets (50, 500, 1000, 1500, and 2000 samples) to simulate different levels of data scarcity.

Table 9 reports statistics for the sampled subsets, including the percentage of unique log sequences ($\mathcal{U}_{\%}$), which is calculated by dividing the number of unique log sequences in each subset by the total number of unique log sequences in the full training dataset. This percentage represents the proportion of labeling effort required for each subset compared to full dataset annotation. The table further provides the percentages of unique log sequences for each class ($\mathcal{U}^{+}_{\%}$ and $\mathcal{U}^{-}_{\%}$). Notably, the smallest subset ($\mathcal{D}_{\#} = 50$) contains $< 1.5\%$ of unique log sequences, representing extreme data scarcity scenarios where the availability of labeled data is highly limited.

Figure 7 depicts the effectiveness of FlexLog and top baselines trained on the sampled subsets of ADFA-U. Less competitive baselines, such as PLELog, LogAnomaly, DeepLog, LogCluster, and PCA, are not illustrated in the figure for brevity. The dashed horizontal line in each figure represents the state-of-the-art results on each dataset, produced by LightAD trained with full datasets.

Table 10 highlights the improvement of FlexLog by calculating F1 score differences in percentage points between FlexLog and baselines. We also report Mann-Whitney U Test results for F1 comparisons across datasets. A "$*$" symbol is appended to the average F1 score if FlexLog significantly outperforms the baseline in terms of F1 score.

Overall, FlexLog consistently achieves superior performance in terms of average F1 scores compared to the baselines when trained with the same amount of labeled data. As shown in Table 10, the average F1-score improvements over the strongest baseline (i.e., FlexLog minus LightAD) for $\mathcal{D}_{\#} = 50$, 500, 1000, 1500, and 2000 are 2 pp, 13 pp, 10 pp, 8 pp, and 7 pp, respectively. Mann-Whitney U tests show the significance of all the improvements, except for the extreme data scarcity scenario with $\mathcal{D}_{\#} = 50$. Notably, even when compared to LightAD trained on the full dataset (represented by the dashed horizontal lines in all plots in Figure 7), FlexLog achieves higher or comparable performance with significantly less labeled data. In the remainder of this section, we elaborate on the performance of FlexLog and baselines under different data scarcity scenarios. Under extreme data scarcity ($\mathcal{D}_{\#} = 50$), we observe that all approaches exhibit poor performance in terms of F1 score, which are lower than 0.60, respectively, across all six configurations of ADFA-U. This limitation likely stems from insufficient training data diversity. As reported in the first row of Table 9, the 50 samples selected for each configuration of ADFA-U contain less than 1.46 % of total unique log sequences in full datasets. The percentages of unique anomalous log sequences are also low, with a maximum of 4.46 % ($\mathcal{D}_{\#} = 50$, the *java configuration*). Such low diversity and small size of the labeled datasets tend to be insufficient for the training of all approaches, including FlexLog and baselines.

Under less severe data scarcity ($\mathcal{D}_{\#} = 500, 1000, 1500, 2000$), all methods exhibit improved effectiveness compared to the extreme label scarcity scenario ($\mathcal{D}_{\#} = 50$). FlexLog outperforms all baselines across all six configurations in terms of F1 score. As reported in Table 10, the maximum percentage points against LightAD are observed at $\mathcal{D}_{\#} = 500$, reaching 13 pp on average. The average difference values diminish to 7 pp as data size increases to 2000, underscoring FlexLog's advantage with limited labeled data. Mann-Whitney U tests confirm the significance of all the differences ($\mathcal{D}_{\#} = 500, 1000, 1500, 2000$).

The answer to RQ2 is that FlexLog outperforms baselines in F1 scores under **varying data scarcity levels** of ADFA-U except at $\mathcal{D}_{\#} = 50$, where all methods perform poorly due to insufficient data.

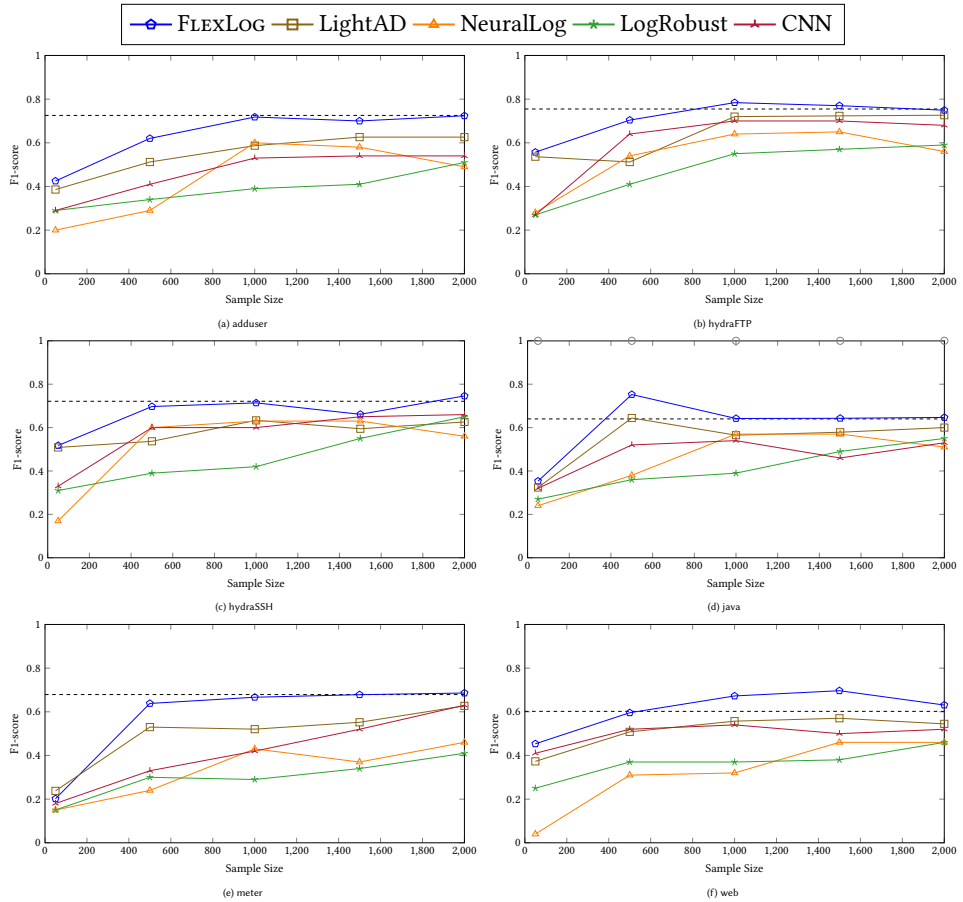


Fig. 7. F1-scores of FlexLog and top four baselines (LightAD, Neurallog, LogRobust, and CNN) trained on varying data scarcity level on six different ULAD configurations (adduser, hydraFTP, hydraSSH, meter, java, and web) of ADFA-U.

5.3 RQ3: Time and Memory Efficiency

5.3.1 Time Efficiency. Table 11 reports the training and inference time of FlexLog and the baselines across different datasets. Column "M" denotes the metric, where "T" represents the training time of full datasets (in seconds) and "I" represents the average inference time of one log sequence (in seconds). Specifically, we include FlexLog's base models—one LLM (Mistral) and three ML models (KNN, DT, and SLFN), which are introduced in § 4.3.2. The reported training and inference times represent the total time aggregated across these models (without considering parallel computation).

FlexLog requires substantially more training time than the baselines, with a maximum of 23,706 seconds (6 hours and 35 minutes) on the LOGEVOL-U Spark and SYNEVOL-U datasets. In contrast, the maximum training time of the baselines is recorded as only 1,550 seconds when training NeuralLog on the LOGEVOL-U Hadoop dataset. However, since training is a one-time cost, such a long training time is generally acceptable.

FlexLog's average inference time, per log sequence, remains below 1 second, ranging from 0.771 on SynHDFS-U to 0.988 on SYNEVOL-U. While this is significantly higher than traditional ML and

Table 10. F1 score differences (in percentage points) and statistical testing results when comparing FlexLog to baseline methods on the ADFA-U dataset.

Configuration	$\mathcal{D}_{\#}$	Supervised				Semi-S	Unsupervised			
		LightAD	NeuralLog	LogRobust	CNN	PLELog	LogAnomaly	DeepLog	LogCluster	PCA
adduser	50	4	22	13	13	3	10	10	23	19
	500	11	33	28	21	24	36	44	42	55
	1000	13	11	32	18	22	40	45	49	64
	1500	7	12	29	16	40	47	47	47	64
	2000	10	23	21	18	23	47	50	48	67
hydraFTP	50	2	27	28	20	20	22	22	26	38
	500	19	16	29	6	48	33	35	38	58
	1000	6	14	23	8	32	42	42	44	64
	1500	5	12	20	7	47	38	39	42	63
	2000	3	19	16	7	28	35	34	37	64
hydraSSH	50	1	35	21	19	6	17	19	22	28
	500	16	10	31	10	40	32	29	36	52
	1000	8	8	29	11	56	37	31	36	53
	1500	7	4	12	2	35	31	30	30	52
	2000	12	18	9	8	33	34	31	37	57
java	50	3	11	8	3	15	5	1	11	7
	500	11	37	39	23	51	41	47	48	48
	1000	6	7	25	10	50	29	35	36	47
	1500	6	7	15	18	46	29	35	35	48
	2000	5	14	10	12	33	29	36	36	51
meter	50	−4	5	5	2	7	0	1	5	10
	500	11	39	33	30	51	45	46	47	53
	1000	15	23	37	24	58	50	52	49	65
	1500	13	30	33	15	53	50	52	48	63
	2000	6	22	27	5	53	48	46	46	67
web	50	8	41	20	4	14	13	13	22	20
	500	9	29	23	8	17	37	36	34	47
	1000	12	35	30	13	35	40	40	41	60
	1500	13	24	32	20	45	43	42	44	58
	2000	9	17	17	11	42	32	31	37	53
average	50	2	23*	18*	10*	11*	11*	11*	18*	20*
	500	13*	27	30	16*	38*	37*	39*	41*	52*
	1000	10*	16*	29*	14*	42*	40*	41*	42*	59*
	1500	8*	15*	23*	13*	44*	40*	41*	41*	58*
	2000	7*	19*	17*	10*	35*	37*	38*	40*	60*

* FlexLog yields a significant higher F1-score than compared baseline.

DL baselines—some of which make one prediction in milliseconds—it reflects the inherent trade-off between model complexity and efficiency. Simpler models such as PCA and LogCluster achieve near instantaneous inference but suffer from low detection effectiveness, failing to detect critical anomalies (§ 5.1). The inference time of FlexLog is acceptable in user-oriented monitoring systems, where inference times on the order of seconds are generally tolerable. Examples include cloud service monitoring (e.g., AWS CloudWatch [19], Microsoft Azure Monitor [12]), where system logs

Table 11. Training (T) and Inference (I) time of FlexLog and the baselines (in seconds) on ADFA-U, LOGEVOL-U, SynHDFS-U, and SYNEVOL-U.

