Log-based Anomaly Detection based on EVT Theory with feedback

Jinyang Liu
Junjie Huang
The Chinese University of Hong Kong
Hong Kong, China

Yintong Huo
Zhihan Jiang
Jiazhen Gu
The Chinese University of Hong Kong
Hong Kong, China

Zhuangbin Chen
School of Software Engineering, Sun Yat-sen University
China

Cong Feng
Computing and Networking Innovation Lab,Huawei Cloud Computing Technology Co., Ltd
China

Minzhi Yan
HCC Lab, Huawei Cloud Computing Technology Co., Ltd
China

Michael R. Lyu
The Chinese University of Hong Kong
China

ABSTRACT

System logs play a critical role in maintaining the reliability of software systems. Fruitful studies have explored automatic log-based anomaly detection and achieved notable accuracy on benchmark datasets. However, when applied to large-scale cloud systems, these solutions face limitations due to high resource consumption and lack of adaptability to evolving logs. In this paper, we present an accurate, lightweight, and adaptive log-based anomaly detection framework, referred to as `ScaleAD`. Our method introduces a Trie-based Detection Agent (TDA) that employs a lightweight, dynamically-growing trie structure for real-time anomaly detection. To enhance TDA's accuracy in response to evolving log data, we enable it to receive feedback from experts. Interestingly, our findings suggest that contemporary large language models, such as ChatGPT, can provide feedback with a level of consistency comparable to human experts, which can potentially reduce manual verification efforts. We extensively evaluate ScaleAD on two public datasets and an industrial dataset. The results show that ScaleAD outperforms all baseline methods in terms of effectiveness, runs 2× to 10× faster and only consumes 5% to 41% of the memory resource.

1 INTRODUCTION

Cloud computing has surged into popularity in recent years. Large-scale cloud vendors, such as AWS, Azure, and GCP, provide 7×24 services to customers over the world. Ensuring the reliability of cloud systems is a critical task [3, 5, 10], because a small period of downtime could result in significant financial loss for both cloud vendors and their customers [2]. A preliminary step to safeguard reliability is timely and accurate detection of suspicious system behaviors, *i.e.*, anomaly detection. Similar to traditional software systems, logs in cloud systems provide valuable insights into the system's functioning and potential issues. Log-based anomaly detection, *i.e.*, timely identifying anomalous log messages for prompt resolution of issues, has been widely recognized as an essential task of cloud system management [6, 12–14, 41].

Existing approaches typically adopt machine learning or deep learning-based techniques to identify anomalous logs. Traditional machine learning-based approaches primarily consider statistical information (*e.g.*, log occurrence counts) and apply models such as isolation forest (IF) [20], support vector machine (SVM) [18], decision tree (DT) [4], logistic regression (LR) [1], to identify anomalies. Besides, recent studies have explored the use of deep learning methods to process logs. Such approaches typically extract semantic information from log messages through word embedding [38, 40] or Bidirectional Encoder Representations from Transformers (BERT) [16], and perform anomaly detection accordingly.

Although previous studies have demonstrated impressive performance on benchmark datasets, they are not practical for production cloud systems due to the following two reasons. First, previous solutions tend to pursue a high detection accuracy while overlooking the optimization of computation and space complexity. Cloud systems often use instances (*i.e.*, virtual servers) to host customers' applications. Conducting log anomaly detection for each instance enables close monitoring of its health status. However, as customers' applications already take up most resources of the instances, there are limited resources left to run an anomaly detector, which renders existing solutions impractical. For instance, deep learning-based methods [16, 40] require heterogeneous accelerators (*e.g.*, GPU and FPGA) [27, 35] to allow real-time inference, which may not be available to every instance. A straightforward approach is to transmit the log data to centralized compute nodes with abundant computational resources, which subsequently return the detection results. However, instances in a cloud can produce extensive amounts of log data (*e.g.*, Azure reported that 5 billion log messages are generated per day [33]), that are distributed across different clusters and datacenters. Transmitting such large-scale distributed logs to compute nodes could cause additional network and I/O overhead, which is prohibitively expensive and time-consuming.

The second reason is that previous approaches struggle to be adaptive to deal with diverse and evolving log data. In cloud systems, frequent launches of new versions of software result in changes to logging statements over time. For instance, Google has reported that there are thousands of newly-added logging statements due to software updates every month [36]. Existing methods typically train models to learn anomaly patterns from historical log data, which can hardly adjust to new logs, causing performance degradation. To address this problem, a recent study [40] exploits the semantic similarity between historical logs and new logs, enabling

the algorithm to transfer knowledge about anomalies from historical data to new data for anomaly identification. However, logs in real systems are complicated, and whether the assumption (*i.e.*, new logs share similar semantics with the historical ones) holds in real cloud systems has not been well investigated.

In this paper, we conduct a study on the logs in Huawei Cloud (a large-scale cloud system) to better understand the characteristics of logs in production cloud systems. We observe that an instance could generate several gigabytes of logs daily, making log anomaly detection critical for it. However, only very limited resources (*e.g.*, one CPU core and 200MB memory) are left for a plug-in anomaly detection process. Besides, logs are evolving and have low semantic similarities. Our study finds that during a one-month-long development cycle of 20 microservices, approximately 14.5% of brand-new logs are introduced on average. When comparing the semantic similarities between two versions of the same microservice, we find that 85% of log message pairs share little semantic similarity.

