CoorLog: Efficient-Generalizable Log Anomaly Detection via Adaptive Coordinator in Software Evolution

Pei Xiao†‡, Chiming Duan†‡, Minghua He†‡, Tong Jia§‖*, Yifan Wu‡, Jing Xu††, Gege Gao††, Lingzhe Zhang‡, Weijie Hong‡, Ying Li‡¶*, Gang Huang‖
‡School of Software and Microelectronics, Peking University, Beijing, China
{xiaopei, duanchiming, hemh2120, zhang.lingzhe, hongwj}@stu.pku.edu.cn, {yifanwu, li.ying}@pku.edu.cn
§Institute for Artificial Intelligence, Peking University, Beijing, China
jia.tong@pku.edu.cn
¶National Engineering Research Center for Software Engineering, Peking University, Beijing, China
li.ying@pku.edu.cn
‖National Key Laboratory of Data Space Technology and System, Beijing, China
{jia.tong, hg}@pku.edu.cn
††Bytedance, Beijing, China
{xujing.66, gaogege.1002}@bytedance.com

***Abstract*—Frequent software updates lead to log evolution, posing generalization challenges for current log anomaly detection. Traditional log anomaly detection research focuses on using small deep learning models (SMs), but these models inherently lack generalization due to their closed-world assumption. Large language models (LLMs) exhibit strong semantic understanding and generalization capabilities, making them promising for log anomaly detection. However, they suffer from computational inefficiencies. To balance efficiency and generalization, we propose a collaborative log anomaly detection scheme (CoorLog) that uses an adaptive coordinator to integrate SM and LLM. The coordinator determines if incoming logs have evolved. Non-evolved logs are routed to the SM, while evolved logs are directed to the LLM for detailed inference using the constructed Evol-CoT. To gradually adapt to evolution, we introduce the adaptive evolution mechanism (AEM), which updates the coordinator to redirect evolved logs identified by the LLM to the SM. Simultaneously, the SM is fine-tuned to inherit the LLM's judgment on these logs. Extensive experiments on real-world datasets demonstrate that CoorLog achieves superior F1-scores in both intra-version and inter-version anomaly detection. Additionally, CoorLog reduces processing time by 91.63% and token consumption by 85.59% compared to using an LLM alone.**

***Index Terms*—System Logs, Anomaly Detection, Large Language Models, Software Evolution, Model Collaboration**

I. Introduction

As software systems grow in scale and complexity, the likelihood of failures increases, often leading to significant losses [1]. Detecting anomalies by analyzing runtime data is crucial to prevent failures from escalating. System logs, which record system states and critical events, play a vital role in understanding system behavior. Log-based anomaly detection

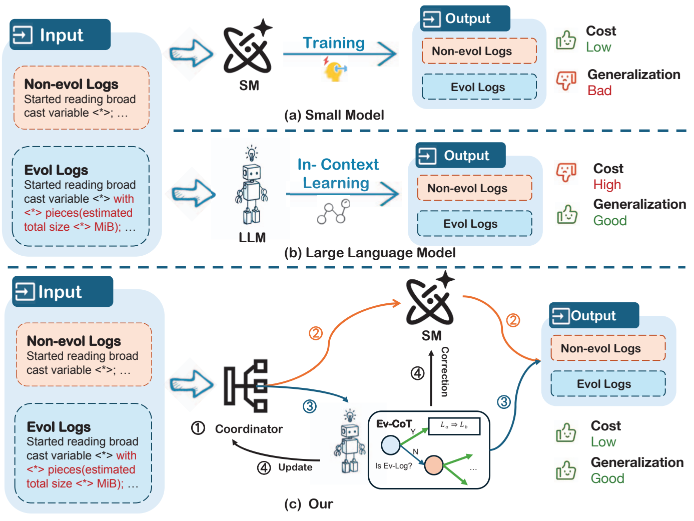


Fig. 1: Comparison of (a) Small Models (SM) are cost-efficient but generalize poorly for evolved logs; (b) Large Language Models (LLMs) offer semantic-aware generalization at high computational cost. (c) Our framework routes non-evolved (Non-evol)/evolved (Evol) logs to SM/LLM respectively via a coordinator, with an adaptive updates mechanism to both while avoiding redundant LLM inferences for duplicates.

(LogAD) has become a key approach for ensuring software reliability and has been extensively studied.

A key challenge in applying log anomaly detection is software evolution. After an initial release, software typically undergoes continuous updates to meet user demands, fix bugs, and introduce new features—a process known as software evo-

† Equal contribution.
* Corresponding authors.

lution [2]–[4]. During software evolution, the log-producing code is frequently modified, leading to corresponding changes in system logs. Previous studies [5], [6] indicate that log statement modifications are common in software evolution, with approximately 33% of logs being altered post-release.

Current log anomaly detection methods can be broadly categorized into two types: traditional methods based on small-scale machine learning models and newer approaches leveraging pre-trained large language models (LLMs). Before LLMs emerged, log anomaly detection research mainly relied on deep learning models—now often known as small models (SMs) because of their compact parameter size [7]–[20]. Compared to LLMs, SMs are more cost-effective and easier to deploy. However, these training-based methods rely on the closed-world assumption, meaning they struggle to detect log sequences that fall outside their training data distribution (Fig. 1(a)). Although existing research [21], [22] attempts to adapt small models to concept drift through incremental learning, they still fail to accurately identify evolved logs upon their first appearance. These methods essentially merely reduce retraining costs but cannot compensate for the inherent lack of generalization in small models.

Recently, pre-trained large language models have demonstrated their exceptional capabilities in understanding semantics, reasoning, and generalization across various domains [23]–[28]. Logs contain rich semantic information, allowing LLMs to significantly improve anomaly detection performance, making LLM-based log anomaly detection a research focus. Prior work [29], [30] confirms that LLMs excel at extracting and utilizing semantic patterns from logs, while also providing key advantages such as interpretability [31], zero-shot generalization across systems [14], and improved robustness [10]. With their robust semantic awareness and contextual reasoning capabilities, LLMs can effectively accommodate deviations caused by log evolution at the semantic level, demonstrating generalization abilities unmatched by small models. However, they face significant cost challenges (Fig. 1(b)). Due to their massive parameter sizes, LLMs suffer from slow inference speeds and high costs. This becomes particularly problematic in large-scale software systems with high-volume log generation, where LLM-based methods struggle to achieve real-time performance.

To leverage the generalization capabilities of LLMs while minimizing additional costs, we propose CoorLog, a collaborative multi-model log anomaly detection method based on an adaptive coordinator. For incoming logs, the coordinator, an autoencoder trained on historical logs, first identifies and distributes them into non-evolved logs and evolved logs. Non-evolved logs, which do not deviate from the training data distribution, are routed to the small model (SM), while evolved logs are directed to the LLM. The SM, a BERT network trained on historical logs, can efficiently and accurately detect anomalies on logs without concept drift. To enhance the LLM's anomaly detection, we developed a hierarchical reasoning framework, Evol-CoT, based on previous research [30], [32], using RAG and Agent technologies. Evol-CoT first identifies evolved relationships within logs and, guided by sample uncertainty, adapts its reasoning path to either perform semantic analysis or conduct semantic comparisons.

Furthermore, we argue that samples of the same type should not be repeatedly submitted to the LLM, to avoid redundant invocations. Therefore, we designed an adaptive evolution mechanism (AEM) for model collaboration, allowing the small model to gradually inherit the results of logs processed by the LLM, and enabling the coordinator to redirect these logs to the small model. Specifically, the evolved logs identified during the Evol-CoT process are collected to update the coordinator and fine-tune the small model. It is worth noting that we do not directly use the results of the LLM for updates; instead, we only use samples that are explicitly identified as evolved logs in the Evol-CoT, as these results have higher confidence. Through the coordinator and AEM, our approach achieves both generalization capability and operational efficiency.

We evaluate our method on the public software evolution dataset LOGEVOL. Experimental results show that our approach achieves the highest F1-score for anomaly detection on both intra-version and cross-version settings. Furthermore, compared to directly using LLMs, our method reduces computational time by 91.63% and token consumption by 85.59%.

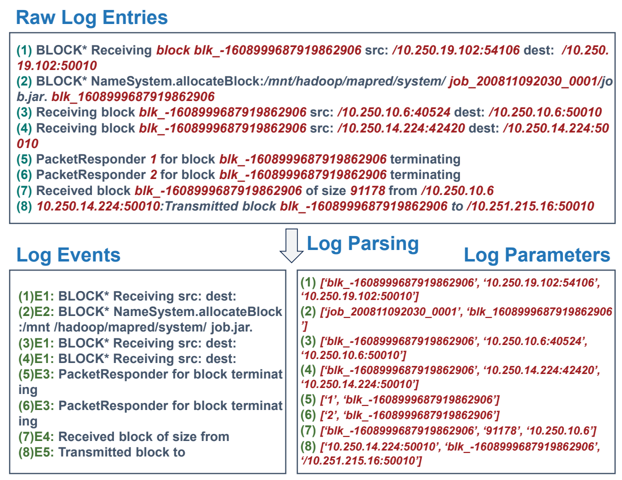
Specifically, the contributions of this paper are as follows:

- We propose a cost-efficient and generalizable log anomaly detection scheme using an adaptive coordinator to enable collaboration between LLMs and small models (SMs).
- We design Evol-CoT, which leverages chain-of-thought reasoning and RAG to identify evolved log relationships and enable more reliable anomaly detection in software evolution scenarios.
- We design an adaptive evolution mechanism (AEM) that continually adapts the collaboration process. High-confidence evolved relationships identified by the LLM update both the coordinator and the SM, allowing similar subsequent samples to be handled by the SM.
- We validate CoorLog on real-world log datasets, where it consistently achieves the best F1-score while significantly reducing cost compared to LLM-only approaches.