Config	M	Supervised									Semi-S	Unsupervised			
		FlexLog	Mistral	KNN	DT	SLFN	LightAD	NeuralLog	LogRobust	CNN	PLELog	LogAnomaly	DeepLog	LogCluster	PCA
ADFA-U															
adduser	T	16,161	16,160	2	<0.001	<0.001	5	922	221	271	285	276	184	4	0.017
	I	0.836	<0.001	<0.001	0.005	0.842	0.012	0.664	0.234	0.164	0.211	0.025	0.012	<0.001	≪ 0.001
hydraFTP	T	15,906	15,906	1	<0.001	<0.001	4	920	220	269	285	276	181	4	0.017
	I	0.818	0.813	<0.001	<0.001	0.005	0.014	0.635	0.229	0.165	0.211	0.024	0.011	<0.001	≪ 0.001
hydraSSH	T	14,147	14,146	1	<0.001	<0.001	5	923	220	269	285	275	183	4	0.017
	I	0.832	0.832	<0.001	<0.001	0.004	0.014	0.655	0.232	0.164	0.211	0.025	0.012	<0.001	≪ 0.001
java	T	13,933	13,932	1	<0.001	<0.001	6	921	220	270	285	275	180	4	0.017
	I	0.875	0.870	<0.001	<0.001	0.005	0.014	0.673	0.236	0.164	0.211	0.025	0.011	<0.001	≪ 0.001
web	T	14,079	14,078	1	<0.001	<0.001	5	918	220	270	285	276	181	4	0.017
	I	0.864	0.860	<0.001	<0.001	0.004	0.013	0.679	0.231	0.165	0.211	0.025	0.012	<0.001	≪ 0.001
meter	T	15,324	15,323	1	<0.001	<0.001	3	921	220	269	285	276	182	4	0.017
	I	0.888	0.885	<0.001	<0.001	0.003	0.014	0.682	0.238	0.165	0.211	0.025	0.012	<0.001	≪ 0.001
LOGEVOL-U															
Hadoop	T	4,031	4,031	6	<0.001	<0.001	30	1,550	178	316	48	1,340	1,020	21	0.180
	I	0.435	0.414	0.003	0.001	0.017	0.067	0.275	0.078	0.077	0.072	0.004	0.001	<0.001	≪ 0.001
Spark	T	23,706	23,706	*N/A*	≪0.001	≪0.001	0.2	712	232	136	220	944	514	9	0.015
	I	0.844	0.838	*N/A*	<0.001	0.006	0.038	0.564	0.082	0.072	0.006	0.007	0.003	< 0.001	≪ 0.001
SynHDFS-U															
average	T	13,587	13,583	4	<0.001	<0.001	22	1,260	293	355	42	976	1,110	20	0.067
	I	0.771	0.771	≪0.001	≪0.001	<0.001	0.005	0.259	0.008	0.016	0.007	0.004	0.002	< 0.001	≪ 0.001
SYNEVOL-U															
average	T	23,706	23,706	*N/A*	<0.001	<0.001	0.2	712	232	136	220	944	514	9	0.015
	I	0.988	0.979	*N/A*	<0.001	0.009	0.038	0.703	0.116	0.076	0.008	0.013	0.004	< 0.001	≪ 0.001

are analyzed to detect performance issues; IT infrastructure monitoring (e.g., Prometheus [80]), where server and network health metrics are updated periodically; and industrial IoT monitoring, where manufacturing systems and smart grids detect equipment failures. In these cases, inference time or response time of an anomaly detector within seconds still allows for timely interventions. In contrast, latency-sensitive domains, such as high-frequency trading systems, demand millisecond-level inference time, as decisions must be made within microseconds to capitalize on market fluctuation [2]. In such scenarios, the inference time of FlexLog, which is on the order of seconds, is not acceptable.

We remark that although FlexLog is not as efficient as traditional methods, its superior effectiveness justifies its application in scenarios where reliability is paramount. In anomaly detection, false negatives—undetected anomalies—can have far greater consequences than minor delays in inference time. For instance, in high-stakes environments like cybersecurity or critical infrastructure monitoring, failing to detect an anomaly or an intrusion attempt could lead to data breaches, financial losses, or system-wide failures. While simpler models offer faster inference, they often sacrifice effectiveness, leading to unacceptable numbers of false positives and false negatives.

To further explain the time cost of FlexLog, Table 11 shows the training and inference time of its base models, which include one LLM (Mistral) and three ML models (KNN, DT, and SLFN). We find that Mistral requires substantially more training time and average inference time per sequence compared to the three ML base models, accounting for the majority of time needed for FlexLog. Specifically, while Mistral's training and inference time remain under 7 hours and 1 s, respectively, all ML base models take less than 6 s for training and less than 0.001 s for inference per log sequence. This discrepancy is primarily due to Mistral's significantly large number of trainable parameters. As a result, Mistral has the most impact on the time efficiency of FlexLog compared to ML models.

However, with parallel computing and more powerful hardware, the efficiency of LLMs can easily be significantly improved, thereby enhancing the overall efficiency of FlexLog.

5.3.2 Memory Efficiency of FlexLog's Cache Mechanism. FlexLog leverages a caching mechanism to improve efficiency during inference by storing predictions for previous log sequences, as described in § 3.2. The cache improves FlexLog's time efficiency by reducing repetitive computation, although it introduces extra memory overhead. To evaluate the memory efficiency of this caching component, we report the memory usage, defined in § 4.2.4, during inference across our datasets. For ADFA-U, which contains the longest sequences with an average length of 461 templates, FlexLog consumes between 17.16 MB and 19.60 MB of memory. For LOGEVOL-U, which has the largest number of log sequences, the memory usage remains below 4 MB—specifically, 1.75 MB for Hadoop and 3.5 MB for Spark. For our synthetic datasets, memory usage also stays below 1 MB, averaging 576 kB and 2.88 MB for SynHDFS-U and SYNEVOL-U, respectively, across different injection ratios. Overall, this level of memory usage demonstrates the scalability potential of FlexLog 's caching mechanism, especially when balanced against the time savings presented as part of the answer to RQ4 (§ 5.4.1). Regarding scalability in systems with extremely long or highly diverse sequences, we note that the cache's *delete* function helps keep memory consumption under control (see § 3.2 for details). Further analysis of memory efficiency will depend on the specific memory resources available in the monitoring system during inference.

The answer to RQ3 is that while FlexLog is **not the most time-efficient** in ULAD inference, it processes each log sequence within **1** s **on average**. FlexLog's cache memory remains **below 4** MB for most datasets (up to 19.6 MB for ADFA-U), confirming its **memory efficiency**.

5.4 RQ4: Configuration Impact

In this section, we investigate the impact of FlexLog's different ablation configurations, such as excluding the cache mechanism (§ 5.4.1), excluding RAG (§ 5.4.2) and different choices of base models in ensemble learning (§ 5.4.3). Additionally, we investigate the impact of alternative LLMs in the configuration of FlexLog (§ 5.4.3) to highlight the advantages of using Mistral Small as the LLM component.

Table 12. FlexLog vs. FlexLog w/o cache — Comparisons of inference time per log sequence (in seconds) on ADFA-U, LOGEVOL-U, SYNEVOL-U, and SynHDFS-U

Config	ADFA-U							LOGEVOL-U			SYNEVOL-U	SynHDFS-U
	adduser	hydraFTP	hydraSSH	java	meter	web	average	Hadoop	Spark	average	average	average
FlexLog	0.842	0.818	0.832	0.875	0.864	0.888	**0.853**	0.435	0.844	**0.628**	**0.941**	**0.771**
FlexLog w/o cache	0.896	0.861	0.898	0.916	0.909	0.952	0.905	0.793	1.086	0.940	0.988	0.794
Difference (s)	-0.054	-0.043	-0.066	-0.041	-0.045	-0.064	-0.052*	-0.358	-0.242	-0.312*	-0.047*	-0.023

* FlexLog yields a significant lower inference time than FlexLog without cache.

5.4.1 Impact of Cache-empowered Inference. FlexLog maintains a cache C to avoid redundant predictions incurred by recurring log sequences, thus improving inference efficiency. To assess the impact of C on inference time, we compare FlexLog and FlexLog without cache (denoted by "w/o cache") in terms of inference time across four datasets: ADFA-U, LOGEVOL-U, SYNEVOL-U, and SynHDFS-U. The results of this comparison are reported in Table 12. Column "Config" indicates the subjects under comparison — FlexLog and FlexLog w/o cache; the following four columns "ADFA-U", "LOGEVOL-U", "SYNEVOL-U" and "SynHDFS-U" report the results of inference time per log sequence (in seconds) for each respective dataset. For the real-world datasets (ADFA-U and

LOGEVOL-U), we report inference time for each configuration. In contrast, for the synthetic datasets (SYNEVOL-U and SynHDFS-U), we provide only the average inference time across configurations. This is because different configurations of SYNEVOL-U and SynHDFS-U vary in the injection ratio of changes, which has minimal impact on inference time, leading to similar inference time across configurations. Hence, we report only the average inference time on synthesized datasets for brevity. The last row "Difference" indicates the difference between the inference time of FlexLog and that for FlexLog w/o cache. Similar to RQ1 (§ 5.1), we conducted Mann-Whitney U tests on each dataset to assess the statistical significance of the differences in inference times; the symbol "*" indicates the inference time of FlexLog is significantly lower than that of FlexLog w/o cache.

We observe that the introduction of the cache consistently reduces the inference time across all datasets. The reduction on two real-world datasets—ADFA-U and LOGEVOL-U—and SYNEVOL-U are more pronounced than SynHDFS-U, with reductions of 0.052, 0.312, and 0.047, respectively. Mann-Whitney U tests confirm these reductions are statistically significant, whereas the reduction on SynHDFS-U is smaller (0.023) and statistically not significant. However, as discussed in § 5.3, the practical impact of caching is limited in user-oriented monitoring systems, where inference times below 1 second are generally considered acceptable. While caching improves efficiency, the reduction in inference time might be too small to make a noticeable difference in such systems.

That said, we remark that the impact of caching previously seen log sequences depends on how frequently identical log sequences reappear in real-world systems. In our experiments, 6 %, 43 %, 2.7 %, and 2.2 % of log sequences appear more than once in the testing dataset of ADFA-U, LOGEVOL-U, SYNEVOL-U, and SynHDFS-U, respectively. Hence, the average reduction of inference time on each dataset, ordered from the greatest to the least, is as follows: LOGEVOL-U (−0.312 seconds), ADFA-U (−0.052 seconds), SYNEVOL-U (−0.047 seconds), and SynHDFS-U (−0.023 seconds). In practice, in many operational environments, such as distributed cloud computing frameworks (e.g., Hadoop [77] and Spark [14]) and security monitoring systems, certain types of log sequences, such as scheduled job reports and system diagnostics, occur repeatedly over time. In these cases, the cache mechanism of FlexLog can significantly improve inference efficiency, but its impact depends on the extent of redundant computations and the acceptable latency requirements of the system.

Table 13. FlexLog vs. FlexLog w/o RAG — Comparisons in terms of F1 score (in percentage points) on the ADFA-U dataset.

Config	adduser	hydraFTP	hydraSSH	java	meter	web	Average
FlexLog	71.8	78.4	72.3	64.2	68.2	67.2	**70.4**
FlexLog w/o RAG	68.8	70.5	64.8	62.2	65.9	64.1	66.0
Difference (pp)	3.0	7.9	7.5	2.0	2.3	3.1	4.4*

* FlexLog yields a significant higher F1-score than FlexLog without RAG.