Based on the above results, we can summarize that a practical log-based anomaly detection method for cloud systems should be *accurate*, *lightweight*, and *adaptive*. It is non-trivial to achieve all the above three requirements simultaneously. Lightweight anomaly detection methods (*e.g.*, logistic regression [1]) cannot effectively handle newly occurring anomalous logs (*i.e.*, not adaptive). On the other hand, adaptive methods (*e.g.*, RobustLog [40]) need to extract semantic information from logs, which is compute-intensive (*i.e.*, not lightweight). It is challenging, if not infeasible, for pure data-driven methods to be both lightweight and adaptive. In practical scenarios, the involvement of engineers is crucial for system operation [5, 9, 34]. In Huawei Cloud, on-call engineers must manually confirm the reported anomalies before initiating the mitigation process. Based on this observation, we propose to incorporate the valuable feedback (*i.e.*, the ground-truth labels of anomalies) from experts, which is necessary for a practical (*i.e.*, both lightweight and adaptive) log-based anomaly detection method in cloud systems.

Based on such intuition, we propose `ScaleAD`, a scalable and adaptive log-based anomaly detection framework with experts in the loop for cloud systems. To meet the lightweight and adaptiveness requirement, ScaleAD employs a trie-based detection agent (TDA) to efficiently perform anomaly detection. TDA utilizes a lightweight trie structure (also known as a prefix tree) that stores received log messages in a compact form by allowing log messages to share the same internal nodes. In addition, the trie structure can be dynamically expanded during runtime to incorporate new logs via incrementally adding new nodes to the trie. Using the trie structure, we can efficiently calculate the occurrence distribution of different logs, based on which we apply extreme value theory (EVT) [31], an efficient statistics-based method, for anomaly detection.

Our TDA is lightweight and can be locally deployed on the instance of interest without consuming excessive resources. It raises an alarm to experts (such as on-call engineers) for verification when anomalous logs are detected. TDA then incorporates the verification results as feedback to continuously improve its accuracy. To reduce the workload of manual verification for on-call engineers, we allow various forms of experts. One option is to use a knowledge base storing rules obtained from historical operations, such as keyword matching. Besides, inspired by the recent success of large language model (LLM) [26] (*e.g.*, ChatGPT [29]), we propose that leveraging LLMs as an expert is a promising approach. By leveraging the LLM's ability to comprehend the semantic meaning of logs, we can determine whether they are anomalous or not. Our case study, which is based on real-world industrial logs in Huawei Cloud, demonstrates that ChatGPT is capable of providing feedback that is consistent with that of on-call engineers to some extent.

To evaluate the performance of ScaleAD, we conduct extensive experiments on two widely-used public datasets and an industrial dataset collected from Huawei Cloud. The experimental results demonstrate that compared with state-of-the-art solutions, ScaleAD achieves the best performance (0.908 to 0.990 F1 scores) in the setting where log data is fixed. ScaleAD retains a high and stable accuracy even if log messages evolve. Moreover, compared with existing methods, ScaleAD is 2× to 10× faster and consumes 5% to 41% fewer memories.

To sum up, this paper makes the following contributions:

- We conduct a study to understand the practical requirements for log-based anomaly detection methods in cloud systems (Section 2.2). The study revealed that a practical method should be accurate, lightweight, and adaptive, which aims to bridge the gap between research on log-based anomaly detection and its real-world application.
- We propose ScaleAD, a novel log-based anomaly detection framework. It comprises a lightweight trie-based detection agent and an expert module in the loop to achieve all three practical requirements, *i.e.*, accurate, lightweight, and adaptive (Section 3).
- We conduct extensive evaluations of ScaleAD on both public and industrial datasets. The results demonstrate that ScaleAD outperforms state-of-the-art methods in terms of effectiveness, adaptability, efficiency, and space consumption (Section 4).

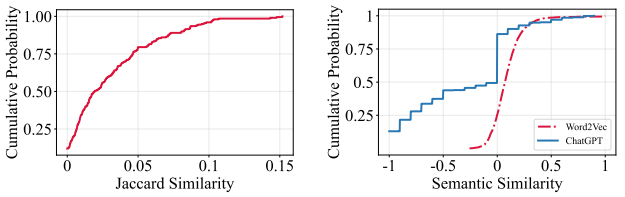
2 BACKGROUND AND MOTIVATION

In this section, we first discuss the background of log-based anomaly detection. Then, we present a study to comprehend the characteristics of logs within Huawei Cloud. Based on this understanding, we outline the industrial requirements that inspire our method design.

2.1 Log-based Anomaly Detection

The goal of log-based anomaly detection is to identify the log messages that may indicate a system problem in the runtime. In the literature, most studies adopt a two-step paradigm *i.e.*, *log parsing* and *anomaly detection* [6, 13, 17]. In the log parsing step, these methods obtain log templates via identifying the constant and variable parts. For instance, consider the raw log message "Finished task 0.0 in stage 6.0 (TID 247)." Its log template would be "Finished task <*> in stage <*> (TID <*>)," where the parameters (0.0, 6.0, and 247) are replaced with "<*>." The sequence of log templates is then processed through various downstream methods to conduct anomaly detection. For example, the library Loglizer [13] applies various machine learning-based methods (*e.g.*, logistic regression and decision tree) to detect anomalies based on the count distribution characteristics of templates.

Although log-based anomaly detection methods have long been recognized as an important problem for software maintenance, their practical application in cloud scenarios is still understudied.