II. Background and Motivation

A. *Log Anomaly Detection*

Logs are textual records that capture crucial information about a system's execution. Each log entry generally consists of two parts: a constant part (log event) and a variable part (runtime parameters such as IP addresses or task IDs). Figure 2 provides an example from the BGL dataset [33]. The log event corresponds to the static text string, whereas log parameters encode dynamic attributes. Log parsing is a preprocessing step that separates events from parameters, producing structured representations of raw logs for downstream analysis, as shown in Figure 2. A log sequence is an ordered collection of logs, typically constructed by grouping logs based on task identifiers or timestamps. They serve as fundamental units for anomaly detection, reflecting system flow and context.



Raw Log Entries

```
(1) BLOCK* Receiving block blk_-1608999687919862906 src: /10.250.19.102:54106 dest:  /10.250.
19.102:50010
(2) BLOCK* NameSystem.allocateBlock:/mnt/hadoop/mapred/system/ job_200811092030_0001/jo
b.jar. blk_1608999687919862906
(3) Receiving block blk_-1608999687919862906 src: /10.250.10.6:40524 dest: /10.250.10.6:50010
(4) Receiving block blk_-1608999687919862906 src: /10.250.14.224:42420 dest: /10.250.14.224:50
010
(5) PacketResponder 1 for block blk_-1608999687919862906 terminating
(6) PacketResponder 2 for block blk_-1608999687919862906 terminating
(7) Received block blk_-1608999687919862906 of size 91178 from /10.250.10.6
(8) 10.250.14.224:50010:Transmitted block blk_-1608999687919862906 to /10.251.215.16:50010
```

Log Parsing

Log Events

```
(1)E1: BLOCK* Receiving src: dest:
(2)E2: BLOCK* NameSystem.allocateBlock
:/mnt /hadoop/mapred/system/ job.jar.
(3)E1: BLOCK* Receiving src: dest:
(4)E1: BLOCK* Receiving src: dest:
(5)E3: PacketResponder for block terminat
ing
(6)E3: PacketResponder for block terminat
ing
(7)E4: Received block of size from
(8)E5: Transmitted block to
```

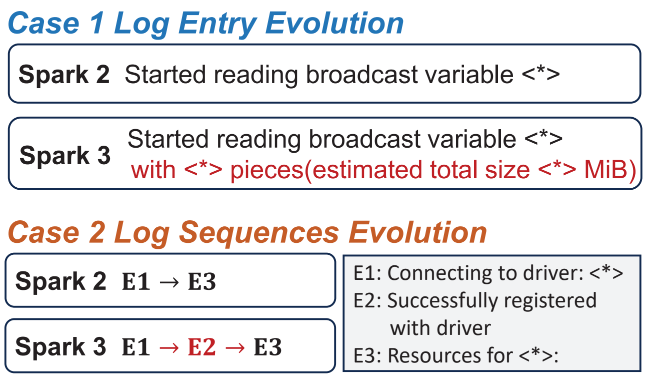
Log Parameters

```
(1) ['blk_-1608999687919862906', '10.250.19.102:54106',
'10.250.19.102:50010']
(2) ['job_200811092030_0001', 'blk_1608999687919862906
']
(3) ['blk_-1608999687919862906', '10.250.10.6:40524',
'10.250.10.6:50010']
(4) ['blk_-1608999687919862906', '10.250.14.224:42420',
'10.250.14.224:50010']
(5) ['1', 'blk_-1608999687919862906']
(6) ['2', 'blk_-1608999687919862906']
(7) ['blk_-1608999687919862906', '91178', '10.250.10.6']
(8) ['10.250.14.224:50010', 'blk_-1608999687919862906',
'/10.251.215.16:50010']
```

Fig. 2: An example of log parsing.

Traditional methods extract structured event sequences based on log parsing and train models based on historical patterns. Any deviation is flagged as an anomaly. However, these approaches rely on a closed-world assumption, limiting adaptability to evolving software. As software evolves, log structures and execution patterns change, leading to distribution shifts. Thus, existing methods struggle to generalize to unseen or modified log formats.

B. Log Evolution



Case 1 Log Entry Evolution

Spark 2 Started reading broadcast variable <*>

Spark 3 Started reading broadcast variable <*> with <*> pieces(estimated total size <*> MiB)

Case 2 Log Sequences Evolution

Spark 2 E1 → E3

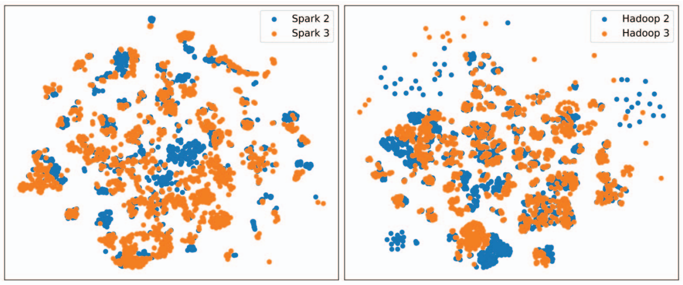
Spark 3 E1 → E2 → E3

E1: Connecting to driver: <*>
E2: Successfully registered with driver
E3: Resources for <*>:

Fig. 3: Two challenges brought by software evolution.

As software evolves, developers modify log statements to improve maintainability and adapt to new requirements. To study log evolution, we analyze version changes in two widely used data processing systems: Apache Spark [34] and Apache Hadoop [35], both included in the LOGEVOL dataset [2]. Specifically, we examine Spark 2.4.0 (Spark 2) vs. Spark 3.0.3 (Spark 3), and Hadoop 2.10.2 vs. Hadoop 3.3.3. These upgrades introduce substantial structural modifications, resulting in significant differences in log formats and content. Log evolution caused by software updates can be categorized into *log entry evolution* and *log sequence evolution*.

1) Log Entry Evolution: Log entry evolution refers to changes in individual log messages after software updates, such as the addition of more descriptive details, as illustrated in Case 1 of Figure 3. In Case 1, due to software evolution, the log entry has been modified, but the log parser is unaware of this change. Consequently, it fails to match the evolved log with its previous version and mistakenly recognizes it as a new log event. Previous studies have shown that nearly 24% of log statements changed from Spark 2 to Spark 3, leading to approximately 10% altered log messages [2]. Existing log parsers, such as Drain [36], often rely on predefined templates to extract structured information. However, these parsers may fail when logs evolve across versions, resulting in inconsistent or erroneous parsing that impacts downstream anomaly detection tasks. To address this challenge, We model log anomaly detection as a text-based binary classification task, directly operating on raw log messages to eliminate reliance on predefined parsing templates.



(a) Spark (b) Hadoop

Fig. 4: The t-SNE visualization of log sequence distribution changes from Spark 2 to Spark 3 and from Hadoop 2 to Hadoop 3.

2) Log Sequence Evolution: Log sequence evolution refers to changes in the order or dependencies among logs after software evolution. As shown in Case 2 of Figure 3, in Spark 2, a normal log sequence follows the pattern $E_1 \rightarrow E_3$. However, with the software update to Spark 3, the normal sequence changes to $E_1 \rightarrow E_2 \rightarrow E_3$. This evolution in log sequences directly alters the data distribution, introducing concept drift. Figure 4 visualizes the distribution shift of log sequences from Spark 2 to Spark 3 and from Hadoop 2 to Hadoop 3 using t-SNE [37]. The substantial differences in distribution highlight the impact of software evolution on log-based anomaly detection models. Traditional anomaly detection models, which rely on the assumption of a fixed data distribution, struggle to generalize to concept-drifted logs. In contrast, LLMs have demonstrated strong zero-shot capabilities, natural language understanding, and contextual reasoning, making them more suitable for handling evolving logs.

Overall, both log entry evolution and log sequence evolution alter the numerical distribution of logs in the feature space. However, they do not significantly impact the semantic meaning of the logs. For example, in Case 1, the evolved log entry introduces more detailed information while preserving the core semantics of the original Spark 2 log. Similarly, Case

2 expands upon the previous log sequence without altering its fundamental meaning. For LLMs, such variations do not hinder their semantic understanding.

C. Challenges of LogAD with LLM

Large Language Models (LLMs) have demonstrated strong capabilities in text comprehension, generalization, and zero-shot reasoning, showing promise for log anomaly detection [29], [30]. However, their use faces two major challenges: hallucination and inefficiency [38]. Hallucination—when LLMs generate plausible but incorrect outputs—can lead to misinterpretations in log analysis, where precision is critical [27], [39]. Existing mitigation strategies, such as domain-adaptive fine-tuning and retrieval-augmented generation (RAG) [40], either require costly domain datasets or introduce heavy computational overhead, limiting scalability in high-throughput scenarios [41]–[43]. Therefore, prior work [41], [44], [45] often used LLMs only for secondary judgment after smaller models. This post-processing method was still limited by the judgments of smaller models, presenting a clear bottleneck. To overcome these limitations, we introduce a coordinator that determines whether an input log deviates from the original distribution. Unlike prior cascade frameworks, where the LLM relies on the small model's judgment, the coordinator enables independent and adaptive collaboration. Furthermore, an Evol-CoT framework integrates RAG-based contextual retrieval with hierarchical reasoning to enhance reliability and mitigate hallucination.

III. Method

We propose a log anomaly detection scheme using an adaptive coordinator to enable collaboration between a large model (LLM) and a small model (SM), leveraging the LLM's generalization capability while minimizing computational costs. The coordinator, an autoencoder trained on pre-evolution data, classifies incoming logs as evolved (Evol) or non-evolved (Non-evol). Non-evol logs are processed by a BERT-based SM, while Evol logs are analyzed by the LLM through the Evol-CoT framework, identifying evolutionary relationships. The Adaptive Evolution Mechanism (AEM) uses these relationships to update both the coordinator and SM, reducing redundant LLM inferences. CoorLog improves detection accuracy, adapts to log evolution, and significantly reduces LLM computational costs.