5.4.2 Impact of RAG. FlexLog employs RAG to retrieve context for log sequences, providing relevant information that enriches the prompts. In our experiments, only ADFA-U has available contextual information, specifically Linux system call descriptions. Hence, RAG is only applied in the experiments on the ADFA-U dataset. To assess the impact of RAG, we compare the F1 scores of FlexLog and FlexLog without RAG (denoted by "w/o RAG") on ADFA-U. The results of this comparison are reported in Table 13. Column "Config" represents the subjects under comparison — FlexLog and FlexLog w/o RAG. Column "adduser", "hydraFP", "hydraSSH", "java", "meter", and "web" reports the F1 scores of six different configurations of ADFA-U; configuration details

are described in § 4.2.2. The last column "Average" indicates the average F1 score across all six configurations; the last row "Difference (pp)" reports the difference between the F1 scores of FlexLog and those of FlexLog w/o RAG. We performed a Mann-Whitney U test to assess the significance of the differences; however, testing on individual configurations was not feasible since we repeated the experiments on each configuration 5 times, which does not provide sufficient statistical power. Instead, a Mann-Whitney U test was run across all six configurations, increasing the test sample size to 30.

The results in Table 13 show that FlexLog outperforms FlexLog w/o RAG on all six configurations of ADFA-U, with differences ranging from 2 pp to 7.9 pp. On average, RAG improves the F1 score of ULAD from 0.660 to 0.704 (4.4 pp). A Mann-Whitney U test confirms this improvement is statistically significant. We conclude that the RAG component of FlexLog is effective in improving F1 scores on ADFA-U. This suggests that RAG could similarly improve performance on other datasets that have contextually rich log information. For example, in cloud computing platforms (e.g., AWS or Azure), where logs often include metadata such as instance IDs, resource types, or service configurations, RAG could enhance the effectiveness of ULAD by providing context that helps identify patterns or anomalies related to specific resources or services.

5.4.3 Impact of Base Model Choices in Ensemble Learning. FlexLog employs an ensemble of four base models for ULAD, leveraging diverse perspectives to enhance predictive effectiveness. As described in § 4.3, the ensemble in FlexLog includes one LLM (Mistral) and three ML models (KNN, DT, and SLFN), which were selected empirically due to their superior performance across different datasets. In this section, we present the results of different base model choices on ADFA-U, LOGEVOL-U, SynHDFS-U, and SYNEVOL-U, including ablation studies (e.g., removing individual base models and removing all ML base models) and replacing Mistral with alternative LLMs (Llama 3.1 and GPT-4o).

Ablation Study Table 14 reports ablation studies of FlexLog on all ULAD configurations. Column "Config" denotes the configuration of Flexlog; for instance, "w/o SLFP" represents FlexLog without SLFP base model, and "w/o ML" represents FlexLog without the three ML models (KNN, DT, and SLFN), resulting in standalone Mistral. To evaluate the individual contribution of each base model, we first assessed four configurations of FlexLog, each obtained by removing a single base model: FlexLog w/o Mistral, w/o KNN, w/o SLFN, and w/o DT. The results in Table 14 indicate that removing Mistral leads to the most significant drop in F1 scores, with a maximum reduction of 6.7 pp on ADFA-U adduser. Removing any of the ML models (KNN, DT, or SLFN) also results in reduced F1 scores across all configurations in all the datasets. To assess the statistical significance of these reductions, we conducted Mann-Whitney U tests on each dataset. The results confirm that the decreases in F1 scores are statistically significant in all cases, except for FlexLog w/o KNN, w/o DT, and w/o SLFN on LOGEVOL-U. This highlights the effectiveness of each ML base model in contributing to FlexLog 's overall effectiveness. Further, we assess the contribution of all ML models by excluding all three ML models from FlexLog, leaving only Mistral for predictions. This resulted in decreased F1 scores across most datasets, except for LOGEVOL-U Spark and SYNEVOL-U, where "w/o ML" outperformed FlexLog by 7 pp (96.2% – 89.2%) and 0.8 pp (97.9% – 97.1%), respectively. Mann-Whitney U tests show that FlexLog significantly outperforms "w/o ML" in terms of F1 score on ADFA-U and SynHDFS-U. The F1 score difference on SYNEVOL-U between FlexLog and "w/o ML" is statistically not significant, whereas on LOGEVOL-U Spark, "w/o ML" achieves a significantly higher F1 score than FlexLog. These results align with the findings from removing individual ML base models, further suggesting that the ML base models contribute negatively to FlexLog's performance on LOGEVOL-U Spark and , leading to "w/o ML" outperforming the ensemble.

A likely explanation for the better performance of FlexLog without any ML base models on LOGEVOL-U Spark and SYNEVOL-U is these datasets' extreme class imbalance. As detailed in § 4.2.1, LOGEVOL-U Spark and SYNEVOL-U share the same training dataset, sampled from LogEvol Spark 2, which is highly imbalanced: only 16 % of log sequences in the sampled training dataset are anomalous, compared to 50 % in ADFA-U and LOGEVOL-U Hadoop, and 57 % in SynHDFS-U. It is well known that traditional ML models, such as KNN, SLFN, and DT, struggle with highly imbalanced datasets [46]. Since FlexLog integrates these ML models, its performance is negatively affected.

To mitigate data imbalance, we experimented with both down-sampling and over-sampling techniques [68]. Down-sampling by reducing normal logs to match the number of anomalous logs led to a decrease of 5 pp in F1 score. For over-sampling, we applied several standard strategies, including duplicating anomalous logs, adding small perturbations to representation vectors, and using the Synthetic Minority Oversampling Technique (SMOTE) [4] on representation vectors. However, none of these approaches improved the effectiveness of the ML models. The key challenge lies in the representation of log sequences as count vectors. Unlike continuous feature spaces where interpolation can generate plausible synthetic samples, count vector representations encode categorical relationships, making common over-sampling techniques such as SMOTE [5] prone to producing unrealistic or noisy data. In contrast, standalone Mistral ("w/o ML") remains robust in predictive effectiveness, leveraging its pretraining on vast and diverse corpora to mitigate data imbalance. Based on these findings, we *recommend* using standalone Mistral instead of the full FlexLog when dealing with extremely imbalanced training datasets, eliminating the negative effect of poorly trained ML models.

For future improvements, more advanced data augmentation techniques, such as Generative Adversarial Networks (GANs), could be used to generate synthetic log sequences directly, rather than modifying their count vector representations. This avoids the issue of unrealistic or noisy data caused by over-sampling in a discrete feature space, making the synthetic data more useful for training ML models on imbalanced datasets like LOGEVOL-U Spark. As a result, this could potentially improve FlexLog 's overall performance.

Alternative LLMs. Table 15 reports the F1-score of FlexLog with three LLM choices. Column "Config" represents different configurations of FlexLog, wherein Mistral is replaced by Llama 3.1 8B ("*Mistral* → *Llama*") and GPT-4o ("*Mistral* → *GPT*"). Column "Source" indicates whether the employed LLM is open-source or closed-source. Experimental results indicate that GPT-4o and Mistral achieve comparable effectiveness, both consistently outperforming Llama across all configurations and datasets. Mann-Whitney U tests at the dataset level confirm that the F1 score differences between FlexLog and " *Mistral* → *GPT*" are not significant across all four datasets. However, FlexLog significantly outperforms "*Mistral* → *Llama*" on ADFA-U, LOGEVOL-U, and SynHDFS-U, while the difference on SYNEVOL-U is not statistically significant. These findings underscore the effectiveness of Mistral for ULAD, demonstrating performance on par with the closed-source GPT-4o while avoiding the financial cost associated with API invoking. Thus, we recommend Mistral as a cost-effective yet competitive base model for FlexLog.

The answer to RQ4 is that the full FlexLog configuration—combining **cache**, **RAG**, and an ensemble of **KNN, DT, SLFN, and Mistral**—performs best, with each component improving efficiency or effectiveness. Among LLMs, **Mistral** is a cost-effective choice, matching the performance of GPT-4o while outperforming Llama.

5.5 Discussion

In this section, to guide AIOps engineers, we highlight the implications of our study for ULAD.

Table 14. Ablation studies of FlexLog on all unstable datasets.

Config	ADFA-U							LOGEVOL-U			SynHDFS-U	SYNEVOL-U
	adduser	hydraFTP	hydraSSH	java	meter	web	average	Hadoop	Spark	average	average	average
FlexLog	**0.718**	**0.784**	**0.723**	**0.642**	**0.682**	**0.672**	**0.704**	0.982	0.892	0.937	**0.972**	0.971
w/o Mistral	0.645	0.769	0.692	0.628	0.639	0.616	0.664*	0.975	0.841	0.908*	0.939*	0.936*
w/o KNN	0.683	0.768	0.654	0.621	0.651	0.579	0.659*	0.980	*N/A*	0.980	0.945*	*N/A*
w/o DT	0.667	0.749	0.690	0.640	0.654	0.623	0.670*	0.973	0.875	0.924	0.948*	0.959*
w/o SLFN	0.677	0.691	0.613	0.641	0.647	0.556	0.637*	0.978	0.871	0.924	0.934*	0.948*
w/o ML	0.579	0.591	0.630	0.628	0.674	0.571	0.612*	**0.998**	**0.962**	**0.980**†	0.928*	**0.979**

* FlexLog yields a significant higher F1-score than the ablation configuration.

† FlexLog yields a significant lower F1-score than the ablation configuration.

N/A Not applicable. KNN is excluded from FlexLog on the SynHDFS-U and LOGEVOL-U datasets (see § 4.3.2)

Table 15. F1 scores of using alternative LLMs in FlexLog.

Config	Source	ADFA-U							LOGEVOL-U			SynHDFS-U	SYNEVOL-U
		adduser	hydraFTP	hydraSSH	java	meter	web	average	Hadoop	Spark	average	average	average
FlexLog	open	0.718	0.784	**0.723**	0.642	0.682	0.672	0.704	0.982	0.892	**0.937**	0.972	**0.971**
Mistral → Llama	open	0.679	0.743	0.707	0.593	0.669	0.591	0.664*	0.941	0.850	0.895*	0.949*	0.970
Mistral → GPT	closed	**0.721**	**0.786**	0.718	**0.648**	**0.690**	**0.696**	**0.710**	**0.981**	0.886	0.933	**0.976**	**0.971**

* FlexLog yields a significant higher F1 score compared to using the alternative LLM.

5.5.1 Token Consumption of LLMs. In the early times of leveraging attentive language models, language models accepted limited input tokens, such as a maximum of 512 input tokens for BERT [18], making it challenging to apply them to long log sequences. However, as language models expanded into LLMs, the maximum input token limit also increased, both for open-source and closed-source models. Table 16 reports the input token limits of the LLMs used in our work and compares them with the maximum token consumption observed for each dataset. Column "Model" denotes LLMs used in our experiments. Column "Source" indicates whether the LLM is open-sourced or closed-source. Column "Input Token Limit" reports the input token limit specific to each LLM. Column "Maximum Input Token" denotes the maximum token consumption of FlexLog's prompts on each dataset. We note that these values vary across LLMs due to variations in their tokenization processes. Our results show that FlexLog's token consumption remains well within input token limits for all LLMs, indicating that the challenge of input token limits has been alleviated and even eliminated for our datasets. The highest usage is only 37.7 % (24,735 out of the maximum limit of 65,536 tokens), with prompts used by GPT-4o for ADFA-U. Notably, in our experiments, we have included various log datasets with log sequences as long as 4,474 templates (see Table 1), and yet LLMs effectively accommodate them. Moreover, recent advances in LLMs can potentially extend the token limit to one million tokens [78, 92], making it no longer a hard limitation for using LLMs.