(a) Templates Jaccard similarity of 20 Microservices

(b) Semantic similarity comparison of log templates between two versions of the same microservices

Figure 1: Statistics of log messages in Huawei Cloud

Unlike traditional software, cloud systems operate on a larger scale, generating massive volumes of logs from millions of physical and virtual instances. This presents new requirements for log analysis, which we aim to investigate in the following section by conducting a study on log data characteristics in cloud systems such as Huawei Cloud.

2.2 Characteristics of Log Data in Huawei Cloud

To study the characteristics of log data in Huawei Cloud, we collect one-month log data from 20 internal microservices in our production environment. These microservices are from the IaaS layer of Huawei Cloud, spanning functionalities such as API Gateway, user management, auto-scaling, and load balance. In the following study, we aim to understand the practical requirements for log-based anomaly detection methods in cloud systems.

2.2.1 Massive and Distributed Log Data. The huge volume of logs generated by modern cloud systems has been widely recognized in recent studies [21, 33]. In Huawei Cloud, a single service can produce tens of gigabytes of log data per day due to the frequent invocation by a large number of users. Moreover, this large volume of log data is distributed on different instances, resulting in great challenges for log analysis. A direct solution for log analysis is to transmit the log data to a central service for analysis, which, however, is not practical given the associated transportation cost in terms of time and bandwidth. Therefore, analyzing log data locally within the instance is desirable. However, in multi-tenant cloud systems, vendors pack resources such as CPU cores and memory as instances (*e.g.*, virtual machines) that are provided to external or internal users to run various applications. The running application on an instance usually can consume most of the resources, leaving very limited resources for plug-in processes such as log-based anomaly detection. For example, in Huawei Cloud, we find that most control plane instances have only ~200MB memory and a single CPU core that can be allocated to an anomaly detector without adversely affecting other applications on the instance.

Recent state-of-the-art solutions using deep learning-based techniques, (*e.g.*, LSTM [7, 24, 40] and transformers[16]), achieves promising anomaly detection accuracy. However, these methods require significant memory resources to store a large number of parameters, such as word embeddings, and their inference efficiency is inadequate without GPU acceleration, as demonstrated in Section 4.5. In conclusion, a lightweight method that can quickly process log analysis without consuming too much memory is essential to address the challenges posed by massive and distributed log data.

2.2.2 Evolving and Diverse Log Data. Besides the huge log volume, the current software development process typically adopts the continuous integration/delivery (CI/DI) paradigm [32], making changes to software and deployments increasingly frequent. As a result, the log data in cloud systems change over time, *e.g.*, modifications to logging statements and new-added log messages, causing great challenges to downstream log analysis tasks.

To better understand the situation of log evolution in Huawei Cloud, we study the number of log template changes from 01/02/2023 to 01/03/2023 of 20 microservices. These templates are obtained by running a log parser Drain [11] as previous studies [33, 40]. Our analysis reveals that a one-month evolution in Huawei Cloud can lead to a 14.5% increase in the number of new log templates. Such changes are typically due to actions like fixing bugs and developing new features. Our findings indicate that log template evolution is a natural and continuous process in Huawei Cloud.

Besides the change of template numbers, we further study how its content changes. First, we enumerate each pair of microservices among the selected 20 microservices and measure how their log templates overlap. Specifically, we denote the template sets of two microservice as set_i and set_j, each of which contains unique templates of corresponding microservices. Then we compute the Jaccard similarity, *i.e.*, $J(set_i, set_j) = \frac{|set_i \cap set_j|}{|set_i \cup set_j|}$, where $|\cdot|$ denotes the number of unique template in a set. We plot the CDF (cumulative distribution function) figure of the similarity distribution in Figure 1(a). We find that around 97% of pairs of microservices have less than 0.1 Jaccard similarities in their template sets. This is intuitive since different microservices tend to log different information.

Second, we measure the extent to which newly-added logs share semantics with historical logs when a microservice undergoes an upgrade. To accomplish this, we selected the most frequently updated microservice and compared the semantic similarity between its log messages before and after the update. We computed the semantic similarity for each pair of historical and new log messages by using Word2Vec [25] to obtain a semantic vector for each log and computed a cosine similarity to represent the semantic similarity between each pair. The cosine similarity ranges from -1 (most dissimilar) to 1 (most similar). Furthermore, to obtain more reliable results, we used ChatGPT, a language model known for its ability to understand text semantics, to score the semantic similarity of each log message pair on a scale of -1 to 1. We plotted the CDF of similarities of these log pairs in Figure 1(b). Our findings reveal that Word2Vec and ChatGPT produce similar results, indicating that most log pairs (~ 95%) have similarities less than 0.3. Notably, ChatGPT regards 85% of log pairs as having little semantic similarity (≤ 0). These results suggest that software updates can introduce new log messages that differ significantly from historical logs.

Zhang et al. [40] attempts to tackle the log evolving issue by ensuring robust representation of log data. The assumption is that different log messages could share similar semantics (such as synonyms). However, our study has demonstrated that this is hardly applicable, given the significant diversity of logs, even within the same microservice. In conclusion, there is a practical need for a method that can adaptively and accurately detect anomalies from the evolving and diverse log data in real-world scenarios.