A. Coordinator

The coordinator is responsible for identifying the input logs, routing non-evolved logs that have not undergone concept drift to the small model, and distributing evolved logs that deviate from the original data distribution to the LLM.

Let $D\_train_{coor} = \{s_1, s_2, \ldots, s_n\}$ represent the set of available log sequences for training prior to software evolution, where s_i denotes a log sequence, comprising an ordered set of logs $s_i = \{l_1, l_2, \ldots, l_m\}$. The coordinator consists of a BERT [46] semantic extraction network and an autoencoder. For a given log sequence s, we first use BERT to encode each log entry $l_i \in s$ into an embedding, then average all embeddings in a log sequence to get its feature vector. This allows the model to handle variable-length logs. Subsequently, a deep autoencoder is employed to capture the distribution of logs prior to software evolution.

Autoencoder [47] is a feed-forward multilayer neural network designed to learn a compressed, encoded representation of input data, and then decode this representation back to a form that is as close as possible to the original input. It consists of two main parts: an encoder, which compresses the input into a latent-space representation, and a decoder, which reconstructs the input from this latent representation. For an input log sequence s and output $\hat{s}$, the autoencoder can be formulated as follows:

$$\begin{gathered}\tilde{s} \sim \text{ Dropout } (s) \\ z = \sigma_1 (W_1 \tilde{s} + b_1) \\ \tilde{z} \sim \text{ Dropout } (z) \\ \hat{s} = \sigma_2 (W_2 \tilde{z} + b_2)\end{gathered} \tag{1}$$

The random mapping function $Dropout()$ randomly zeroes out a portion of the input dimensions, resulting in $\tilde{s}$ as a noise-corrupted version of s. σ_1 and σ_2 are the activation functions for the encoding and decoding layers, respectively, while $\theta = \{W_1, b_1, W_2, b_2\}$ represents the model parameters.

We use an autoencoder to capture all log patterns of the log sequences prior to software evolution. During training, the autoencoder learns to compress the log data into a lower-dimensional space and reconstruct it back to its original form, minimizing the reconstruction error for normal logs. The reconstruction loss error is measured using the Euclidean distance, as shown in:

$$\begin{gathered}MSE(s) = \|s - \hat{s}\|^2 \\ \min_{\theta} L = \frac{1}{n} \sum_{i=1}^{n} MSE(s_i)\end{gathered} \tag{2}$$

For anomaly detection, given a log sequence s from D_test, we determine whether it is an evolved log sequence by checking if it can be captured by the trained autoencoder. Specifically, we measure the deviation of s by calculating the reconstruction loss of s through the autoencoder, as shown in the following equation:

$$\textit{Evolved Log} = \begin{cases} False & MSE(s) < \tau \\ True & MSE(s) \geq \tau \end{cases} \tag{3}$$

If the reconstruction loss exceeds the given threshold τ, it indicates that the sample deviates from the original data distribution and is considered an evolved log sequence; otherwise, it is a non-evolved log sequence. The non-evolved log sequences are then dispatched to the small model for processing, while the evolved log sequences are handled by the LLM.

B. Small Model

1) Semantic-based modeling: Traditional log anomaly detection architectures typically rely on log parsing, which

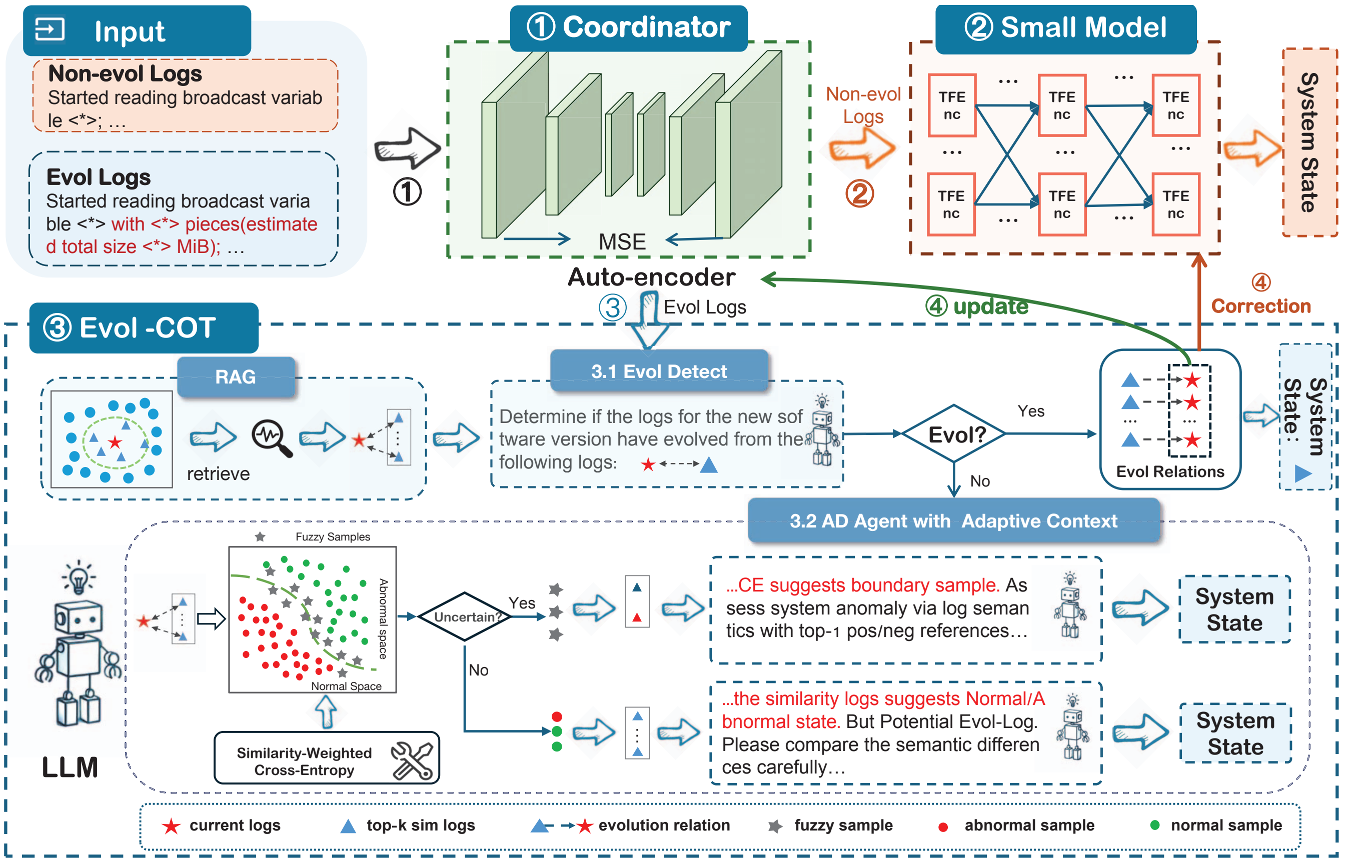

Input
Non-evol Logs
Started reading broadcast variable <*>; ...
Evol Logs
Started reading broadcast variable <*> with <*> pieces(estimated total size <*> MiB); ...
① Coordinator
MSE
Auto-encoder
② Small Model
Non-evol Logs
TFEnc
System State
④ update
④ Correction
③ Evol -COT
Evol Logs
RAG
retrieve
3.1 Evol Detect
Determine if the logs for the new software version have evolved from the following logs:
Evol?
Yes
No
Evol Relations
System State:
3.2 AD Agent with Adaptive Context
Fuzzy Samples
Abnormal space
Normal Space
Uncertain?
Similarity-Weighted Cross-Entropy
...CE suggests boundary sample. As sess system anomaly via log semantics with top-1 pos/neg references...
...the similarity logs suggests Normal/Abnormal state. But Potential Evol-Log. Please compare the semantic differences carefully...
System State
LLM
current logs
top-k sim logs
evolution relation
fuzzy sample
abnormal sample
normal sample

Fig. 5: Overview of our method: ① First, we use the coordinator, a trained autoencoder on pre-evolution data, to determine if the incoming logs have evolved; ② Non-evolved (Non-evol) logs are routed to the small model, a trained BERT-based deep neural network, for efficient and rapid processing; ③ Evolved (Evol) logs are directed to the LLM, which performs customized fine-grained inference through the Evol-CoT framework and identifies evolutionary relationships between logs; ④ Finally, the adaptive evolution mechanism (AEM) uses the evolutionary relationships identified by the LLM to update the coordinator and SM, preventing redundant inference by the LLM.

extracts log events by identifying constant parts in the logs. However, evolved logs, compared to historical logs, often undergo structural changes, making them prone to parsing errors. To overcome the limitations of traditional log architectures, we model the log anomaly detection task as a binary semantic similarity classification task, which is better suited to handle evolved logs. This approach also enables collaboration between small deep models and large language models from a semantic perspective.

2) Training with Contrastive Loss: To discriminate similar samples and enable convenient updates efficiently, we adopt a siamese network as the small model. A siamese network consists of two identical subnetworks with shared weights and parameters, which encode each input into a feature vector [48]. The similarity between two inputs is then computed using a distance metric (e.g., cosine similarity) over their embeddings. This design naturally fits our problem setting: anomaly detection in logs can be formulated as a similarity task, where normal logs and their evolved variants should lie close in the embedding space, while abnormal logs should be distant. By leveraging parameter sharing, the Siamese network provides a stable and efficient mechanism for capturing semantic similarity between log sequences, making it more robust to distribution shifts than direct classification approaches [49]. Moreover, the pair-based formulation enables continual updates, as newly identified evolution pairs can be incorporated as additional positive or negative pairs without retraining from scratch [50].