5.5.2 Error Analysis. In RQ4, we observed that removing any base model from FlexLog generally decreases effectiveness. To better understand the role of the LLM base model (i.e., Mistral) and ML base models (i.e., KNN, DT, and SLFN), in this section, we analyze the log sequences that FlexLog correctly classify but mis-classify when either Mistral or ML models are removed. We denote these mis-classified log sequences as $\texttt{MIS}_{w/o\ Mistral}$ and $\texttt{MIS}_{w/o\ ML}$, respectively. We investigate, for each dataset, whether $\texttt{MIS}_{w/o\ Mistral}$ and $\texttt{MIS}_{w/o\ ML}$ differ in various characteristics to highlight the unique contribution of LLM and ML to the effectiveness of FlexLog. Specifically, we focus

Table 16. Overview of token consumption of ADFA-U, LOGEVOL-U, SYNEVOL-U, and SynHDFS-U on open-source and closed-source LLMs

Model	Accessibility	Input Token Limit	Maximum Input Token			
			ADFA-U	LOGEVOL-U	SynHDFS-U	SYNEVOL-U
Mistral Small	open	128,000	25,848	28,934	1,243	16,840
Llama 3.1 8B	open	128,000	21,371	24,798	862	12,533
GPT-4o	closed	65,536	21,383	24,735	860	12,479

on the unseen log templates in $\texttt{MIS}_{w/o\ Mistral}$ and $\texttt{MIS}_{w/o\ ML}$ because unseen log templates pose a key challenge in ULAD. Traditional anomaly detectors often rely on predefined patterns and struggle with generalization; in contrast, FlexLog, by means of the integration of LLM, may be better at detecting anomalies involving novel log templates. By analyzing the misclassified cases in $\texttt{MIS}_{w/o\ Mistral}$ and $\texttt{MIS}_{w/o\ ML}$, we can determine whether the LLM (Mistral) or ML models (KNN, DT, and SLFN) contribute more to handling novel log templates, helping us understand their complementary strengths in FlexLog.

Table 17 reports the error analysis results across all four datasets: ADFA-U, LOGEVOL-U, SYNEVOL-U, and SynHDFS-U. Column "Data" specifies the dataset; Column "Config" reports the configuration of each dataset; Column "Condition" shows the subject of the statistics, including the testing dataset, $\texttt{MIS}_{w/o\ Mistral}$, and $\texttt{MIS}_{w/o\ ML}$. Column "# Sequences" reports the number of sequences; column "Sequence Length" indicates the average, minimum, and maximum length of the sequences, denoted as "avg", "min", and "max", respectively. Column "% Anomaly" reports the percentage of anomalous log sequences, and column "% Unseen Template" denotes the percentages of log sequences that have at least one unseen log template. Overall, for each dataset, $\texttt{MIS}_{w/o\ Mistral}$ and $\texttt{MIS}_{w/o\ ML}$ demonstrate different characteristics in terms of log sequence length, percentage of anomaly, and percentage of unseen templates, highlighting how Flexlog's base models complement each other in an effective prediction.

In terms of unseen templates, in LOGEVOL-U Hadoop and SynHDFS-U, we did not observe any in either $\texttt{MIS}_{w/o\ ML}$ or $\texttt{MIS}_{w/o\ Mistral}$. In ADFA-U, $\texttt{MIS}_{w/o\ Mistral}$ contains 7.38 % unseen log templates, much higher than that of $\texttt{MIS}_{w/o\ ML}$ (1.89 %). Similarly, in LOGEVOL-U Spark, the percentage of unseen log templates in $\texttt{MIS}_{w/o\ Mistral}$ reaches 33.33 %, while $\texttt{MIS}_{w/o\ ML}$ contains no unseen template-related mis-classifications. In SYNEVOL-U, 11.76 % of misclassifications in $\texttt{MIS}_{w/o\ Mistral}$ are related to unseen templates, higher than that in $\texttt{MIS}_{w/o\ ML}$ (9.28 %). Overall, we observe a higher percentage of unseen templates in $\texttt{MIS}_{w/o\ Mistral}$ than $\texttt{MIS}_{w/o\ ML}$ across most datasets. The higher percentage of unseen log templates in $\texttt{MIS}_{w/o\ Mistral}$, across several datasets, indicates that Mistral plays a crucial role in handling unseen log templates. This highlights LLM's adaptability in recognizing new templates, making it particularly valuable for ULAD, where logs are unstable due to environment and system evolution. In contrast, perhaps unsurprisingly, ML models appear to rely more on established patterns, struggling with new templates.

5.6 Threats to Validity

5.6.1 Internal Validity. **Data Leakage from LLMs.** One potential threat to internal validity is data leakage, where test data may inadvertently overlap with training data. Although we performed de-duplication across all test datasets, the risk of data leakage cannot be entirely eliminated, as some test data may have been included in the pretraining corpus of LLMs. This could lead to inflated effectiveness. To mitigate this risk, we evaluated FlexLog not only on publicly available

Table 17. Overview of Error Analysis of ADFA-U, LOGEVOL-U, SYNEVOL-U, and SynHDFS-U when Mistral or ML models are removed from FlexLog

Data	Config	Condition	# Sequence	Sequence Length			% Anomaly	% Unseen Template
				avg	min	max		
ADFA-U	all ULAD	Testing Dataset	5,146	523.98	75	4,494	14.34	3.54
		$\texttt{MIS}_{w/o\ Mistral}$	149	476.74	82	2,513	0	7.38
		$\texttt{MIS}_{w/o\ ML}$	264	610.48	80	3,089	11.74	1.89
LOGEVOL-U	Hadoop ULAD	Testing Dataset	5,329	13.41	1	50	9.8	74.37
		$\texttt{MIS}_{w/o\ Mistral}$	5	41.8	9	50	0	0
		$\texttt{MIS}_{w/o\ ML}$	0	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
	Spark ULAD	Testing Dataset	4,045	81.70	1	1,977	1.78	38.07
		$\texttt{MIS}_{w/o\ Mistral}$	6	15.33	4	35	100	33.33
		$\texttt{MIS}_{w/o\ ML}$	1	26	26	26	100	0
	all ULAD	Testing Dataset	9,374	42.88	1	1,977	6.34	58.70
		$\texttt{MIS}_{w/o\ Mistral}$	11	29.83	4	50	54.54	18.18
		$\texttt{MIS}_{w/o\ ML}$	1	26	26	26	100	0
SynHDFS-U	all ULAD	Testing Dataset	3,744	26.91	10	57	22.22	0.43
		$\texttt{MIS}_{w/o\ Mistral}$	24	40.08	35	48	0	0
		$\texttt{MIS}_{w/o\ ML}$	92	32.43	19	52	5.43	0
SYNEVOL-U	all ULAD	Testing Dataset	15,406	151.07	1	1,022	2.96	91.3
		$\texttt{MIS}_{w/o\ Mistral}$	34	135.03	6	323	100	11.76
		$\texttt{MIS}_{w/o\ ML}$	23	39.04	4	177	0	9.28

N/A Not applicable as no prediction errors were observed in this configuration

datasets (ADFA, LOGEVOL, and SYNEVOL-U), but also on the SynHDFS-U dataset, which we synthesized ourselves. In addition, the cut-off date of Mistral Small is August 2023, and LOGEVOL and SYNEVOL-U were introduced after this date. Hence, these datasets cannot be part of its pretraining corpus, ensuring a more reliable assessment of FlexLog 's effectiveness.

Selection of Training Dataset Sizes. Another threat to internal validity arises from the selection of training dataset sizes ($\mathcal{D}_{\#}$) for evaluating data efficiency. Since $\mathcal{D}_{\#}$ directly affects the performance of FlexLog and the baselines, an exhaustive evaluation across all possible values is infeasible. To address this limitation, we systematically experimented with multiple $\mathcal{D}_{\#}$ values, as detailed in § 5.2, ranging from 50 to 2,000. This range allows us to provide a comprehensive analysis of data efficiency while remaining within our computational constraints.

Selection of FlexLog's Base Models. Since FlexLog is an ensemble learning-based approach, its effectiveness is significantly influenced by the selection of base models. While it is impractical to evaluate all possible model combinations, we carefully selected representative base models from the literature on log-based anomaly detection, including KNN, DT, SLFN, LightAD [97], NeuralLog [51], CNN [62], LLaMA 3.1, and GPT-4o. Through extensive empirical evaluation, we tested multiple model combinations and ultimately selected KNN, DT, MLP, and Mistral due to their consistently high effectiveness and robustness across both real-world and synthesized datasets.

Synthetic Datasets. The synthetic datasets in our experiments are generated by injecting different levels of instability. One potential threat to internal validity lies in the realism of this injected

instability and the validity of labels after injections. To mitigate this threat, we adopted datasets synthesized by experts and applied well-established injection strategies from the literature. Specifically, the SYNEVOL-U dataset, proposed by Huo et al. [39], was annotated by two experienced Spark developers and carefully reviewed after annotation. For SynHDFS-U datasets, which we constructed following the SYNEVOL-U approach, we applied only sequence-level changes that are unlikely to affect the overall label of a log sequence. To guide this process, we leveraged the decision tree analysis [89] of the original dataset to identify low-impact templates. These precautions help maintain the reliability of the labels, despite the synthetic nature of the data.

Consistency of Labels. For faster inference, FlexLog's cache mechanism reuses the prediction for previously seen, identical log sequences. However, the assumption that identical log sequences always correspond to the same label may not hold in extreme log evolution cases, particularly in security-critical domains. This assumption is nevertheless valid for all of our datasets, where each unique log sequence is consistently labeled. To further mitigate this risk, the *update* function in FlexLog's cache enables label revisions by system administrators if changes are observed over time.

Cascading Effect of Parsing Errors. The use of log parsing introduces a potential risk of parsing errors, which could negatively affect the effectiveness of FlexLog. However, prior work by Khan et al. [47] reports no strong correlation between log parsing accuracy and anomaly detection accuracy, suggesting that minor parsing errors do not necessarily degrade the effectiveness of FlexLog. Our evaluation shows that FlexLog, as a parsing-based approach, achieves significantly higher effectiveness than NeuralLog and LightAD, both of which operate on raw logs. Specifically, LightAD uses some of the same base models as FlexLog but applies them without log parsing. In FlexLog, this risk is further mitigated through two design factors. First, we adopt widely used and reliable parsers to reduce the likelihood of significant parsing errors (see § 4.3.1). Second, the architecture limits error propagation: predictions are aggregated via majority voting across multiple base models, preventing a single erroneous output from dominating the final decision. Additionally, the fine-tuned LLM in FlexLog leverages contextual information, making it resilient to potential inconsistencies introduced by parsing.

5.6.2 Conclusion Validity. It is widely acknowledged that LLMs often produce non-deterministic responses even when provided with identical prompts, posing a potential threat to the conclusion validity of FlexLog. To address this challenge, as detailed in § 4.3, we configured the LLMs to generate responses with minimal randomness, achieved by setting the temperature parameter to 0. To further reduce the influence of randomness from our evaluation results, as discussed in § 4.3, we ran each experiment configuration five times and calculated the average value as the final result. We also conducted Mann-Whitney U test to assess the significance of effectiveness and efficiency differences between FlexLog and baselines. To ensure sufficient statistical power, we performed statistical testing at the dataset level rather than the configuration level, since each configuration only has five samples. Aggregation at the dataset level ensures each test includes at least 10 samples.