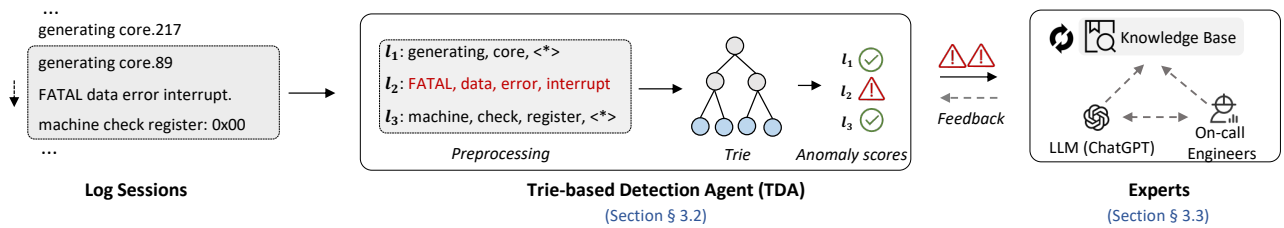


Figure 2: The overall framework of ScaleAD

2.3 Demanding Industrial Requirements

We summarize the following industrial requirements based on our study in Huawei Cloud as introduced in Section 2.2.

- **Accurate**. High accuracy is crucial for an anomaly detector, as it indicates that the detector can identify potential issues in logs without generating too many false positives, which may distract on-call engineers. This is the primary priority of a practical log-based anomaly detector.
- **Lightweight**. To process large amounts of log data on instances with limited computational capability and memory without affecting running applications, a lightweight anomaly detector should (1) use minimal memory resources and (2) efficiently handle massive streaming log messages, preferably without the acceleration of special devices (e.g., GPU).
- **Adaptive**. A practical method for log-based anomaly detection should be able to learn adaptively from diverse microservices log data, incorporate human feedback to address false positives and maintain accuracy, and operate continuously in an online production environment without interruption, such as retraining and redeployment.

3 METHOD

In this section, we present the proposed framework, ScaleAD, designed to address the stringent industrial requirements. We begin with an overview of ScaleAD in Section 3.1, followed by a detailed explanation of its components in the subsequent sections.

3.1 Overview

ScaleAD comprises two core elements: a trie-based detection agent (TDA) and an expert module, as illustrated in Figure 2. We first collect streaming log messages generated by an instance and arrange them into sessions (or windows). Next, TDA receives the windows of log messages as input and produces an anomaly score for each log line within the window. During this process, TDA identifies any suspicious logs and forwards them to experts for further verification. The experts then return their feedback to TDA, which integrates this feedback to continuously refine its accuracy.

Within this framework, TDA utilizes a lightweight trie structure to parse log messages into templates and detect anomalies based on the count distribution of those templates. The trie structure stores a compact record of encountered log templates and their corresponding counts to ensure efficient processing of high-volume log data. Moreover, the trie structure is designed to be adaptive and can be dynamically expanded during runtime to accommodate new templates. Due to its efficiency in design, TDA can be deployed locally on an instance, while we allow a remote expert to provide feedback to TDA. The expert could take many forms, such as an on-call engineer, a knowledge base containing various rules, or a computational-intensive large language model (LLM). By leveraging the expertise of a remote specialist, TDA can further improve its accuracy and reduce the likelihood of false positives.

3.2 Trie-based Detection Agent

Trie-based Detection Agent (TDA) processes raw log messages as input and generates an anomaly score for each log message as output. The TDA comprises five steps: *preprocessing*, *internal node traverse*, *leaf node update*, *trie update*, and *anomaly detection*. At the heart of the TDA is a trie, consisting of *internal nodes* and *leaf nodes*. Log messages traverse the trie based on heuristics defined at internal nodes, which take into account various log message characteristics. The objective is to assign similar log messages to the same leaf node in a coarse-grained manner. Upon reaching a leaf node, a log message is assigned to a *log cluster*. Each log cluster comprises log messages sharing the same template. Finally, anomaly detection is carried out efficiently using the statistical information gathered from the log clusters. In the following, we provide a detailed explanation of each step within the TDA.

3.2.1 Preprocess. For an input log message l_i, we follow a common practice [33, 41] to use pre-defined regular expressions to extract parameter fields such as IP address and URL. Note that one can also apply other regular expressions according to logs produced by a specific system. After that, we tokenize the log message with non-alphanumeric splitters (*i.e.*, any characters that are not letters or numbers), generating log tokens $\hat{l}_i = [w_1, w_2, ...]$. In the following, we consider each log message comes in a streaming manner.

3.2.2 Internal node traverse. We defined the following three general heuristics within internal nodes of the Trie, which can be generally applied to log messages generated by various systems. Figure 3 presents a running example of Trie.

(1) Traverse by domain knowledge. Initially, log messages are separated based on distinguishing features, such as levels (e.g., INFO and DEBUG) and components, since log messages with different levels or generated by different components are more likely to belong to separate templates.

(2) Traverse by most frequent tokens. As frequently occurring tokens in log messages are more likely to be constant parts of a template [21], we extract the K most frequent tokens from each log message's tokens l_i as a *token key*. Log messages with different token keys are then separated. Generally, K is set to 3, but may be increased for systems producing lengthy log messages. ScaleAD maintains a vocabulary that counts token occurrences while processing log messages. English stopwords are discarded when extracting token keys to avoid grouping dissimilar log messages.