We train the model using the standard contrastive loss [51] of Siamese networks to inject the knowledge from the training set into CoorLog. The binary classification Siamese network consists of a pre-trained language model BERT [52], followed by an RNN network. First, BERT is used to extract the log semantics and convert them into feature vectors. Then, an RNN is attached to learn the temporal dependencies, i.e., the sequential features of the log sequences.

Let $D\_SM_{\text{train}} = \{s_1, s_2, \ldots, s_n | y_i \in \{0, 1\}\}$ represent the set of available log sequences for training, where s_i represents a log sequence, a set of ordered logs $s_i = \{l_1, l_2, \ldots, l_m\}$, and $y_i = \{0, 1\}$ denotes the two categories of the sequence, where 0 indicates normal and 1 indicates abnormal. Log sequences are paired, and if s_i and s_j belong to the same class, the contrastive learning process proceeds by minimizing the distance between their feature representations. Specifically, the contrastive loss is defined as:

$$L(s_i, s_j) = \begin{cases} \|\mathbf{f}(s_i) - \mathbf{f}(s_j)\|_2^2, & y_{ij} = 1 \\ \max(0, \Delta - \|\mathbf{f}(s_i) - \mathbf{f}(s_j)\|_2)^2, & y_{ij} = 0 \end{cases} \tag{4}$$

where $\mathbf{f}(s_i)$ and $\mathbf{f}(s_j)$ are the feature vectors of s_i and s_j, respectively, extracted by the BERT-based network. The y_{ij} is defined as: $y_{ij} = 1$ if $y_i = y_j$, and $y_{ij} = 0$ otherwise. Additionally, the term $\|\mathbf{f}(s_i) - \mathbf{f}(s_j)\|_2^2$ denotes the Euclidean distance between the feature vectors, while Δ is a margin parameter specifying the minimum distance between different classes. The objective is to reduce the distance between similar samples and increase the distance between dissimilar samples. During training, the network minimizes the contrastive loss by adjusting the weights to ensure that the feature vectors of similar log sequences are closer together, while those of dissimilar sequences are farther apart. This enables the network to learn discriminative features that can be used for effective log anomaly detection.

3) Anomaly Detection: After training, the small deep learning model learns to map log sequences to feature vectors that capture semantic similarities. The model's input consists of pairs of log sequences, which are processed by the BERT and RNN components to extract semantic and sequential features. The output is a similarity score, representing the degree of similarity between the two input sequences. The similarity between s_i and s_j is defined as:

$$\text{sim}(s_i, s_j; w) = \cos(\mathbf{f}(s_i), \mathbf{f}(s_j)) \tag{5}$$

Then classify an incoming log sequence s_i based on its similarity to the top-k most similar historical logs $S_{\text{top-k}} = \{s_1, s_2, \ldots, s_k\}$. For each log, calculate an anomaly detection score $S(s_i)$ using the following equation:

$$S(s_i) = \sum_{j=1}^{k} \text{sim}(s_i, s_j; w) \cdot (2y_j - 1) \tag{6}$$

We classify s_i as anomalous if $S(s_i) > 0$, and as normal otherwise.

C. Evol-CoT Framework

For evolved logs that deviate from the original data distribution, we use LLM to perceive changes in logs at the semantic level to avoid misunderstanding newly generated logs as anomalous. To mitigate the issue of LLMs' lack of domain knowledge, we use RAG, task decomposition [53], and agent technology to construct Evol-CoT.

1) Evolution Detection via LLM and RAG: Leveraging the powerful semantic understanding and reasoning capabilities of LLMs, we first employ LLMs to determine whether a log has evolved through version updates. Log evolution causes moderate distribution shifts in semantic space compared to non-evolved logs. Specifically, for an evolved log s and its pre-evolution counterpart s_{pre}, their semantic similarity $sim(s, s_{pre})$ remains relatively higher than that with other non-target logs $s_i \in \{D_{train} \mid s_i \neq s_{pre}\}$.

Based on this observation, we hypothesize that the pre-evolution logs s_{pre} of s reside among its top-k most similar historical logs. We therefore construct a candidate set of potential pre-evolution logs using Retrieval-Augmented Generation (RAG) technology:

$$S_c = \text{Top-}k\, \{s_{pre} \in D_{train} \mid \text{sim}(s, s_{pre})\} \tag{7}$$

where $\text{sim}(s, s_{pre})$ represents the semantic similarity between the potentially evolved logs s and candidate pre-evolution logs s_{pre}. The computation of $\text{sim}(s, s_{pre})$ follows Equation (5), which is the cosine similarity between feature vectors extracted by the Small Model. S_c denotes the candidate set of s's pre-evolution logs, meaning s may have evolved from one of the logs in S_c. We then employ the LLM to determine whether the current log s exhibits an evolved relationship with any log in S_c:

$$Evol(s_i, s) = \text{LLM}(\text{TD}, s, s_i, \text{sim}(s, s_i; w)), \quad s_i \in S_c \tag{8}$$

where TD represents the task description provided to the LLM. The LLM evaluates the current log s, potential pre-evolution logs s_i, and their similarity scores to make the determination. We explicitly apply chain-of-thought (CoT) prompting to encourage the LLM to reason step by step, thereby enhancing reliability [53]. Evol(s_i, s) = 1 indicates that s_i has evolved into s in the new software version (i.e., $s_i \Rightarrow s$), while Evol(s_i, s) = 0 indicates no evolved relationship between them.

If there exists s_i that forms an evolved relationship with s (denoted as $s_i \Rightarrow s$), the anomaly detection result of s_i will be inherited by s. If no such relationship is identified, the process proceeds to the next step.

2) Anomaly Detection Agent with Adaptive Context: For log sequences s that do not match an evolved relationship, we invoke a Context-adaptive Agent for anomaly detection. To enrich domain knowledge, we retrieve the top-k most similar samples $S_c = \{s_1, s_2, \ldots, s_k \mid y_i \in \{0, 1\}\}$ to provide contextual references.

The reference value of these k similar samples (S_c) varies in different situations. To guide the LLM in following the correct reasoning path under different S_c contexts, we design a context-adaptive tool that allows the Agent to dynamically adjust its reasoning strategies. Specifically, given a candidate set $S_c = \{s_1, s_2, \ldots, s_k \mid y_i \in \{0, 1\}\}$, we introduce a weighted uncertainty measure based on cross-entropy [54] and inter-sample similarity. For each $s_i \in S_c$, , y_i denotes the ground-truth label of s_i, and p_i is the proportion of samples in S_c sharing the same label as s_i. The motivation is that logs with higher semantic similarity to s should contribute more when estimating its uncertainty. Formally, the uncertainty of a target log s is computed by aggregating the uncertainty of all its neighbors in S_c, weighted by their semantic similarity:

$$U(s) = -\sum_{i=1}^{k} \frac{e^{\text{sim}(s,s_i)}}{\sum_{j=1}^{k} e^{\text{sim}(s,s_j)}} \left[y_i \log p_i + (1-y_i)\log(1-p_i) \right] \tag{9}$$

Here, $\text{sim}(s, s_i)$ represents the similarity between s and s_i, and p_i is used to estimate the predicted probability based on the proportion of samples in the current category relative to the total number of samples.

Intuitively, if most similar logs in S_c belong to the same class, s is more certain, so uncertainty $U(s)$ is low. If S_c is evenly split between normal and abnormal logs, s is more uncertain, and $U(s)$ is high. In practice, high uncertainty indicates conflicting evidence among retrieved neighbors, which may mislead the LLM if all are presented. Therefore, we provide only the most representative positive and negative examples to minimize noise and focus the LLM on the intrinsic semantics of s. Conversely, when uncertainty is low, neighbors are consistent and thus valuable; in this case, all similar samples are supplied to support fine-grained comparison.

By supplying only the most informative context with adaptive instructions, the LLM achieves more reliable semantic reasoning and anomaly detection.

D. Adaptive Evolution Mechanism

LLM-based fine reasoning remains costly. To reduce overhead, we argue that logs already processed by the LLM with highly reliable results need not be resubmitted. Thus, we propose an evolution mechanism that leverages these reliable outputs to update both the coordinator and the small model (SM). The coordinator routes recurring samples to the SM and incrementally fine-tunes the SM for accurate judgment. However, since the reliability of the LLM's anomaly detection results cannot be guaranteed, directly feeding them back risks cumulative errors. In Evol-CoT, the evolutionary relationship detection task is essentially a text similarity judgment problem, which is simpler than direct anomaly detection, with clearer objectives and targets. It instructs the LLM to match evolutionary relationships only under very clear circumstances. Therefore, its results are more reliable. Hence, we utilize the evolution log relationships clearly identified by the LLM (i.e., $D_{evol} = \{(s_i, s_j) \mid Evol(s_i, s_j) = 1\}$) to update the coordinator and correct the Small Model.

1) Update Coordinator: Samples that have been clearly identified as evolved logs by the LLM should be routed to the small model again through the Coordinator. Therefore, the logs identified as evolved (i.e., $\{s_i, s_j \mid Evol(s_i, s_j) = 1\}$) are collected to fine-tune the trained autoencoder using the reconstruction loss shown in Equation (2).

2) Correcting Small Model: The Small Model should possess the ability to correctly discriminate new samples routed by the updated coordinator or maintain consistency with the LLM's discrimination results, avoiding degradation in accuracy. Therefore, continue to use $(s_i, s_j) \in D_{evol}$ as a similar sample set to fine-tune SM using the contrastive loss shown in Equation (4). This approach enables the SM to have the capability to discriminate evolved logs.

The task of determining textual associations is simpler than anomaly detection, and we constrain the decision rules strictly to ensure reliability. Moreover, the primary purpose of AEM is to avoid redundant LLM usage. Even if the LLM occasionally fails in its judgment, resubmitting the same case would yield the same result, meaning that AEM does not compromise the original detection capability.