5.6.3 External Validity. A potential threat to external validity is the impact of highly unstable logs on anomaly detection performance. While FlexLog achieves state-of-the-art effectiveness, its performance, like that of other methods, may degrade when log instability is extreme. For example, the introduction of YARN for job management in Hadoop 2 resulted in substantial architectural modifications, leading to significant changes in its logs [83]. Such drastic shifts pose challenges for all existing methods, potentially limiting their applicability in highly unstable logging environments. To address this concern, we evaluate FlexLog and the baselines across diverse datasets that capture

two primary sources of log instability: software evolution (LOGEVOL-U) and environment change (ADFA-U). Additionally, we conduct experiments on two synthesized datasets (SynHDFS-U and SYNEVOL-U) that simulate instability levels ranging from 0 % to 30 %. Our findings indicate that while all methods experience performance degradation as instability increases, FlexLog remains robust in terms of precision, recall and F1 score. With a minimum F1 score of 0.948 (30 % template level injections on SYNEVOL-U), FlexLog consistently outperforms all the baselines, demonstrating greater robustness in handling highly unstable logs. Nevertheless, further evaluation on a broader range of real-world systems is necessary to fully assess the limitations of its applicability.

6 RELATED WORK

6.1 Anomaly Detection on Unstable Logs

Log-based anomaly detection has been extensively studied in the literature to enhance the dependability of software-intensive systems [50, 53, 97]. However, only a few studies have investigated anomaly detection on unstable logs [38, 39, 56, 102], a common situation in practice. Zhang et al. [102] first identified such a challenge and proposed LogRobust (see § 4.2.3) to leverage an attention-based Bi-LSTM as an anomaly detector. They also created a new unstable log dataset called Synthetic HDFS to evaluate the effectiveness and robustness of LogRobust. This inspired a number of follow-up works, including supervised [38, 56] and unsupervised approaches [39].

Supervised approaches like HitAnomaly [38] and SwissLog [56] require training with a labeled dataset, encompassing both normal and anomalous data. SwissLog adopts the same architecture (i.e., Bi-LSTM) as LogRobust and aims to further improve it by incorporating time embeddings and Bert-based semantic embeddings. HitAnomaly, however, leverages a much larger model based on a hierarchical transformer architecture. The high complexity of the HitAnomaly model allows it to tackle not only the static parts of log messages but also dynamic parts, such as numerical values that have been masked in the log templates. Experiment results demonstrate the superiority of HitAnomaly on stable logs compared to LogRobust, while showing a robust performance for small injection ratios (under 20%) in unstable logs and being outperformed by LogRobust from 20% to 30%.

Huo et al. [39] proposed EvLog, an unsupervised approach leveraging a multi-level semantics extractor and attention mechanism to identify anomalous log messages in unstable logs. Unlike FlexLog, EvLog is a parser-free method to combat potential parsing errors in a dataset. However, by avoiding log parsing, EvLog may face challenges in generalizing to datasets with diverse or domain-specific log formats, as it relies solely on semantic extraction without leveraging structured context. As part of the EvLog study, they also introduced two log datasets: LOGEVOL and SYNEVOL (referred to as SYNEVOL-U in our paper), which serve as valuable benchmarks for ULAD research, including our work.

To summarize, compared to existing ULAD approaches, FlexLog: 1) leverages the synergy of ML models and LLMs through ensemble learning 2) requires significantly less training data, reduces labeling cost, 3) achieves state-of-the-art effectiveness across all datasets, outperforming baselines in terms of F1 scores consistently while being trained on limited labeled data.

6.2 Application of LLMs to Log Analysis

Over the past few years, LLMs have been widely adopted on different log-related software engineering tasks to enhance effectiveness and generalizability, including anomaly detection and log parsing [59, 63, 88].

Anomaly detection. The application of LLMs in the field of anomaly detection started by leveraging BERT [18] to capture contextual information of logs with semantic-based representations.

LogBERT [25] utilizes BERT to learn the semantics of normal log messages and predicts an anomaly where the representation of log messages of an input sequence deviates from the distribution of normal log sequences. Le and Zhang [51] proposed NeuralLog (discussed in § 4.2.3). Han et al. [27] introduced LogGPT, which leverages reinforcement learning to fine-tune GPT-2 for anomaly detection. More recently, Liu et al. [59] proposed LogPrompt, which adopts LLMs such as GPT-3 and Vicuna [8] for online log parsing and anomaly detection via in-context learning; we discussed the reasons for not considering LogPrompt as baseline in § 4.2.3. He et al. [29] proposed LLMeLog, which leverages a BERT model fine-tuned with log sequences enriched with contextual information that was retrieved by GPT-3.5. Inspired by LLMeLog, we further explored RAG with various LLMs, including closed-source and open-source LLMs. Additionally, we equipped LLMs with a cache mechanism and ensemble learning to enhance their effectiveness and data efficiency. Note that we did not consider LLMeLog as a baseline in this work due to the unavailability of their replication package.

Log Parsing. Le and Zhang [52] explored the in-context learning of ChatGPT [73] on log parsing and achieved promising results with zero-shot and few-shot prompts. Xu et al. [88] proposed DivLog, another few-shot, in-context learning method that constructs prompts with five labeled examples for each target log template. DivLog explicitly optimizes the diversity of included examples using the Determinantal Point Process (DPP) [6], reducing the potential biases in the examples by maximizing sample diversity. Jiang et al. [43] proposed a novel parser called LILAC, equipping LLMs with an adaptive cache to reduce the LLM query times and, consequently, the efficiency. Recently, Pei et al. [74] introduced a self-evolutionary LLM-based parser, which identifies new log templates by grouping history log messages. Fewer studies focus on fine-tuning LLMs for log parsing. Le and Zhang [54] introduced LogPPT to fine-tune RoBERTa for log parsing. In addition, an adaptive random sampling strategy was designed to select a small yet diverse training dataset. Ma et al. [63] compared in-context learning and fine-tuning using open-source LLMs such as Flan-T5 [11] and LLaMA [79] on log parsing. Zhi et al. [104] introduced YALP, which leverages the capabilities of ChatGPT (gpt-3.5-turbo) in conjunction with traditional methods — Longest Common Subsequence,without incorporating user labeling (zero-shot learning). Zhong et al. [105] proposed LogParser-LLM, which essentially blends a prefix tree and an LLM-based template extractor. This extractor parses log messages with different LLMs, including GPT-3.5-turbo, GPT-4, and Llama-2-13B, in either ICL or fine-tuning manner; the highest results were obtained using GPT-4 with ICL. Xiao et al. [87] introduced LogBatcher, which is a cost-effective LLM-based log parser based on GPT-3.5-Turbo. Similar to YALP, they control the cost of using closed-source LLM by storing inferred messages in a basic cache. Moreover, it does not require any training by prompting the LLM with a group of high-diversity log messages to ensure that the LLM understands the diversity of the dataset in a zero-shot manner. Overall, the results of the above works align with our findings: the vast pretrained knowledge of LLMs enables data efficiency and robustness on unseen log templates.

7 CONCLUSION AND FUTURE WORK

This paper proposed a novel approach, FlexLog, for anomaly detection on unstable logs (ULAD), exploiting the synergy between Large Language Models (LLMs) and Machine Learning (ML) models via ensemble learning. FlexLog incorporates four base models, one LLM (Mistral), and three ML models (KNN, DT, and SLFN), which are trained on limited stable logs, reducing the usage of labeled data significantly. To classify unstable logs, FlexLog first processes unstable logs into log sequences through log parsing and partitioning. Then, FlexLog employs Retrieval-Augmented Generation (RAG) to fetch relevant information (if available) for these sequences, constructing context-enriched prompts. The fine-tuned LLM processes these prompts, while ML models use the

log sequences directly for prediction. Finally, FlexLog combines the predictions of all the base models using majority voting to produce the final classification.

Our extensive experiments on two real-world and two synthesized datasets show that FlexLog achieves state-of-the-art effectiveness on all datasets while reducing the usage of labeled data by 62.87 pp to 78.43 pp, respectively. Further experiments on ADFA-U with varying limited training data size demonstrate that FlexLog maintains robust effectiveness under varying levels of data scarcity, except the extreme data scarcity scenario ($\mathcal{D}_{\#} = 50$), where all methods exhibit poor performance due to insufficient labeled data. FlexLog outperforms the top baseline in terms of F1 by 13 pp when the training dataset contains only 500 samples. However, experiments assessing time efficiency show that FlexLog trades off some time efficiency for this effectiveness but still manages to keep the inference time below one second per log sequence. This suggests FlexLog is applicable for most systems, except those with stringent latency requirements, such as high-frequency trading systems. Furthermore, in terms of memory efficiency, FlexLog 's cache memory remains **below 4** MB for most datasets (up to 19.6 MB for ADFA-U) confirming its **memory efficiency**. Finally, we conducted ablation studies on individual components of FlexLog, as well as evaluating alternative LLM choices. We confirmed the significant contributions of the cache-based inference, RAG, and ensemble learning with Mistral as LLM and KNN, DT, and SLFN as ML models.

In the future, we plan to further enhance the effectiveness of FlexLog by exploring more powerful open-source LLMs as base models, such as DeepSeek R1 [15], along with more advanced prompting techniques such as agent-based prompting [84]. Additionally, we wish to investigate different ensemble learning techniques that dynamically decide the optimal ensemble composition for a given log sequence. To enhance the generalizability of our caching mechanism, we plan to incorporate similarity-based retrieval in the future, allowing the model to infer labels from similar seen log sequences instead of identical ones. Furthermore, to address the dataset imbalance observed in certain datasets, we aim to explore techniques like Generative Adversarial Networks (GANs) for data augmentation, potentially improving the performance of ML-based models within the ensemble.

ACKNOWLEDGMENTS

This work was partly supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), through the Canada Research Chairs and discovery programs, the Research Ireland grant 13/RC/2094-2, the Luxembourg National Research Fund (FNR), grant reference C22/IS/17373407/LOGODOR. The experiments conducted in this work were enabled in part by the computation support provided by the Digital Research Alliance of Canada.[5]

REFERENCES

[1] Andrea Arcuri and Lionel Briand. 2011. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Proceedings of the 33rd International Conference on Software Engineering* (Waikiki, Honolulu, HI, USA) *(ICSE '11)*. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/1985793.1985795

[2] Halima Oluwabunmi Bello, Adebimpe Bolatito Ige, and Maxwell Nana Ameyaw. 2024. Deep learning in high-frequency trading: conceptual challenges and solutions for real-time fraud detection. *World Journal of Advanced Engineering Technology and Sciences* 12, 02 (2024), 035–046.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[4] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16 (2002), 321–357.

[5] https://alliancecan.ca/en

[5] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. 2002. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research* 16 (June 2002), 321–357. https://doi.org/10.1613/jair.953

[6] Laming Chen, Guoxin Zhang, and Hanning Zhou. 2018. Fast Greedy MAP Inference for Determinantal Point Process to Improve Recommendation Diversity. arXiv:1709.05135 [cs.IR] https://arxiv.org/abs/1709.05135

[7] Mike Y. Chen, Alice X. Zheng, Jesper Lloyd, Michael I. Jordan, and Eric Brewer. 2004. Failure diagnosis using decision trees. *International Conference on Autonomic Computing, 2004. Proceedings.* (2004), 36–43. https://api.semanticscholar.org/CorpusID:56849250

[8] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality. https://lmsys.org/blog/2023-03-30-vicuna/

[9] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, Alessandro Moschitti, Bo Pang, and Walter Daelemans (Eds.). ACL, 1724–1734. https://doi.org/10.3115/V1/D14-1179

[10] Guohao Chu, Jiaqi Wang, Qi Qi, Haiyang Sun, Shengtao Tao, and Jun Liao. 2021. Prefix-Graph: A Versatile Log Parsing Approach Merging Prefix Tree with Probabilistic Graph. In *Proceedings of the 37th International Conference on Data Engineering (ICDE)*. IEEE, Virtual Event, 2411–2422. https://ieeexplore.ieee.org/document/9458609

[11] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling Instruction-Finetuned Language Models. arXiv:2210.11416 [cs.LG]

[12] Michael Collier and Robin Shahan. 2015. *Microsoft azure essentials-fundamentals of azure.* Microsoft Press.