(3) Traverse by prefix tokens. Inspired by the observation of Drain [11] that tokens at the beginning of a log message are more likely to be constants, we divide log messages based on their first d prefix tokens one by one, *e.g.*, "open" and "file" in the example of Figure 3, Generally, setting d to 3 yields satisfactory results. However, prefix tokens may sometimes be parameters, resulting in many child nodes in the trie. To address this issue, we use a hyper-parameter c_{max} to limit the maximum number of child nodes. For example, when setting $c_{max} = 3$, the 4th child of a leading token will be replaced by a parameter "<*>" that can match any tokens.

3.2.3 Leaf node update. Upon traversing all internal nodes in the Trie, a log message l_i will eventually reach a leaf node. Leaf node update is then performed to extract a log template for each log. In particular, each leaf node contains multiple *log clusters*, represented as tuples of the form $C_i = (\mathbf{L_i}, t_i)$, where $\mathbf{L_i}$ records the IDs of all log messages in the cluster, and t_i denotes the template shared by these log messages.

As shown in the right side of Figure 3, we obtain the template of the arriving log by matching it with a log cluster in the leaf node. We first attempt to match it with existing log clusters (*i.e.*, exact or partial match). If this matching process fails (*i.e.*, no match), we create a new log cluster containing only the arriving log message. The associated template of the log cluster is then updated based on the arriving log message. We elaborate on exact match, partial match and not match as follows.

(1) Exact match. Firstly, we regard the template of each log cluster as a regular expression (e.g., replace <*>" as .*?") to match the given log message. This approach allows the template to match longer log messages, even if they contain varying-length parameters, as the regular expression allows matching tokens of any length. For example, the regular expression "dumping .*?" can successfully match "dumping item1, item2" and "dumping item0".

(2) Partial match. Next, if the exact match fails, we conduct a partial match. Specifically, we tokenize the template and calculate the Jaccard similarity between the template and the candidate log message. We denote the tokenized template as $\hat{t}_i = [w_1, w_2, ...]$. The Jaccard similarity $J(\hat{l}_i, \hat{t}_i)$ can be calculated as $J(\hat{l}_i, \hat{t}_i) = (|\hat{l}_i \cap \hat{t}_i|)/(|\hat{l}_i \cup \hat{t}_i|)$, where $|\cdot|$ denotes the number of unique tokens in a list. We calculate the distance between $\hat{l}_i$ and each template of existing log clusters. If the *maximum* distance $\hat{d}_J(\hat{l}_i, \hat{t}_i)$ is larger than a threshold θ_{match} (typically set to 0.5), l_i can match t_i, otherwise, it cannot match any existing log clusters (*i.e.*, no match).

(3) No match. If no log clusters in the leaf node can match the given log message, we create a new log cluster (e.g., $C_j = (L_j, t_j)$) in the leaf node. The template for this new log cluster is the log message itself, *i.e.*, $t_j = l_i$, and this log cluster contains only one log message, *i.e.*, $L_j = [i]$.

Template update. If the given log message l_i matches a log cluster C_i, we add the identifier i to the record L_i of C_i, and update the template t_i for the matched log cluster using l_i. Specifically, we first identify the common set of tokens shared by both t_i and l_i. Next, we choose the list that has more tokens between $\hat{t}_i$ and $\hat{l}_i$. Finally, we replace any token in the longger list that is not in the common token set with the placeholder "<*>". This process generates the new template for the log cluster C_i.

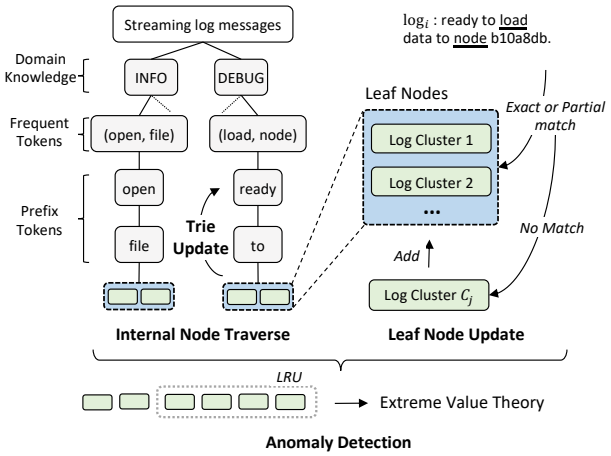


Figure 3: The workflow of trie-based detection agent (TDA)

3.2.4 Trie update. Diverse log messages received in a continuous stream can result in inaccurate template extraction since false templates can be produced before a sufficient number of log messages accumulate. This, in turn, affects the accuracy of both exact and partial matching steps, producing more false log clusters. For instance, two log clusters with similar templates may coexist: t_1 as "deleted items: <*>" and t_2 as "deleted items: obj1, obj2, obj3, obj4". The reason is that t_2 has many parameters (obj1-obj4), leading to higher Jaccard distance and cannot be merged with t_1 when doing leaf node update (*i.e.*, no match).

To tackle this problem, we propose a bottom-up Trie update approach that merges log clusters sharing the same template into a single cluster and reconstructs the Trie based on the new template of the merged log cluster. The approach starts by identifying a set of log cluster candidates denoted as $S_c = [C_1, C_2, ...C_M]$ that share the same "domain knowledge" and "frequent token" internal nodes. Candidates are then sorted based on the number of "<*>" in their templates, and those with more parameters are prioritized. The approach iterates through each candidate template and checks if the preceding template *contains* the succeeding one. If so, the succeeding log cluster is deleted, and its log sequences are merged with the preceding one. To determine if template A *contains* template B, we generate a regular expression from A (by replacing "<*>" with ".*?") and verify its match with template B. Finally, the Trie is constructed using the remaining log templates, yielding a more concise and accurate trie with a reduced template count.