IV. Experiment and Evaluation

A. Datasets and Experiment Settings

We conduct extensive experiments on LOGEVOL [2](https://github.com/YintongHuo/EvLog), a publicly available dataset that records software evolution activities. LOGEVOL is generated using the HiBench benchmarking suite [55]. LOGEVOL specifically selects two major versions of Spark (Spark 2.4.0 and Spark 3.0.3) and Hadoop (Hadoop 2.10.2 and Hadoop 3.3.3) due to their significant structural changes, making them ideal for studying log evolution. The dataset consists of over 6.7 million log messages, among which approximately 69,513 are labeled as anomalous logs.

We configure the experimental setup as follows. The coordinator is a deep autoencoder composed of three fully connected layers. The threshold τ is determined by traversing the percentile of reconstruction loss in the valid set, and we select the value at the inflection point of the F1-score change (i.e., Q90 for Spark and Q90 for Hadoop). The Small Model is constructed with a BERT followed by a bidirectional LSTM (hidden size 128), a fully connected layer (size 64), and a final feature vector (dimension 32). During training, the coordinator is trained for 40 epochs, and the SM for 20 epochs. In AEM updates, the coordinator is fine-tuned for 4 epochs and the SM for 2 epochs. The batch size is set to 256 in all cases, and the Adam optimizer is used with a learning rate of 0.001. We set $k = 4$ for Spark and $k = 5$ for Hadoop. For the LLM, we use Qwen-plus [56] as the base LLM, and set the temperature to 0.01 and top-p to 0.95. In our prompts, we consistently guide the LLM to reason step by step (CoT) to better leverage its reasoning capability [53]. The code, dataset, detailed implementations are available at https://github.com/XiaoPeiCode/CoorLog.git.

All experiments are conducted on a Linux server equipped with an Intel(R) Xeon(R) Gold 6126T 2.60GHz CPU, 128GB of RAM, and four RTX A4000 GPUs, each with 16GB of GPU memory.

B. Baseline Methods and Metrics

To evaluate the effectiveness, we compare CoorLog against a diverse set of baselines: traditional statistical methods (LogSed), unsupervised models (DeepLog, LogAnomaly), supervised models (BERT, LogRobust), semi-supervised methods (LogOnline, LogBERT), and LLM-based approaches (LLMeLog). We also include EvLog, which targets software evolution scenarios. More details are as follows:

TABLE I: Comparison of different methods (train set→test set).

LOGEVOL-HADOOP												
	Intra-version						**Inter-version**					
	Hadoop 2 → Hadoop 2			Hadoop 3 → Hadoop 3			Hadoop 2 → Hadoop 3			Hadoop 3 → Hadoop 2		
Method	Pr	Re	F1	Pr	Re	F1	Pr	Re	F1	Pr	Re	F1
LogSed	0.910	**0.995**	0.951	0.925	0.986	0.955	0.371	0.988	0.540	0.390	0.993	0.560
DeepLog	0.913	0.985	0.947	0.926	**1.000**	0.961	0.386	**0.999**	0.556	0.410	0.971	0.576
LogAnomaly	0.926	0.994	0.958	0.939	0.988	0.963	0.389	0.998	0.560	0.407	**0.995**	0.578
BERT	0.928	0.731	0.817	0.959	0.837	0.894	0.865	0.706	0.778	0.952	0.763	0.847
LogRobust	0.935	0.981	0.957	0.948	0.983	0.965	0.782	0.824	0.803	0.813	0.846	0.829
LogBERT	0.941	0.977	0.959	0.953	0.987	0.970	0.875	0.852	0.863	0.898	0.871	0.884
LogOnline	0.948	0.984	0.966	0.963	0.989	0.976	0.893	0.895	0.894	0.913	0.908	0.911
LLMeLog	0.952	0.967	0.959	0.963	0.975	0.969	0.912	0.923	0.917	0.928	0.934	0.931
EvLog	0.945	0.982	0.963	0.952	0.988	0.970	0.770	0.941	0.847	0.857	0.913	0.884
Our	**0.993**	0.968	**0.980**	**0.997**	0.982	**0.990**	**0.946**	0.983	**0.964**	**0.994**	0.957	**0.975**

LOGEVOL-SPARK												
	Intra-version						**Inter-version**					
	Spark 2 → Spark 2			Spark 3 → Spark 3			Spark 2 → Spark 3			Spark 3 → Spark 2		
Method	Pr	Re	F1	Pr	Re	F1	Pr	Re	F1	Pr	Re	F1
LogSed	0.842	0.914	0.877	0.907	0.923	0.915	0.013	0.917	0.026	0.010	0.914	0.020
DeepLog	0.862	**0.952**	0.905	0.858	**0.976**	0.914	0.017	**0.947**	0.032	0.014	0.909	0.026
LogAnomaly	0.931	0.939	0.935	0.898	0.947	0.922	0.020	0.923	0.038	0.017	**0.948**	0.034
BERT	0.943	0.750	0.835	**1.000**	0.684	0.812	0.550	0.696	0.615	**1.000**	0.568	0.715
LogRobust	0.949	0.837	0.889	0.974	0.857	0.912	0.732	0.753	0.742	0.934	0.783	0.851
LogBERT	0.948	0.875	0.909	0.973	0.886	0.927	0.805	0.813	0.809	0.949	0.822	0.881
LogOnline	0.954	0.904	0.928	0.986	0.899	0.940	0.843	0.864	0.853	0.957	0.875	0.914
LLMeLog	0.959	0.920	0.938	0.950	0.857	0.900	0.840	0.799	0.828	0.762	0.934	0.851
EvLog	0.970	**0.974**	**0.972**	0.944	0.888	0.915	0.922	0.700	0.795	0.920	0.812	0.863
Our	**0.976**	0.932	0.954	0.979	0.918	**0.947**	**0.904**	0.855	**0.879**	0.968	0.904	**0.935**

- **LogSed** [57] extracts key sequential relationships from interleaved logs using a two-stage approach.
- **DeepLog** [58] models next-event prediction in log sequences, treating mismatches as anomalies.
- **LogAnomaly** [59] employs an attention-based LSTM to integrate semantic representations for unsupervised detection.
- **LogRobust** [60] applies supervised learning with semantic vectors and attention to capture contextual information.
- **BERT** [52] fine-tunes pre-trained BERT for binary log anomaly classification, serving as a strong baseline.
- **LogBERT** [61] uses self-supervised BERT to detect deviations by learning normal log patterns.
- **LLMeLog** [62] leverages LLMs to enrich logs with semantic context for anomaly detection.
- **LogOnline** [21] adapts to evolving logs via semi-supervised online learning without manual labels.
- **EvLog** [2] targets software evolution with parser-free multi-level semantics and an attention-based discriminator.

To assess each component's impact, we evaluate ablation variants in Table II: **Our$_{\textbf{SM}}$** uses only the trained SM (② in Figure 5), while **Evol-CoT** represents the LLM-based part of our method (③ in Figure 5). Within Evol-CoT, **Evol Detect** and **AD Agent** correspond to its two internal components (3.1 and 3.2 in Figure 5). **Vanilla** denotes a baseline that employs LLMs without our Evol-CoT strategy, using only the enhancement techniques from prior work [30], [41]. The **+AEM** setting in Table II and III is used to study how the AEM (④ in Figure 5) affects the coordinator's routing decisions and performance. In Table V, the variant **(w/o Coord)** refers to the setting where our coordinator is removed and all logs are processed directly by the LLM.

The evaluation metrics include the commonly used F1-score (**F1**), Precision (**Pr**), and Recall (**Re**). We also report **Proportion**, denoting the ratio of samples in the entire test set, to quantify workload distribution among components. The accuracy of identifying evolutionary relationships (**Accuracy**) is assessed by checking the label consistency between pre-evolution and post-evolution logs. Finally, to evaluate the cost savings introduced by the coordinator, we measure Total LLM Calls **(Calls)**, Average Processing Time per sample **(Time)**, and Total Token Consumption **(Token)**. For average processing time, our method also accounts for the computational overhead of the small model.

C. Main Results

To evaluate our approach, we conduct experiments under two different settings: (1) Intra-version, where anomaly detection is performed within the same software version (e.g., Hadoop 2 → Hadoop 2), and (2) Inter-version, where anomaly detection is performed across different software versions (e.g., Hadoop 2 → Hadoop 3). As shown in Table I, our method achieves the highest F1-score across both datasets in both

TABLE II: Ablation study on different log categories (*Evol* and *Non-evol* denote the evolved and non-evolved logs identified by the coordinator, respectively.)

Logs	Method	Spark 2 → 3			Hadoop 2 → 3		
		Pr	Re	F1	Pr	Re	F1
Non-evol	**Our$_{SM}$**	0.950	0.901	0.925	0.985	0.979	0.982
	+AEM	0.935	0.925	0.930	0.955	0.943	0.949
Evol	**Our$_{SM}$**	0.468	0.417	0.441	0.375	0.402	0.387
	+AEM	0.640	0.610	0.625	0.537	0.489	0.512
	Vanilla	0.785	0.209	0.330	0.705	0.518	0.595
	Evol-CoT	0.829	0.913	0.870	0.944	0.885	0.914
ALL	**Our$_{SM}$**	0.698	0.848	0.766	0.893	0.861	0.876
	+AEM	0.721	0.867	0.794	0.895	0.872	0.883
	Our	0.904	0.855	0.879	0.946	0.983	0.964

settings. We observe that existing anomaly detection methods generally perform well in the intra-version setting. For example, in the Hadoop 3 → Hadoop 3 setting, most of them achieve an F1-score above 90%. However, in the inter-version setting, most methods suffer significant performance degradation due to their reliance on log parsing and the closed-world assumption. Specifically, LogSed, DeepLog, LogAnomaly, and LogOnline heavily depend on structured log parsing. As log entries evolve across versions, this dependency leads to a high false-negative rate, resulting in poor Precision—often below 5%. LogBERT, EvLog, and BERT, on the other hand, avoid reliance on log parsing but still depend on the training distribution. Consequently, they struggle to generalize to concept-drifted samples, leaving room for improvement in Recall. The core of LLMeLog is to utilize LLM for the semantic alignment of log templates, which can alleviate log entry evolution to a certain extent. However, it still relies on log template extraction and has not addressed the issue of log sequence evolution, thus its performance in inter-version evolution remains suboptimal.