[13] Gideon Creech and Jiankun Hu. 2013. Generation of a new IDS test dataset: Time to retire the KDD collection. In *2013 IEEE Wireless Communications and Networking Conference (WCNC)*. 4487–4492. https://doi.org/10.1109/WCNC.2013.6555301

[14] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* 51, 1 (2008), 107–113. https://doi.org/10.1145/1327452.1327492

[15] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, and Xiao Bi. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. arXiv:2501.12948 [cs.CL] https://arxiv.org/abs/2501.12948

[16] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American society for information science* 41, 6 (1990), 391–407.

[17] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient Finetuning of Quantized LLMs. arXiv:2305.14314 [cs.LG] https://arxiv.org/abs/2305.14314

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. , 4171–4186 pages. https://doi.org/10.18653/v1/n19-1423

[19] Ewere Diagboya. 2021. *Infrastructure Monitoring with Amazon CloudWatch: Effectively monitor your AWS infrastructure to optimize resource allocation, detect anomalies, and set automated actions.* Packt Publishing Ltd.

[20] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Tianyu Liu, et al. 2022. A survey on in-context learning. *arXiv preprint arXiv:2301.00234* (2022).

[21] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. DeepLog: Anomaly detection and diagnosis from system logs through deep learning. , 1285–1298 pages. https://doi.org/10.1145/3133956.3134015

[22] Evelyn Fix and Joseph L. Hodges. 1989. Discriminatory Analysis - Nonparametric Discrimination: Consistency Properties. *International Statistical Review* 57 (1989), 238. https://api.semanticscholar.org/CorpusID:120323383

[23] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, and Aiesha Letman. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783

[24] Gongde Guo, Hui Wang, David Bell, Yaxin Bi, and Kieran Greer. 2003. KNN Model-Based Approach in Classification. In *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, Robert Meersman, Zahir Tari, and Douglas C. Schmidt (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 986–996.

[25] Haixuan Guo, Shuhan Yuan, and Xintao Wu. 2021. LogBERT: Log Anomaly Detection via BERT. In *2021 International Joint Conference on Neural Networks (IJCNN)*. 1–8. https://doi.org/10.1109/IJCNN52387.2021.9534113

[26] Fatemeh Hadadi, Qinghua Xu, Domenico Bianculli, and Lionel Briand. 2025. Replication Package. https://doi.org/10.6084/m9.figshare.25988170

[27] Xiao Han, Shuhan Yuan, and Mohamed Trabelsi. 2023. LogGPT: Log Anomaly Detection via GPT. In *2023 IEEE International Conference on Big Data (BigData)*. 1117–1122. https://doi.org/10.1109/BigData59044.2023.10386543

[28] Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. 2024. Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey. arXiv:2403.14608 [cs.LG]

[29] M. He, T. Jia, C. Duan, H Cai, Y. Li, , and G. Huang. 2024. LLMeLog: An Approach for Anomaly Detection based on LLM-enriched Log Events. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE Computer Society, Tsukuba, Japan.

[30] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. 2017. Drain: An online log parsing approach with fixed depth tree. , 33–40 pages. https://doi.org/10.1109/ICWS.2017.13

[31] Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R. Lyu. 2021. A Survey on Automated Log Analysis for Reliability Engineering. *ACM Comput. Surv.* 54, 6, Article 130 (jul 2021), 37 pages. https://doi.org/10.1145/3460345

[32] Shilin He, Jieming Zhu, Pinjia He, and Michael R. Lyu. 2016. Experience Report: System Log Analysis for Anomaly Detection. , 207–218 pages. https://doi.org/10.1109/ISSRE.2016.21

[33] Shilin He, Jieming Zhu, Pinjia He, and Michael R. Lyu. 2020. Loghub: A Large Collection of System Log Datasets Towards Automated Log Analytics. *CoRR* abs/2008.06448 (2020). https://arxiv.org/abs/2008.06448

[34] Maryamsadat Hejazi and Yashwant Prasad Singh. 2013. One-class support vector machines approach to anomaly detection. *Applied Artificial Intelligence* 27, 5 (2013), 351–366.

[35] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (nov 1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[36] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. LoRA: Low-Rank Adaptation of Large Language Models. arXiv:2106.09685 [cs.CL]

[37] Guangbin Huang, Yan Qiu Chen, and Haroon Atique Babri. 2000. Classification ability of single hidden layer feedforward neural networks. *IEEE transactions on neural networks* 11 3 (2000), 799–801. https://api.semanticscholar.org/CorpusID:1852628

[38] Shaohan Huang, Yi Liu, Carol Fung, Rong He, Yining Zhao, Hailong Yang, and Zhongzhi Luan. 2020. HitAnomaly: Hierarchical Transformers for Anomaly Detection in System Log. *IEEE Transactions on Network and Service Management* 17, 4 (2020), 2064–2076. https://doi.org/10.1109/TNSM.2020.3034647

[39] Y. Huo, C. Lee, Y. Su, S. Shan, J. Liu, and M. R. Lyu. 2023. EvLog: Identifying Anomalous Logs over Software Evolution. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE Computer Society, Los Alamitos, CA, USA, 391–402. https://doi.org/10.1109/ISSRE59848.2023.00018

[40] Intel. 2021. HiBench. https://github.com/Intel-bigdata/HiBench.

[41] Chandra Irugalbandara, Ashish Mahendra, Roland Daynauth, Tharuka Kasthuri Arachchige, Jayanaka Dantanarayana, Krisztian Flautner, Lingjia Tang, Yiping Kang, and Jason Mars. 2024. Scaling Down to Scale Up: A Cost-Benefit Analysis of Replacing OpenAI's LLM with Open Source SLMs in Production. In *2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 280–291. https://doi.org/10.1109/ISPASS61541.2024.00034

[42] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B. arXiv:2310.06825 [cs.CL] https://arxiv.org/abs/2310.06825

[43] Zhihan Jiang, Jinyang Liu, Zhuangbin Chen, Yichen Li, Junjie Huang, Yintong Huo, Pinjia He, Jiazhen Gu, and Michael R. Lyu. 2024. LILAC: Log Parsing using LLMs with Adaptive Parsing Cache. arXiv:2310.01796 [cs.SE]

[44] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. 2016. FastText.zip: Compressing text classification models. arXiv:1612.03651 [cs.CL]

[45] Suhas Kabinna, Weiyi Shang, Cor-Paul Bezemer, and Ahmed E. Hassan. 2016. Examining the Stability of Logging Statements. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. 326–337. https://doi.org/10.1109/SANER.2016.29

[46] Harsurinder Kaur, Husanbir Singh Pannu, and Avleen Kaur Malhi. 2019. A systematic review on imbalanced data challenges in machine learning: Applications and solutions. *ACM computing surveys (CSUR)* 52, 4 (2019), 1–36.

[47] Zanis Ali Khan, Donghwan Shin, Domenico Bianculli, and Lionel C. Briand. 2024. Impact of log parsing on deep learning-based anomaly detection. *Empirical Software Engineering* 29, 6 (Aug. 2024). https://doi.org/10.1007/s10664-024-10533-w

[48] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Alessandro Moschitti, Bo Pang, and Walter Daelemans (Eds.). Association for Computational Linguistics, Doha, Qatar, 1746–1751. https://doi.org/10.3115/v1/D14-1181

[49] Herb Krasner. 2021. The cost of poor software quality in the US: A 2020 report. *Proc. Consortium Inf. Softw. QualityTM (CISQTM)* (2021), 1–46.

[50] Max Landauer, Sebastian Onder, Florian Skopik, and Markus Wurzenberger. 2023. Deep learning for anomaly detection in log data: A survey. *Machine Learning with Applications* 12 (2023), 100470. https://doi.org/10.1016/j.mlwa.2023.100470

[51] V. Le and H. Zhang. 2021. Log-based Anomaly Detection Without Log Parsing. , 492-504 pages. https://doi.org/10.1109/ASE51524.2021.9678773

[52] V. Le and H. Zhang. 2023. Log Parsing: How Far Can ChatGPT Go?. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE Computer Society, Los Alamitos, CA, USA, 1699–1704. https://doi.org/10.1109/ASE56229.2023.00206

[53] Van-Hoang Le and Hongyu Zhang. 2022. Log-based anomaly detection with deep learning: how far are we?. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 1356–1367. https://doi.org/10.1145/3510003.3510155

[54] 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

[55] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436.

[56] 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. *Proceedings - International Symposium on Software Reliability Engineering, ISSRE* 2020-Octob (2020), 92–103. https://doi.org/10.1109/ISSRE5003.2020.00018

[57] 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, ICSE - Companion Volume*. ACM, Austin, TX, USA, 102–111. https://doi.org/10.1145/2889160.2889232

[58] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation Forest. In *2008 Eighth IEEE International Conference on Data Mining*. 413–422. https://doi.org/10.1109/ICDM.2008.17

[59] Yilun Liu, Shimin Tao, Weibin Meng, Feiyu Yao, Xiaofeng Zhao, and Hao Yang. 2024. LogPrompt: Prompt Engineering Towards Zero-Shot and Interpretable Log Analysis. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings* (<conf-loc>, <city>Lisbon</city>, <country>Portugal</country>, </conf-loc>) *(ICSE-Companion '24)*. Association for Computing Machinery, New York, NY, USA, 364–365. https://doi.org/10.1145/3639478.3643108

[60] Steven Locke, Heng Li, Tse-Hsun Peter Chen, Weiyi Shang, and Wei Liu. 2022. LogAssist: Assisting Log Analysis Through Log Summarization. *IEEE Transactions on Software Engineering* 48, 9 (2022), 3227–3241. https://doi.org/10.1109/TSE.2021.3083715

[61] Long Ouyang Jeff Wu Xu Jiang Diogo Almeida Carroll L. Wainwright Pamela Mishkin Paul Christiano Jan Leike Ryan Lowe et al. 2023. GPT-4 Technical Report. https://api.semanticscholar.org/CorpusID:257532815

[62] Siyang Lu, Xiang Wei, Yandong Li, and Liqiang Wang. 2018. Detecting Anomaly in Big Data System Logs Using Convolutional Neural Network. *IEEE Access* 6 (2018), 21929–21940. https://doi.org/10.1109/ACCESS.2018.2811530

[63] Zeyang Ma, An Ran Chen, Dong Jae Kim, Tse-Hsun Chen, and Shaowei Wang. 2024. LLMParser: An Exploratory Study on Using Large Language Models for Log Parsing. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (<conf-loc>, <city>Lisbon</city>, <country>Portugal</country>, </conf-loc>) *(ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 99, 13 pages. https://doi.org/10.1145/3597503.3639150

[64] Leland McInnes, John Healy, and Steve Astels. 2017. hdbscan: Hierarchical density based clustering. *J. Open Source Softw.* 2, 11 (2017), 205.