Updating trie excessively for each log message will obfuscate TDA's efficiency, as it involves matching regular expressions between existing templates and reconstructing the Trie. To optimize efficiency while ensuring Trie accuracy, we trigger Trie updates periodically, such as every million lines of log messages.

3.2.5 Anomaly detection. Log messages are parsed to templates after traversing the Trie structure. In previous approaches [13, 24], feature extraction and anomaly detection are then performed separately, leading to additional computation and potential maintenance costs. To address this issue, we propose a lightweight anomaly detection module that integrates seamlessly with the Trie.

Our approach aims to identify anomalies by detecting templates that appear significantly less frequently than others. To achieve

this, we leverage the Generalized Extreme Value (GEV) distribution within the Extreme Value Theory (EVT) framework [31], which can efficiently identify extreme values within a large set of values. Specifically, we enumerate each log cluster, count the occurrences of each template, and generate the count list $L_{count} = [x_1, x_2, ..., x_R]$, where x_i represents the count of the i_{th} template, and R is the number of templates considered. Next, we apply the GEV to detect anomalies in L_{count} by fitting the GEV distribution as follows:

$$f(x|\mu, \sigma, \xi) = \frac{1}{\sigma}\left[1 + \xi\left(\frac{x-\mu}{\sigma}\right)\right]^{-\frac{1}{\xi}}, \quad (1)$$

where $f(\cdot)$ represents the cumulative distribution function (CDF) of GEV, x is variable, and μ, σ, and ξ are the location, scale, and shape parameters of the distribution, respectively, We choose EVT because (1) it enables the efficient estimation of GEV parameters by considering only extreme values (such as minimal values). (2) the GEV distribution requires only three parameters (μ, σ, and ξ), making it memory-friendly. We proceed to compute an anomaly score tp for each log template, given by the following equation:

$$tp = \frac{f(x)^\tau}{\sum_{j=1}^{R} f(x_j)^\tau}. \quad (2)$$

In order to obtain a softer distribution of anomaly scores, inspired by Hinton et al [15], we introduce a temperature parameter τ into the formula. A lower τ value produces softer scores that are more evenly distributed across templates, while a higher τ value generates more extreme scores that concentrate heavily on a smaller subset of templates. We find that setting $\tau = 10$ leads to a satisfactory distribution of anomaly scores in practice.

When dealing with a large number of log templates, the entire set of log clusters can be computationally expensive when fitting the GEV distribution. To address this issue, we propose using the Least-Recently-Used (LRU) strategy to limit the number of log templates considered for fitting the distribution. The LRU strategy maintains a cache of the most recently seen log templates and discards the least recently used templates when the size of the cache reaches a user-defined size R (as used in Equation 2). The LRU strategy is based on the observation that only a small proportion of log messages are typically generated within a short time interval, such as ten minutes. Therefore, we can focus on the most recently seen log templates, which are likely to be relevant for anomaly detection during that time interval.

3.3 Leveraging Expert Feedback

Log evolution can inevitably lead to false positives when applying the TDA. For instance, when a new log template appears in a log window, its occurrence count is one, generating a high anomaly score. It is challenging to determine whether this truly indicates an anomalous log message. While semantic-based methods such as RobustLog [40] may address such cases by comparing the semantics with seen logs in training data, the semantics among different log messages usually exhibit significant differences (as shown in Section 2.2.1), rendering these solutions ineffective.

To address this issue, we propose to incorporate the knowledge of experts to facilitate the anomaly detection process, *i.e.*, ground-truth labels for log messages. It is a common practice in cloud systems that on-call engineers must manually verify every reported suspicious anomaly before mitigating a problem [5, 34]. This inspires us to seamlessly integrate such valuable feedback to ScaleAD. Specifically, when TDA identifies a suspicious anomaly, it issues a query to experts for confirmation. We use a querying threshold θ_{query} to control the number of queries issued. Only log templates with an anomaly score tp greater than θ_{query} are sent to an expert. Upon receiving a log template, an expert provides feedback in the form of a tuple ($decision$, ep_i), where $decision = 1$ indicates that the log template represents an anomaly, while $decision = 0$ indicates a normal log message. The expert also assigns a confidence score ep, which ranges from 0 to 1. Then we compute an integrated anomaly score p, which is a weighted average of TDA's output and the expert's feedback, where we use the expert's confidence as the weight. Formally:

$$p = \begin{cases} ep + (1-ep) \times tp, & \text{if } decision = 1 \\ 1 - (ep + (1-ep) \times (1-tp)), & \text{if } decision = 0 \end{cases} \quad (3)$$

To enable TDA to retain the expert's feedback, we store the decision, i.e. ($decision$, ep_i), within the corresponding log clusters in TDA after a log template is confirmed. This enables TDA to make the same decision guided by the expert in the future.