In our method, the small model formulates log anomaly detection as a semantic binary classification task, transforming discrete log events into continuous semantic representations, which can avoid false positives caused by log parsing. Most importantly, our collaborative design lets the small model handle in-distribution samples, while the large model focuses on those with concept drift. The LLM, through our constructed Evol-CoT, utilizes RAG and Agent techniques to conduct detailed semantic comparative analysis and contextual reasoning, enabling better anomaly detection for evolving logs. This significantly mitigates the impact of log sequence evolution on anomaly detection performance. As a result, our method outperforms EvLog by over 10% in F1-score on the Hadoop dataset and achieves the best overall results on the Spark dataset. These results demonstrate that our hybrid approach effectively balances efficiency and generalization, making it a robust solution for log-based anomaly detection in evolving software systems.

TABLE III: Proportion (relative to the test set) and performance across different components.

Logs	Method	Spark 2 → 3		Hadoop 2 → 3	
		Proportion	F1	Proportion	F1
Non-evol	**Our$_{SM}$**	94.84%	0.925	95.18%	0.982
	+AEM	95.79%	0.930	97.56%	0.949
Evol	**Evol-CoT**	5.16%	0.870	4.82%	0.914
	Evol Detect	5.16%	0.893	4.82%	0.857
	AD Agent	1.65%	0.829	1.72%	0.948

TABLE IV: Statistics of identified evolution relations, including their count, detection accuracy, and average per-sample training time of AEM.

	#Relations	Accuracy	AEM Time (ms)
Spark 2 → 3	288	0.993	0.18
Hadoop 2 → 3	541	0.986	0.02

D. Ablation Study

After presenting the main results, we further analyze the contribution of each component through ablation studies.

*1) **Effect of Coordinator**:* The coordinator aims to identify logs that the small model can handle well, avoiding unnecessary LLM usage. As shown in Table II and III, the coordinator assigns more than 90% of the non-evolved logs (e.g., 94.84% in Spark and 95.18% in Hadoop) to the SM, which achieves high F1-scores of 0.925 and 0.982 on these logs. In contrast, the SM's performance drops sharply on the remaining evolved logs, confirming that the coordinator successfully identifies samples where the SM is unreliable. The coordinator identifies evolved logs by checking whether the autoencoder's reconstruction loss exceeds a threshold τ, which is set based on different percentiles of the valid set losses. As shown in Figure 6, higher τ values lead to selected logs that deviate more from the training data, confirming that the coordinator correctly routes logs where the SM struggles, thereby validating its effectiveness. Moreover, τ controls the routing load to the LLM: small values increase cost, while large values impair routing and reduce performance. Based on this analysis, we select τ at the inflection point (i.e., Q90 for both Spark and Hadoop), where the SM's F1-score on evolved logs begins to decline.

*2) **Effect of Small Model**:* As shown in Table II, the small model (Our$_{SM}$) performs well on non-evolved logs, with F1-scores exceeding 90%. However, its performance degrades notably on evolved logs. Our method explicitly identifies evolved samples caused by concept drift and leverages the reasoning capability of the LLM to determine anomalies from a semantic perspective. Compared to relying solely on the small model, this design effectively improves detection. As presented in Table II, our approach consistently outperforms the small model. We further analyze the impact of the key parameter k in Figure 6. Similar trends are observed across both datasets: when $k = 1$, performance is poor due to insufficient reference samples. As k increases, performance improves, but overly large k values introduce dissimilar samples, degrading the effectiveness of majority voting. We select the optimal k based on the highest F1-score on the validation set for both anomaly detection and RAG retrieval.

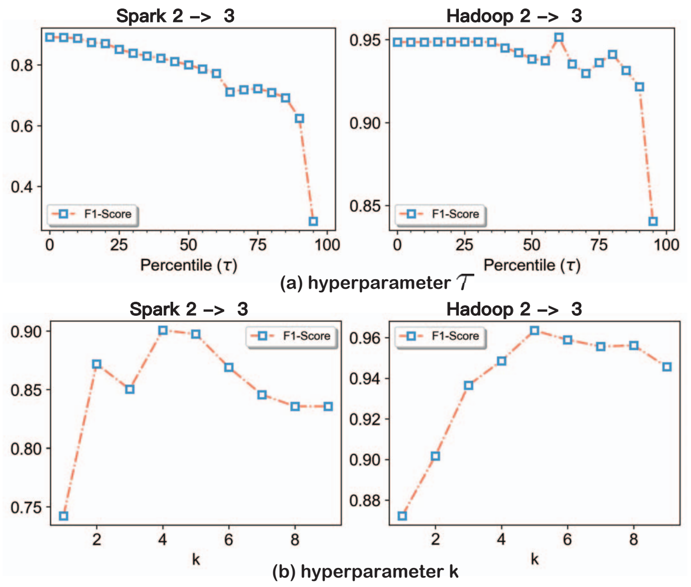


Fig. 6: Ablation study on hyperparameter (a) τ and (b) k.

*3) **Effect of Evol-CoT**:* Table III reports the proportions of logs processed by different components of our method, along with their corresponding performance. In Spark 2→3 and Hadoop 2→3, 5.16% and 4.82% of the samples are identified as evolved logs and routed to Evol-CoT. These samples are subject to concept drift, where the small model fails to maintain reliable performance. Moreover, directly applying the LLM for anomaly detection (i.e., Vanilla) performs poorly due to insufficient contextual information, achieving only 0.330 F1 on the Spark dataset. In contrast, Evol-CoT customizes log anomaly detection for software evolution scenarios through task decomposition and RAG-based knowledge injection, yielding a substantial F1-score improvement.

Evol-CoT consists of two key components: *Evol Detect* and the *AD Agent*. All evolved logs are first passed to Evol Detect, which, together with the top-k retrieved historical logs from RAG, attempts to identify evolution relations. If such a relation is found, the class label of the pre-evolution log is directly reused as the detection result. Otherwise, the input proceeds to the AD Agent for anomaly detection. As shown in Table III, 3.51% and 3.10% of logs are recognized as evolution cases, while the remaining logs (1.65% in Spark and 1.72% in Hadoop) are handled by the AD Agent. Both parts achieve F1-scores above 80%, benefiting from task decomposition: Evol Detect only needs to determine semantic relations between logs, reducing task complexity, while the AD Agent performs anomaly detection with reliable reasoning by leveraging RAG to provide the most relevant information and adaptive contexts tailored to specific scenarios.

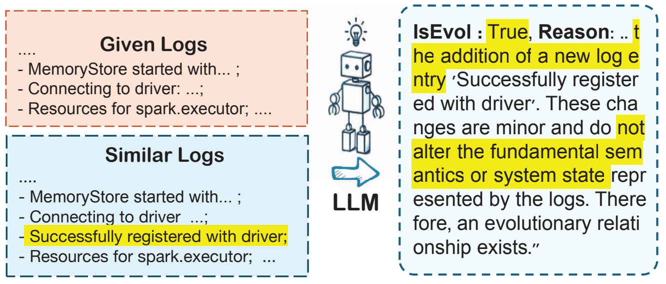
*4) **Effect of AEM**:* AEM leverages the evolution relations identified by Evol Detect to update both the coordinator and the SM, thereby avoiding redundant LLM usage for similar samples. Table IV reports the number of evolution relations used by AEM, the detection accuracy of these relations, and the average per-sample update time. The accuracy, measured by label consistency between pre-evolution and post-evolution logs, reaches 99% on Spark and 98.6% on Hadoop, demonstrating the reliability of LLM-based relation identification. Furthermore, the update time is negligible (0.18 ms and 0.02 ms), especially compared to LLM inference. Moreover, even if the LLM occasionally makes an incorrect judgment, reprocessing the same log would produce the same result, meaning that AEM does not compromise overall detection capability. As shown in Table III, applying AEM increases the proportion of logs routed to the small model (e.g., from 94.84% to 95.79% on Spark), which correspondingly reduces the proportion routed to the LLM (e.g., from 5.16% to 4.21%). This shift alleviates redundant LLM usage and thereby improves overall efficiency. At the same time, Table II shows that AEM substantially improves the detection of evolved logs (e.g., F1-score on Spark increases from 0.441 to 0.625), while keeping the detection of non-evolved logs nearly unaffected. Consequently, AEM delivers consistent overall improvements in F1-score across both datasets. Moreover, even if the LLM occasionally makes an incorrect decision, reprocessing the same log would reproduce the same outcome, ensuring that AEM does not compromise detection reliability.

TABLE V: Comparison of efficiency and computational cost.