[65] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, et al. 2019. Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs.. In *IJCAI*, Vol. 19. 4739–4745.

[66] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. 2018. A Simple Neural Attentive Meta-Learner. arXiv:1707.03141 [cs.AI] https://arxiv.org/abs/1707.03141

[67] MistralAI. 2024. Mistral Technologies. https://mistral.ai/technology/ accessed 2024-09-24.

[68] Roweida Mohammed, Jumanah Rawashdeh, and Malak Abdullah. 2020. Machine learning with oversampling and undersampling techniques: overview study and experimental results. In *2020 11th international conference on information and communication systems (ICICS)*. IEEE, 243–248.

[69] Marius Mosbach, Tiago Pimentel, Shauli Ravfogel, Dietrich Klakow, and Yanai Elazar. 2023. Few-shot Fine-tuning vs. In-context Learning: A Fair Comparison and Evaluation. In *Findings of the Association for Computational Linguistics: ACL 2023*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 12284–12314. https://doi.org/10.18653/v1/2023.findings-acl.779

[70] Adam Oliner and Jon Stearley. 2007. What Supercomputers Say: A Study of Five System Logs. , 575–584 pages.

[71] OpenAI. 2024. Fine-tuning: preparing your dataset. https://platform.openai.com/docs/guides/fine-tuning accessed 2024-01-02.

[72] OpenAI. 2024. OpenAI Models. https://platform.openai.com/docs/models accessed 2024-05-10.

[73] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke E. Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Francis Christiano, Jan Leike, and Ryan J. Lowe. 2022. Training language models to follow instructions with human feedback. *ArXiv* abs/2203.02155 (2022). https://api.semanticscholar.org/CorpusID:246426909

[74] C. Pei, Z.Liu, Jianhui Li, E. Zhang, L. Zhang, H. Zhang, W. Chen, D. Pei, and G. Xie. 2024. Self-Evolutionary Group-wise Log Parsing Based on Large Language Model. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE Computer Society, Tsukuba, Japan.

[75] Robi Polikar. 2012. Ensemble learning. *Ensemble machine learning: Methods and applications* (2012), 1–34.

[76] M. Schuster and K.K. Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing* 45, 11 (1997), 2673–2681. https://doi.org/10.1109/78.650093

[77] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. 2010. The Hadoop Distributed File System. , 10 pages.

[78] Qwen Team. 2025. Qwen2.5-1M: Deploy Your Own Qwen with Context Length up to 1M Tokens. https://qwenlm.github.io/blog/qwen2.5-1m/

[79] Hugo Touvron, Louis Martin, Kevin Stone, and Peter Albert. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288 [cs.CL]

[80] James Turnbull. 2018. *Monitoring with Prometheus*. Turnbull Press.

[81] Unsloth. 2024. Unsloth fine-tuning package. https://github.com/unslothai/unsloth accessed 2024-11-30.

[82] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 6000–6010.

[83] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, et al. 2013. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*. 1–16.

[84] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science* 18, 6 (2024), 186345.

[85] Xiaolei Wang, Lin Yang, Dongyang Li, Linru Ma, Yongzhong He, Junchao Xiao, Jiyuan Liu, and Yuexiang Yang. 2022. MADDC: Multi-Scale Anomaly Detection, Diagnosis and Correction for Discrete Event Logs. In *Proceedings of the 38th Annual Computer Security Applications Conference* (<conf-loc>, <city>Austin</city>, <state>TX</state>, <country>USA</country>, </conf-loc>) *(ACSAC '22)*. Association for Computing Machinery, New York, NY, USA, 769–784. https://doi.org/10.1145/3564625.3567972

[86] Mark Winteringham. 2024. Software Testing with Generative AI. , 304 pages.

[87] Yi Xiao, Van-Hoang Le, and Hongyu Zhang. 2024. Demonstration-Free: Towards More Practical Log Parsing with Large Language Models. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering* (Sacramento, CA, USA) *(ASE '24)*. Association for Computing Machinery, New York, NY, USA, 153–165. https://doi.org/10.1145/3691620.3694994

[88] Junjielong Xu, Ruichun Yang, Yintong Huo, Chengyu Zhang, and Pinjia He. 2024. DivLog: Log Parsing with Prompt Enhanced In-Context Learning. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (<conf-loc>, <city>Lisbon</city>, <country>Portugal</country>, </conf-loc>) *(ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 199, 12 pages. https://doi.org/10.1145/3597503.3639155

[89] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. 2009. Online System Problem Detection by Mining Patterns of Console Logs. , 588-597 pages. https://doi.org/10.1109/ICDM.2009.19

[90] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I. Jordan. 2009. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles* (Big Sky, Montana, USA) *(SOSP '09)*. Association for Computing Machinery, New York, NY, USA, 117–132. https://doi.org/10.1145/1629575.1629587

[91] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I. Jordan. 2010. Detecting large-scale system problems by mining console logs. , 37–44 pages.

[92] An Yang, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoyan Huang, Jiandong Jiang, Jianhong Tu, Jianwei Zhang, Jingren Zhou, Junyang Lin, Kai Dang, Kexin Yang, Le Yu, Mei Li, Minmin Sun, Qin Zhu, Rui Men, Tao He, Weijia Xu, Wenbiao Yin, Wenyuan Yu, Xiafei Qiu, Xingzhang Ren, Xinlong Yang, Yong Li, Zhiying Xu, and Zipeng Zhang. 2025. Qwen2.5-1M Technical Report. *arXiv preprint arXiv:2501.15383* (2025).

[93] Lin Yang, Junjie Chen, Zan Wang, Weijing Wang, Jiajun Jiang, Xuyuan Dong, and Wenbin Zhang. 2021. Semi-Supervised Log-Based Anomaly Detection via Probabilistic Label Estimation. , 1448-1460 pages. https://doi.org/10.1109/ICSE43902.2021.00130

[94] Sohee Yang, Nora Kassner, Elena Gribovskaya, Sebastian Riedel, and Mor Geva. 2024. Do Large Language Models Perform Latent Multi-Hop Reasoning without Exploiting Shortcuts? arXiv:2411.16679 [cs.CL] https://arxiv.org/abs/2411.16679

[95] Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Zhibo Sun, and Yue Zhang. 2024. A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing* (2024), 100211.

[96] Boxi Yu, Jiayi Yao, Qiuai Fu, Zhiqing Zhong, Haotian Xie, Yaoliang Wu, Yuchi Ma, and Pinjia He. 2024. Deep Learning or Classical Machine Learning? An Empirical Study on Log-Based Anomaly Detection. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) *(ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 35, 13 pages. https://doi.org/10.1145/3597503.3623308

[97] Boxi Yu, Jiayi Yao, Qiuai Fu, Zhiqing Zhong, Haotian Xie, Yaoliang Wu, Yuchi Ma, and Pinjia He. 2024. Deep Learning or Classical Machine Learning? An Empirical Study on Log-Based Anomaly Detection. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) *(ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 35, 13 pages. https://doi.org/10.1145/3597503.3623308

[98] Weihao Yu, Mi Luo, Pan Zhou, Chenyang Si, Yichen Zhou, Xinchao Wang, Jiashi Feng, and Shuicheng Yan. 2022. MetaFormer Is Actually What You Need for Vision. arXiv:2111.11418 [cs.CV] https://arxiv.org/abs/2111.11418

[99] Shichao Zhang. 2021. Challenges in KNN classification. *IEEE Transactions on Knowledge and Data Engineering* 34, 10 (2021), 4663–4675.

[100] T. Zhang, H. Qiu, G. Castellano, M. Rifai, C. Chen, and F. Pianese. 2023. System Log Parsing: A Survey. *IEEE Transactions on Knowledge & Data Engineering* 35, 08 (aug 2023), 8596–8614. https://doi.org/10.1109/TKDE.2022.3222417

[101] W. Zhang, T. Jia, C. Duan, H Cai, Y. Li, , and G. Huang. 2024. Leveraging RAG-Enhanced Large Language Model for Semi-Supervised Log Anomaly Detection. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE Computer Society, Tsukuba, Japan.

[102] 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

[103] Yuzhen Zhang, Jingjing Liu, and Wenjuan Shen. 2022. A review of ensemble learning algorithms used in remote sensing applications. *Applied Sciences* 12, 17 (2022), 8654.

[104] Chen Zhi, Liye Cheng, Meilin Liu, Xinkui Zhao, Yueshen Xu, and Shuiguang Deng. 2024. LLM-powered Zero-shot Online Log Parsing. In *2024 IEEE International Conference on Web Services (ICWS)*. 877–887. https://doi.org/10.1109/ICWS62655.2024.00106

[105] Aoxiao Zhong, Dengyao Mo, Guiyang Liu, Jinbu Liu, Qingda Lu, Qi Zhou, Jiesheng Wu, Quanzheng Li, and Qingsong Wen. 2024. LogParser-LLM: Advancing Efficient Log Parsing with Large Language Models. arXiv:2408.13727 [cs.SE] https://arxiv.org/abs/2408.13727

[106] Zhi-Hua Zhou and Zhi-Hua Zhou. 2021. *Ensemble learning*. Springer.

A APPENDIX

A.1 Alternative Ensembling Strategies

FlexLog employs a majority voting strategy, where in the case of a tie between base models, the label is set to *normal*, as anomalies are rare. This ensemble approach uses binary labels for voting rather than anomaly confidence scores, allowing us to adopt base models that directly output the final label, such as our fine-tuned version of Mistral. Alternative ensembling strategies include the alternative majority voting method, which assigns a random label in the case of a tie, and state-of-the-art meta-learning approaches that adaptively learn from base model performance on validation data. Specifically, SNAIL [66] is an attentive CNN-based meta-learner, while MetaFormer [98] introduces a novel attention module called *token mixer*; both have shown significant improvements over traditional meta-learning approaches. We experimented with these methods using their original implementation code.

Table 18. F1 scores of using alternative Ensembling Strategies in FlexLog.

Ensembling Config	ADFA-U							LOGEVOL-U			SynHDFS-U	SYNEVOL-U
	adduser	hydraFTP	hydraSSH	java	meter	web	average	Hadoop	Spark	average	average	average
Majority Voting (FlexLog)	**0.718**	**0.784**	**0.723**	**0.642**	**0.682**	**0.672**	**0.704**	**0.982**	**0.892**	**0.937**	**0.972**	**0.971**
Majority Voting (alternative)	0.641	0.711	0.691	0.615	0.656	0.635	0.650*	0.964	0.840	0.902*	0.936*	0.929*
SNAIL	0.671	0.714	0.683	0.624	0.628	0.651	0.661*	0.965	0.854	0.909*	0.958*	0.949*
MetaFormer	0.666	0.706	0.691	0.615	0.607	0.663	0.658*	0.976	0.866	0.921	0.954*	0.951*

* FlexLog yields a significant higher F1 score compared to using the alternative ensembling strategy.

Table 18 reports the F1 scores of FlexLog using different ensemble strategies, including the original majority voting (*Majority Voting (alternative)*), and two meta-learning approaches, *SNAIL* and *MetaFormer*. Experimental results show that FlexLog's majority voting consistently outperforms these alternatives with an average difference of 5 pp, 3 pp, 2 pp, and 3 pp, respectively, for ADFA-U, LOGEVOL-U, SynHDFS-U, and LOGEVOL-U. Mann-Whitney U tests at the dataset level further confirm that the differences in F1 score between FlexLog and the alternative strategies are statistically significant across all four datasets, except for MetaFormer on LOGEVOL-U. Overall, the modified majority voting in FlexLog remains the recommended strategy due to its simplicity and effectiveness.