The experts can take multiple forms. *On-call engineers* with substantial domain knowledge can usually provide reliable feedback. However, too many queries from TDA could be distracting and introduce intensive manual efforts. Besides minimizing the query numbers by controlling θ_{query}, we find *Large language models (LLM)* such as ChatGPT could work as a promising expert (will show in Section 4.6). LLMs possess significant capabilities in understanding the semantics of natural languages, which enables them to identify potential errors described in logs. As depicted in Figure 2, the feedback obtained from experts is continuously cached to a centralized knowledge base that is shared among the different TDAs distributed across different instances. This ensures that redundant queries are avoided, making ScaleAD execute more efficiently. In addition, EVT only detects a small portion of anomalous log templates. Consequently, the number of queries to experts is considerably lower than the total amount of log data (will show in Section 4.4). As a result, sending queries does not occupy an excessive amount of network bandwidth.

4 EVALUATION

We evaluate ScaleAD by answering the research questions (RQs).

- RQ1: What is the effectiveness of ScaleAD under the *offline* setting?
- RQ2: What is the effectiveness of ScaleAD under the *online* setting?
- RQ3: How experts' feedback affects the performance of ScaleAD?
- RQ4: What is the time and space efficiency of ScaleAD?

4.1 Evaluation Setting

4.1.1 Dataset. We evaluate ScaleAD on two widely-used public datasets and an industrial dataset gathered from the production environment of Huawei Cloud. Table 1 provides the statistics of these datasets. Specifically, the BGL (Blue Gene/L) dataset is a

supercomputing system log dataset collected by Lawrence Livermore National Labs (LLNL) [14, 28]. The Thunderbird dataset originates from a Thunderbird supercomputer at Sandia National Labs (SNL) [14, 28]. Although existing studies often use 10 million continuous lines from the Thunderbird dataset for evaluation [16, 17], they do not specify which 10 million logs they employed; therefore, we use the first 10 million logs in this study. The industrial dataset is collected from the production environment of Huawei Cloud between 01/02/2023 to 01/03/2023, generated by 20 internal microservices providing the abilities of API Gateway, user management, auto-scaling, load balance, etc. We manually identify and label the anomalies in the log messages based on the diagnostic reports provided by our site reliability engineering (SRE) team. These reports document the starting times of actual system issues, which allows us to manually identify the anomalous log messages. After a thorough labeling process, we obtained a log data dataset with approximately 1.5 million lines of log messages, covering 60 types of system problems such as service unavailability, incorrect request parameters, and null pointers Although the obtained number of log lines is smaller than that of the BGL and Thunderbird datasets due to the extensive manual labeling effort, our dataset contains a significantly higher number of templates, showcasing its diversity.

4.1.2 Log Message Grouping. It is a common practice to group log messages into windows and decide whether each window is anomalous or not, *e.g.*, grouping log messages within fixed or sliding time windows[17]. For the industry dataset, we use short sliding time windows, specifically a ten-minute time window with a step size of two minutes. We choose this approach to replicate real-world scenarios that require real-time anomaly detection. The use of a short sliding time window has two main advantages. First, it ensures that anomalies are promptly detected and conserves memory. Second, there should be some overlap between two adjacent windows to ensure that anomalies spanning multiple windows are not missed. Furthermore, to avoid data leakage, we keep the time windows in chronological order without shuffling. The recent empirical study [17] has demonstrated that the length of the window size and whether or not to shuffle the windows can significantly impact the performance of log-based anomaly detection. Therefore, to ensure consistency with existing studies, we use the same window setting for the public dataset as in [17], namely, using a fixed window size of one hour without overlapping.

4.1.3 Implementation Details. We implemented ScaleAD as a microservice with approximately 1000 lines of Python code for easy use in Huawei Cloud. We conducted the following experiments on a Linux server running Ubuntu 18.04.6 LTS with 32GB of RAM and an Intel(R) Xeon(R) Platinum 8268 CPU @ 2.90GHz. We set the hyperparameters empirically and fixed them as they were found to generalize well across different datasets without tuning. For internal node traversal, we set the length of the token key, i.e., $K = 3$, and the maximum number of child nodes $c_{max} = 3$. For leaf node update, we set the matching similarity threshold $\theta_{match} = 0.5$. For anomaly detection, we fixed the temperature τ in Equation 2 to 10.

4.1.4 Evaluation Metrics. For each window, we calculate four measures: TP (true positive), which is the number of correctly predicted anomaly windows; FP (false positive), which is the number of predicted anomaly windows that are actually normal; TN (true negative), which is the number of correctly predicted normal windows; and FN (false negative), which is the number of predicted normal windows that are actually anomalous. Based on these numbers, we calculate precision (PC)= $\frac{TP}{TP+FP}$ and recall (RC) = $\frac{TP}{TP+FN}$. We also use F1 scores= $\frac{2\times PC\times RC}{PC+RC}$ to evaluate the overall performance.

Table 1: Statistics of evaluation datasets

Dataset	BGL	Thunderbird	Industry
# Log messages	4,747,963	10,000,000	1,488,073
# Templates	456	1,504	3,241
# Train windows (anomaly ratio)	2,884 (21%)	416 (55%)	3,048 (13%)
# Test windows (anomaly ratio)	722 (24%)	105 (30%)	933 (18%)