Metric	Method	Spark 2 → 3	Hadoop 2 → 3
Calls	w/o coord	4,246	34,302
	Our	289	2,174
Token (k)	w/o coord	9,048.85	21,347.73
	Our	1,557.30	2,480.65
Time (ms)	LogRobust	0.96	0.43
	LogAnomaly	1.69	0.83
	Our$_{SM}$	0.23	0.04
	w/o coord	810.10	724.61
	Our	78.40	54.30

E. Efficiency and Cost Analysis

Table V compares the efficiency and cost of our method against directly using an LLM for anomaly detection without coordinator collaboration (w/o coord). The results demonstrate a significant reduction in computational overhead achieved by our approach. Our method reduces the number of LLM calls by over 85%, thereby minimizing the reliance on costly inference operations. This leads to a substantial decrease in both the average processing time per log sequence and token consumption, making our approach more practical for real-world deployment. By reducing the number of LLM calls, the average processing time per sample is reduced by over 90%. Similarly, token consumption is reduced by more than 80% (up to 88%), saving millions of tokens. Since LLMs inherently suffer from higher latency due to their large model size and network overhead, our method still shows an efficiency gap compared to lightweight baselines such as LogAnomaly. However, the coordinator design makes LLM-based approaches more practical for meeting industrial-speed requirements.



Given Logs

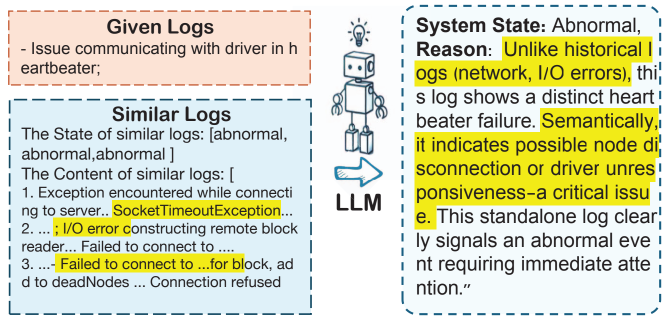
....
- MemoryStore started with... ;
- Connecting to driver: ...;
- Resources for spark.executor; ....

Similar Logs

....
- MemoryStore started with... ;
- Connecting to driver ...;
- Successfully registered with driver;
- Resources for spark.executor; ...

IsEvol : True, **Reason:** .. the addition of a new log entry 'Successfully registered with driver'. These changes are minor and do not alter the fundamental semantics or system state represented by the logs. Therefore, an evolutionary relationship exists."

(a) Case study on 3.1 Evol Detect



Given Logs

- Issue communicating with driver in heartbeater;

Similar Logs

The State of similar logs: [abnormal, abnormal,abnormal]
The Content of similar logs: [
1. Exception encountered while connecting to server.. SocketTimeoutException...
2. ... ; I/O error constructing remote block reader... Failed to connect to ....
3. ...- Failed to connect to ...for block, add to deadNodes ... Connection refused

System State: Abnormal,
Reason: Unlike historical logs (network, I/O errors), this log shows a distinct heartbeater failure. Semantically, it indicates possible node disconnection or driver unresponsiveness–a critical issue. This standalone log clearly signals an abnormal event requiring immediate attention."

(b) Case study on 3.2 AD Agent

Fig. 7: Case studies on Evol-CoT: (a) Evol Detect and (b) AD Agent.

F. Case Study

Evol-CoT integrates a customized workflow with CoT prompts, enabling the LLM to provide explicit reasoning traces. To illustrate this, we present two representative cases corresponding to its two core components:

1) Case of Evol Detect (3.1): As shown in Fig. 7(a), the current log differs from its most similar historical log only by an additional entry serving as supplementary information. The LLM explicitly identifies this difference, reasons that the added entry does not alter the underlying system semantics or state, and concludes that the two logs are evolutionarily related. Consequently, the anomaly label of the historical log can be reused, with the reasoning trace providing a clear justification for this decision.

2) Case of AD Agent (3.2): Fig. 7(b) illustrates the complementary scenario. When the current log exhibits substantial semantic differences from similar historical logs, no evolutionary relation is detected, and the sample is routed to the AD Agent. The agent first employs tools to check determinism (i.e., whether similar logs are consistently labeled as anomalies), and then applies RAG to retrieve informative examples. Guided by CoT, the LLM contrasts the current log with retrieved anomalies (e.g., network or I/O errors), highlights semantic differences (e.g., heartbeat failure vs. connection errors), and shifts to semantic reasoning to interpret the failure mode. This process leads to correct anomaly detection while simultaneously providing an interpretable rationale.

V. Related Work

Log-based anomaly detection is essential for ensuring system reliability and security by capturing abnormal patterns that may indicate potential failures. Traditional approaches range from rule-based methods [63], [64] to machine learning (ML) [11] and deep learning (DL) [10], [13], [57], [65]–[68], with DL models [18], [60], [69] showing strong capability in learning complex log patterns. However, most training-based methods rely on a closed-world assumption, limiting generalization under software evolution. Log entry and sequence evolution cause distribution shifts that lead to parsing errors and misclassifications. Recent efforts, such as EvLog [2] and LogOnline [21], address log entry evolution by directly learning semantic embeddings from raw logs, reducing parsing inconsistencies but remaining ineffective against concept drift from sequence evolution. Moreover, DL-based methods generally suffer from poor interpretability.

More recently, LLM-based approaches have emerged, but they often suffer from efficiency issues. Some enhance log parsing [62], while others use LLMs for post-hoc verification or interpretability based on deep learning model results [41], [44], [45]. In essence, these methods are still small model–dominated, with LLM-based detection heavily reliant on prior outputs from the small model. As a result, when concept drift degrades small model reliability, overall detection performance cannot be guaranteed. In contrast, our method employs an adaptive coordinator to classify logs upfront, enabling the LLM to operate independently when necessary, thus improving robustness and flexibility.

VI. Conclusion

In this paper, we propose an efficient and generalizable anomaly detection scheme using an adaptive coordinator to enable collaboration between large language models (LLMs) and small models (SMs). The SM handles non-evolved logs within the training distribution, while the LLM processes evolved logs via the Evol-CoT framework. Furthermore, the Adaptive Evolution Mechanism (AEM) ensures reliable LLM decisions are routed to the SM, reducing redundant LLM use. Future work will explore different model combinations and validate their effectiveness in various scenarios.

Acknowledgment

This work was supported by the PKU-ByteDance Cooperation Research Project.

References

[1] S. Elliot, "Devops and the cost of downtime: Fortune 1000 best practice metrics quantified," *International Data Corporation (IDC)*, 2014.

[2] Y. Huo, C. Lee, Y. Su, S. Shan, J. Liu, and M. R. Lyu, "Evlog: Identifying anomalous logs over software evolution," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2023, pp. 391–402.

[3] M. M. Lehman and J. F. Ramil, "Software evolution and software evolution processes," *Ann. Softw. Eng.*, vol. 14, no. 1-4, pp. 275–309, 2002. [Online]. Available: https://doi.org/10.1023/A:1020557525901

[4] W. Shang, Z. M. Jiang, B. Adams, A. E. Hassan, M. W. Godfrey, M. N. Nasser, and P. Flora, "An exploratory study of the evolution of communicated information about the execution of large software systems," *J. Softw. Evol. Process.*, vol. 26, no. 1, pp. 3–26, 2014. [Online]. Available: https://doi.org/10.1002/smr.1579

[5] D. Yuan, S. Park, and Y. Zhou, "Characterizing logging practices in open-source software," in *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 2012, pp. 102–112.

[6] B. Chen and Z. M. Jiang, "Characterizing logging practices in java-based open source software projects–a replication study in apache software foundation," *Empirical Software Engineering*, vol. 22, pp. 330–374, 2017.

[7] W. Xu, L. Huang, A. Fox, D. A. Patterson, and M. I. Jordan, "Online system problem detection by mining patterns of console logs," in *ICDM 2009, The Ninth IEEE International Conference on Data Mining, Miami, Florida, USA, 6-9 December 2009*. IEEE Computer Society, 2009, pp. 588–597. [Online]. Available: https://doi.org/10.1109/ICDM.2009.19

[8] H. Amar, L. Bao, N. Busany, D. Lo, and S. Maoz, "Using finite-state models for log differencing," in *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE, Lake Buena Vista, FL, USA, November 04-09, 2018*. ACM, 2018, pp. 49–59. [Online]. Available: https://doi.org/10.1145/3236024.3236069

[9] H. Wang, Z. Wu, H. Jiang, Y. Huang, J. Wang, S. Köprü, and T. Xie, "Groot: An event-graph-based approach for root cause analysis in industrial settings," in *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering, ASE, Melbourne, Australia, November 15-19, 2021*. IEEE, 2021, pp. 419–429. [Online]. Available: https://doi.org/10.1109/ASE51524.2021.9678708

[10] T. Jia, Y. Li, Y. Yang, and G. Huang, "Hilogx: noise-aware log-based anomaly detection with human feedback," *The VLDB Journal*, vol. 33, no. 3, pp. 883–900, 2024.

[11] L. G. Mandagondi, "Anomaly detection in log files using machine learning techniques," 2021.

[12] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 1285–1298.

[13] B. Yu, J. Yao, Q. Fu, Z. Zhong, H. Xie, Y. Wu, Y. Ma, and P. He, "Deep learning or classical machine learning? an empirical study on log-based anomaly detection," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, 2024, pp. 1–13.

[14] C. Zhang, T. Jia, G. Shen, P. Zhu, and Y. Li, "Metalog: Generalizable cross-system anomaly detection from logs with meta-learning," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–12.

[15] C. Duan, T. Jia, Y. Li, and G. Huang, "Aclog: An approach to detecting anomalies from system logs with active learning," in *2023 IEEE International Conference on Web Services (ICWS)*. IEEE, 2023, pp. 436–443.

[16] P. Xiao, T. Jia, C. Duan, H. Cai, Y. Li, and G. Huang, "Logcae: An approach for log-based anomaly detection with active learning and contrastive learning," in *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2024, pp. 144–155.

[17] J. Tong, L. Ying, T. Hongyan, and W. Zhonghai, "An approach to pinpointing bug-induced failure in logs of open cloud platforms," in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. IEEE, 2016, pp. 294–302.