A.2 Impact of De-duplication

Table 19 reports the F1 scores of FlexLog and baseline models evaluated on both de-duplicated and original test data. The column "dedup" indicates whether the test set was de-duplicated from the training set. Overall, all models exhibit inflated F1 scores when the test set is not de-duplicated from the training set, highlighting the impact of data leakage. For instance, NeuralLog shows an average inflation of 16 pp, 2 pp, 1 pp, and 24 pp on ADFA-U, LOGEVOL-U, SynHDFS-U, and SYNEVOL-U, respectively. Mann-Whitney U tests confirm that the differences are statistically significant for Neurallog. In contrast to the most effective baseline (LightAD), FlexLog keeps the inflation in F1 score below 2 pp across all datasets, where the difference between FlexLog's performance on de-duplicated and original test data is statistically insignificant, confirmed by Mann-Whitney U tests. These results indicate that FlexLog delivers more reliable and robust performance. To avoid any risk of inflated results, all reported performances in this paper are based on de-duplicated test data.

A.3 Baselines with Limited Data

Table 20 presents the effectiveness of FlexLog compared to baselines when all models are trained on the same limited dataset. FlexLog consistently achieves the highest F1 score across all datasets under this setting. For instance, FlexLog outperforms the supervised baselines LightAD, NeuralLog, LogRobust, and CNN by 10 pp, 16 pp, 30 pp, and 14 pp, respectively, on average for ADFA-U. This advantage is important because supervised models such as NeuralLog and CNN depend on large labeled datasets to achieve high accuracy. Mann-Whitney U tests confirm that the observed performance gaps between FlexLog and each baseline are statistically significant across all datasets, demonstrating that none of the baselines match FlexLog 's performance with limited data. Notably, FlexLog trained with limited data even outperforms baselines trained with full datasets in terms of predictive effectiveness. Detailed results are presented and discussed in RQ1 (§ 5.1).

A.4 Effectiveness on SYNEVOL-U with Sequence Level Changes

Table 21 presents the effectiveness of FlexLog compared to baselines on SYNEVOL-U under varying log sequence instability levels. FlexLog consistently achieves the highest F1 score across all

Table 19. Effectiveness of FlexLog and baselines for ULAD on ADFA-U, LOGEVOL-U, SynHDFS-U, and SYNEVOL-U with and without de-duplication.

Model	Dedup	ADFA-U							LOGEVOL-U			SynHDFS-U	SYNEVOL-U
		adduser	hydraFTP	hydraSSH	java	meter	web	average	Hadoop	Spark	average	average	average
FlexLog	Yes	0.718	0.784	**0.723**	0.642	0.682	0.672	0.704	0.982	0.892	**0.937**	0.972	**0.971**
	No	0.723	0.790	0.726	0.652	0.701	0.673	0.711	0.996	0.895	0.945	0.974	0.987
LightAD	Yes	0.725	0.754	0.666	0.639	0.679	0.602	0.677	0.980	0.829	0.898	0.959	0.956
	No	0.745	0.778	0.745	0.646	0.695	0.618	0.704*	0.995	0.876	0.935*	0.968	0.981*
NeuralLog	Yes	0.412	0.388	0.368	0.511	0.461	0.457	0.433	0.948	0.834	0.891	0.946	0.765
	No	0.606	0.681	0.601	0.570	0.645	0.492	0.599*	0.961	0.859	0.910*	0.951	0.905*
LogRobust	Yes	0.524	0.408	0.374	0.558	0.651	0.449	0.494	0.927	0.757	0.833	0.760	0.941
	No	0.636	0.597	0.504	0.662	0.660	0.496	0.592*	0.981	0.789	0.885*	0.929*	0.966*
CNN	Yes	0.641	0.750	0.750	0.635	0.711	0.621	0.685	0.980	0.840	0.910	0.942	0.918
	No	0.703	0.765	0.777	0.644	0.724	0.634	0.707*	0.989	0.863	0.925	0.961*	0.925
PLELog	Yes	0.361	0.388	0.473	0.430	0.253	0.233	0.356	0.709	0.165	0.437	0.499	0.164
	No	0.405	0.428	0.494	0.443	0.311	0.277	0.393*	0.761	0.223	0.492*	0.528*	0.236*
LogAnomaly	Yes	0.291	0.451	0.480	0.422	0.218	0.343	0.368	0.310	0.216	0.263	0.446	0.304
	No	0.305	0.464	0.486	0.399	0.237	0.351	0.373	0.619	0.212	0.415*	0.498*	0.336*
DeepLog	Yes	0.292	0.481	0.458	0.339	0.253	0.353	0.363	0.367	0.122	0.244	0.741	0.253
	No	0.340	0.499	0.450	0.341	0.249	0.369	0.374	0.685	0.141	0.413*	0.776*	0.291*
LogCluster	Yes	0.334	0.418	0.461	0.348	0.175	0.304	0.340	0.430	0.485	0.458	0.519	0.714
	No	0.326	0.431	0.523	0.317	0.211	0.336	0.357*	0.798	0.614	0.706*	0.759*	0.786*
PCA	Yes	0.165	0.144	0.140	0.158	0.241	0.197	0.174	0.360	0.108	0.234	0.404	0.103
	No	0.155	0.152	0.163	0.277	0.211	0.198	0.192*	0.454	0.097	0.275*	0.567*	0.189*

* The same model shows a significant F1 score difference between deduplicated and original test data.

Table 20. Effectiveness of FlexLog and Baselines Trained with Limited Data on ADFA-U, LOGEVOL-U, SynHDFS-U, and SYNEVOL-U.

Model	ADFA-U							LOGEVOL-U			SynHDFS-U	SYNEVOL-U
	adduser	hydraFTP	hydraSSH	java	meter	web	average	Hadoop	Spark	average	average	average
FlexLog	**0.718**	**0.784**	**0.723**	**0.642**	**0.682**	**0.672**	**0.704**	**0.982**	**0.892**	**0.937**	**0.972**	**0.971**
LightAD	0.587	0.720	0.633	0.564	0.520	0.557	0.596*	0.973	0.821	0.897*	0.925*	0.915*
NeuralLog	0.607	0.645	0.630	0.577	0.438	0.328	0.537*	0.975	0.232	0.603*	0.914*	0.270*
LogRobust	0.393	0.551	0.424	0.392	0.299	0.374	0.405*	0.851	0.254	0.552*	0.930*	0.903*
CNN	0.534	0.703	0.603	0.545	0.422	0.549	0.559*	0.819	0.217	0.518*	0.925*	0.863*
PLELog	0.495	0.457	0.163	0.144	0.102	0.322	0.280*	0.309	0.313	0.311*	0.378*	0.160*
LogAnomaly	0.310	0.359	0.346	0.345	0.178	0.274	0.302*	0.298	0.105	0.201*	0.152*	0.251*
DeepLog	0.261	0.358	0.409	0.287	0.154	0.272	0.290*	0.355	0.061	0.208*	0.188*	0.219*
LogCluster	0.226	0.340	0.363	0.281	0.185	0.254	0.274*	0.460	0.321	0.390*	0.547*	0.445*
PCA	0.077	0.136	0.187	0.164	0.037	0.066	0.111*	0.134	0.051	0.092*	0.103*	0.120*

* FlexLog yields a significant higher F1 score than compared baselines.

injection ratios, surpassing LightAD by 1.3 pp on average (98.2% − 96.9%). A Mann-Whitney U test confirms this difference is statistically insignificant, indicating comparable effectiveness between FlexLog and LightAD. However, FlexLog achieves this performance while being trained on only 22.96 % of unique log sequences, reducing usage of labeled data by 77.04 pp. Unlike in SynHDFS-U, where only LightAD and FlexLog remain robust across instability levels, all supervised methods—including FlexLog, LightAD, NeuralLog, LogRobust, and CNN—do not show a strong correlation between performance and increasing log instability in SYNEVOL-U. In contrast, semi-supervised and unsupervised methods exhibit a clear decline in precision, recall, and F1 score as instability increases. One possible reason is that changes at the log sequence level in SYNEVOL-U often involve minor modifications, such as adding or removing a single template, which may have a limited

impact on ULAD. For instance, at a 30 % injection ratio, 55 % of changes involve just one template, making the overall sequence structure relatively stable despite modifications.

Table 21. Effectiveness of FlexLog and baselines under different sequence-level injection ratios on SYNEVOL-U.

Data	Unstable	M	limited data	full training set								
				supervised				Semi-S	Unsupervised			
			FlexLog	LightAD	NeuralLog	LogRobust	CNN	PLELog	LogAnomaly	DeepLog	LogCluster	PCA
0%	No	P	0.999	0.999	0.999	0.941	0.999	0.172	0.501	0.512	0.771	0.072
		R	0.969	0.939	0.636	0.969	0.878	0.129	0.393	0.443	0.818	0.471
		F1	**0.984**	0.968	0.777	0.952	0.935	0.243	0.441	0.475	0.794	0.125
5%	Yes	P	0.999	0.999	0.999	0.969	0.999	0.179	0.388	0.384	0.783	0.057
		R	0.971	0.942	0.628	0.914	0.885	0.141	0.440	0.428	0.828	0.457
		F1	**0.985**	0.970	0.771	0.941	0.939	0.158	0.394	0.405	0.805	0.102
10%	Yes	P	0.999	0.999	0.999	0.971	0.999	0.177	0.326	0.271	0.769	0.072
		R	0.973	0.945	0.648	0.918	0.891	0.145	0.459	0.432	0.810	0.471
		F1	**0.986**	0.972	0.786	0.944	0.942	0.160	0.382	0.333	0.789	0.125
15%	Yes	P	0.999	0.999	0.999	0.971	0.999	0.163	0.224	0.229	0.769	0.063
		R	0.973	0.945	0.621	0.918	0.891	0.141	0.351	0.378	0.810	0.475
		F1	**0.986**	0.972	0.766	0.944	0.942	0.151	0.273	0.285	0.789	0.112
20%	Yes	P	0.999	0.999	0.999	0.973	0.999	0.189	0.205	0.200	0.804	0.053
		R	0.975	0.951	0.583	0.902	0.878	0.154	0.365	0.439	0.804	0.470
		F1	**0.987**	0.975	0.738	0.936	0.935	0.170	0.263	0.274	0.804	0.095
25%	Yes	P	0.999	0.999	0.999	0.975	0.999	0.167	0.180	0.165	0.800	0.071
		R	0.953	0.930	0.651	0.930	0.906	0.141	0.441	0.418	0.837	0.511
		F1	**0.976**	0.963	0.788	0.952	0.951	0.153	0.256	0.236	0.818	0.125
30%	Yes	P	0.999	0.999	0.999	0.972	0.972	0.174	0.162	0.125	0.659	0.058
		R	0.950	0.925	0.600	0.900	0.875	0.145	0.475	0.425	0.775	0.434
		F1	**0.974**	0.961	0.750	0.935	0.921	0.158	0.242	0.193	0.712	0.102
Average	Yes	P	0.999	0.999	0.999	0.972	0.994	0.175	0.247	0.229	0.764	0.062
		R	0.966	0.94	0.622	0.914	0.888	0.144	0.422	0.42	0.811	0.47
		F1	**0.982**	0.969	0.767*	0.942*	0.938*	0.158*	0.302*	0.288*	0.786*	0.11 *

* FlexLog yields a significant higher F1-score than compared baseline.