[18] C. Duan, T. Jia, H. Cai, Y. Li, and G. Huang, "Afalog: A general augmentation framework for log-based anomaly detection with active learning," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2023, pp. 46–56.

[19] T. Jia, Y. Li, Y. Yang, G. Huang, and Z. Wu, "Augmenting log-based anomaly detection models to reduce false anomalies with human feedback," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 3081–3089.

[20] M. He, C. Duan, P. Xiao, T. Jia, S. Yu, L. Zhang, W. Hong, J. Han, Y. Wu, Y. Li *et al.*, "United we stand: Towards end-to-end log-based fault diagnosis via interactive multi-task learning," *arXiv preprint arXiv:2509.24364*, 2025.

[21] X. Wang, J. Song, X. Zhang, J. Tang, W. Gao, and Q. Lin, "Logonline: A semi-supervised log-based anomaly detector aided with online learning mechanism," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2023, pp. 141–152.

[22] M. Du, Z. Chen, C. Liu, R. Oak, and D. Song, "Lifelong anomaly detection through unlearning," in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, 2019, pp. 1283–1297.

[23] S. Frieder, L. Pinchetti, R.-R. Griffiths, T. Salvatori, T. Lukasiewicz, P. Petersen, and J. Berner, "Mathematical capabilities of chatgpt," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[24] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao, "Large language models: A survey," *arXiv preprint arXiv:2402.06196*, 2024.

[25] B. Zhang, B. Haddow, and A. Birch, "Prompting large language model for machine translation: A case study," in *International Conference on Machine Learning*. PMLR, 2023, pp. 41 092–41 110.

[26] C. Liu, W. Zhang, Y. Zhao, A. T. Luu, and L. Bing, "Is translation all you need? a study on solving multilingual tasks with large language models," *arXiv preprint arXiv:2403.10258*, 2024.

[27] Y. Zhang, P. Xiao, L. Wang, C. Zhang, M. Fang, Y. Du, Y. Puzyrev, R. Yao, S. Qin, Q. Lin *et al.*, "Ruag: Learned-rule-augmented generation for large language models," *arXiv preprint arXiv:2411.03349*, 2024.

[28] L. Zhang, L. Fang, C. Duan, M. He, L. Pan, P. Xiao, S. Huang, Y. Zhai, X. Hu, P. S. Yu *et al.*, "A survey on parallel text generation: From parallel decoding to diffusion language models," *arXiv preprint arXiv:2508.08712*, 2025.

[29] J. Qi, S. Huang, Z. Luan, S. Yang, C. Fung, H. Yang, D. Qian, J. Shang, Z. Xiao, and Z. Wu, "Loggpt: Exploring chatgpt for log-based anomaly detection," in *2023 IEEE International Conference on High Performance Computing & Communications, Data Science & Systems, Smart City & Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*. IEEE, 2023, pp. 273–280.

[30] Y. Liu, S. Tao, W. Meng, F. Yao, X. Zhao, and H. Yang, "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*, 2024, pp. 364–365.

[31] R. Li, Q. Li, Y. Zhang, D. Zhao, Y. Jiang, and Y. Yang, "Interpreting unsupervised anomaly detection in security via rule extraction," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[32] W. Zhang, Q. Zhang, E. Yu, Y. Ren, Y. Meng, M. Qiu, and J. Wang, "Leveraging rag-enhanced large language model for semi-supervised log anomaly detection," in *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2024, pp. 168–179.

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

[34] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[35] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*. Ieee, 2010, pp. 1–10.

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

[37] V. D. M. Laurens and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. 2605, pp. 2579–2605, 2008.

[38] Z. Ji, T. Yu, Y. Xu, N. Lee, E. Ishii, and P. Fung, "Towards mitigating llm hallucination via self reflection," in *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023, pp. 1827–1843.

[39] D. McDonald, R. Papadopoulos, and L. Benningfield, "Reducing llm hallucination using knowledge distillation: A case study with mistral large and mmlu benchmark," *Authorea Preprints*, 2024.

[40] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020.

[41] W. Zhang, Q. Zhang, E. Yu, Y. Ren, Y. Meng, M. Qiu, and J. Wang, "Lograg: Semi-supervised log-based anomaly detection with retrieval-augmented generation," in *2024 IEEE International Conference on Web Services (ICWS)*. IEEE, 2024, pp. 1100–1102.

[42] J. Pan, S. L. Wong, and Y. Yuan, "Raglog: Log anomaly detection using retrieval augmented generation," *arXiv preprint arXiv:2311.05261*, 2023.

[43] C. Duan, T. Jia, Y. Yang, G. Liu, J. Liu, H. Zhang, Q. Zhou, Y. Li, and G. Huang, "Eagerlog: Active learning enhanced retrieval augmented generation for log-based anomaly detection," in *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2025, pp. 1–5.

[44] P. Xiao, T. Jia, C. Duan, M. He, W. Hong, X. Yang, Y. Wu, Y. Li, and G. Huang, "Clslog: Collaborating large and small models for log-based anomaly detection," in *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*, 2025, pp. 686–690.

[45] L. Ma, W. Yang, Y. Li, B. Fei, M. Zhou, S. Li, S. Jiang, B. Xu, and Y. Xiao, "Adaptivelog: An adaptive log analysis framework with the collaboration of large and small language model," *ACM Transactions on Software Engineering and Methodology*, 2025.

[46] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2018.

[47] A. Ng *et al.*, "Sparse autoencoder," *CS294A Lecture notes*, vol. 72, no. 2011, pp. 1–19, 2011.

[48] I. Melekhov, J. Kannala, and E. Rahtu, "Siamese network features for image matching," in *2016 23rd international conference on pattern recognition (ICPR)*. IEEE, 2016, pp. 378–383.

[49] S. Cheng, B. Zhong, G. Li, X. Liu, Z. Tang, X. Li, and J. Wang, "Learning to filter: Siamese relation network for robust tracking," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 4421–4431.

[50] H. Zhang, L. Cao, Y. Yan, S. Madden, and E. A. Rundensteiner, "Continuously adaptive similarity search," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 2601–2616.

[51] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, "Supervised contrastive learning," *Advances in neural information processing systems*, vol. 33, pp. 18 661–18 673, 2020.

[52] M. V. Koroteev, "Bert: a review of applications in natural language processing and understanding," *arXiv preprint arXiv:2103.11943*, 2021.

[53] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[54] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Annals of operations research*, vol. 134, pp. 19–67, 2005.

[55] "Hibench," Intel, 2021. [Online]. Available: https://github.com/Intel-bigdata/HiBench

[56] J. Bai, S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge, Y. Han, F. Huang *et al.*, "Qwen technical report," *arXiv preprint arXiv:2309.16609*, 2023.

[57] T. Jia, L. Yang, P. Chen, Y. Li, F. Meng, and J. Xu, "Logsed: Anomaly diagnosis through mining time-weighted control flow graph in logs," in *Proceedings of the 10th IEEE International Conference on Cloud Computing, CLOUD, Honolulu, HI, USA, June 25-30, 2017*. IEEE Computer Society, 2017, pp. 447–455. [Online]. Available: https://doi.org/10.1109/CLOUD.2017.64

[58] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 24th ACM SIGSAC Conference on Computer and Communications Security, CCS, Dallas, TX, USA, October 30 - November 03, 2017*. ACM, 2017, pp. 1285–1298. [Online]. Available: https://doi.org/10.1145/3133956.3134015

[59] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun, and R. Zhou, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI, Macao, China, August 10-16, 2019*, S. Kraus, Ed. ijcai.org, 2019, pp. 4739–4745. [Online]. Available: https://doi.org/10.24963/ijcai.2019/658

[60] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, J. Chen, X. He, R. Yao, J. Lou, M. Chintalapati, F. Shen, and D. Zhang, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE, Tallinn, Estonia, August 26-30, 2019*. ACM, 2019, pp. 807–817. [Online]. Available: https://doi.org/10.1145/3338906.3338931

[61] H. Guo, S. Yuan, and X. Wu, "Logbert: Log anomaly detection via bert," in *2021 international joint conference on neural networks (IJCNN)*. IEEE, 2021, pp. 1–8.

[62] M. He, T. Jia, C. Duan, H. Cai, Y. Li, and G. Huang, "Llmelog: An approach for anomaly detection based on llm-enriched log events," in *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2024, pp. 132–143.

[63] J. Breier and J. Branišová, "A dynamic rule creation based anomaly detection method for identifying security breaches in log records," *Wireless Personal Communications*, vol. 94, pp. 497–511, 2017.

[64] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection," in *2010 USENIX Annual Technical Conference (USENIX ATC 10)*, 2010.

[65] T. Jia, Y. Wu, C. Hou, and Y. Li, "Logflash: Real-time streaming anomaly detection and diagnosis from system logs for large-scale software systems," in *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2021, pp. 80–90.

[66] A. Nandi, A. Mandal, S. Atreja, G. B. Dasgupta, and S. Bhattacharya, "Anomaly detection using program control flow graph mining from execution logs," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 215–224.

[67] T. Jia, P. Chen, L. Yang, Y. Li, F. Meng, and J. Xu, "An approach for anomaly diagnosis based on hybrid graph model with logs for distributed services," in *Proceedings of the 24th IEEE International Conference on Web Services, ICWS, Honolulu, HI, USA, June 25-30, 2017*. IEEE, 2017, pp. 25–32. [Online]. Available: https://doi.org/10.1109/ICWS.2017.12

[68] M. He, T. Jia, C. Duan, P. Xiao, L. Zhang, K. Wang, Y. Wu, Y. Li, and G. Huang, "Walk the talk: Is your log-based software reliability maintenance system really reliable?" *arXiv preprint arXiv:2509.24352*, 2025.

[69] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 1285–1298.