4.1.5 Comparative Methods. We have selected the following state-of-the-art studies as our comparative methods:

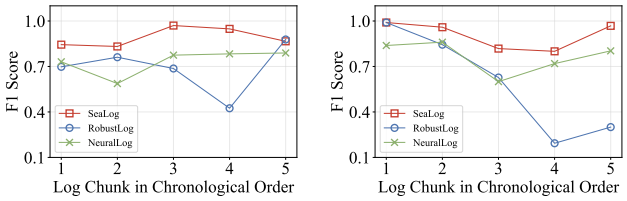
- **Loglizer (IF/LR/DT)** [13] is a log-based anomaly detection library encompassing a variety of machine learning (ML) methods. The library processes parsed log messages as input and computes a count feature vector representing the distribution of template counts. This count vector is subsequently used as input for ML-based algorithms to perform anomaly detection. Prominent ML algorithms within Loglizer include isolation forest (IF), linear regression (LR), and decision tree (DT).
- **DeepLog** [7] employs a Long Short-Term Memory (LSTM) model as its core component. DeepLog takes windows of log messages as input and predicts the next log event. Anomalies are detected if the prediction differs from the actual event.
- **LogAnomaly** [24] aims to detect anomalies by combining log count vectors and log semantic vectors, which also utilizes a forecasting-based mechanism to reflect anomalies.
- **LogRobust** [40] aims to capture the semantics of log messages through word embeddings to address the issue of ever-changing log messages. Each log message is encoded as a representation based on word vectors, which is modeled by an attention-based Bidirectional LSTM and trained in a supervised manner.
- **NeuralLog** [16] aims to bypass the log parsing step; instead, NeuralLog extracts semantic meaning from raw log messages and represents them as vectors to detect anomalies using a Transformer-based classification model.

4.2 Effectiveness under Offline Setting (RQ1)

In this RQ, we evaluate the effectiveness of ScaleAD in an offline setting, where we train and test the models with the previously collected data. To simulate this process, we split the whole dataset in chronological order, with the first 80% data for training and the remaining 20% data for testing, and keep them fixed following previous studies [16, 17]. The statistics of training and testing data are shown in Table 1. We incorporate the labels as feedback when training ScaleAD, and we do not use any feedback during testing, which is consistent with supervised models.

Table 2: Experimental results of offline anomaly detection (We use ∗ to denote unsupervised methods, others are supervised ones)

Method	BGL			Thunderbird			Industry		
	Precision	Recall	**F1 score**	Precision	Recall	**F1 score**	Precision	Recall	**F1 score**
IF*	0.125	0.615	0.208	0.291	0.968	0.448	0.176	0.994	0.299
DT	1.000	0.570	0.726	1.000	0.893	<u>0.912</u>	0.942	0.788	0.858
LR	0.738	0.437	0.549	0.842	0.516	0.640	0.818	0.655	0.727
DeepLog*	0.241	0.895	0.380	0.295	1.000	0.456	0.358	0.909	0.513
LogAnomaly*	0.268	0.862	0.409	0.307	1.000	0.470	0.360	0.927	0.519
RobustLog	0.942	0.961	<u>0.951</u>	1.000	0.710	0.830	0.984	0.764	0.860
NeuralLog	0.881	0.886	0.883	0.713	0.719	0.715	0.889	0.895	<u>0.887</u>
ScaleAD	0.993	0.993	**0.990**	1.000	0.903	**0.949**	1.000	0.931	**0.908**



(a) BGL (b) Industry

Figure 4: Experimental results of online anomaly detection

The evaluation results are presented in Table 2, which categorizes the baseline methods into ML-based (upper half) and DL-based (lower half) approaches. The highest F1 score is emphasized in **bold**, and the second-best score is <u>underlined</u>. We can make the following observations: (1) <u>Unsupervised methods</u> (IF, DeepLog, and LogAnomaly) <u>show significantly lower effectiveness than supervised approaches</u> in terms of F1 score. The reason is that they are originally designed to consider all training windows as normal. Therefore, during inference, they classify every window containing unseen log templates as anomalies, resulting in high recall but low precision. (2) Supervised methods achieve a better balance between recall and precision by learning from labeled data. Furthermore, with sufficient labels, ML-based methods (i.e., DT) can still achieve comparable or better performance (on Thunderbird) compared to DL-based methods. This observation suggests that the count distributions of different templates could facilitate anomaly detection. The two observations (1) and (2) are consistent with the recent empirical studies [6, 17] that benchmark existing DL-based methods. (3) ScaleAD attains the highest F1 score across all three datasets, ranging from 0.908 to 0.990. This result demonstrates the effectiveness of detecting anomalies from count distribution (i.e., results of TDA), as well as utilizing the labeled data in the training data (i.e., experts' feedback).

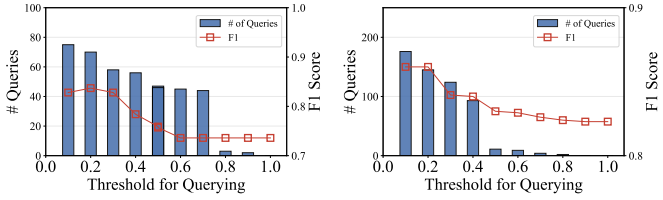
Answer to RQ1. ScaleAD achieves the best F1 score (0.990, 0.949 and 0.908) among all state-of-the-art baselines across two public datasets and one industrial dataset. The results demonstrate that ScaleAD can effectively fuse the results of TDA and experts' feedback in the offline setting, thus meeting the requirement of being accurate for practical log-based anomaly detection.

4.3 Effectiveness under Online Setting (RQ2)

Our preliminary study has shown that real-world log data is continuously evolving, which highlights the need for a practical anomaly detection system that can adapt to these changes and maintain high accuracy. To address this requirement, we investigate the effectiveness of ScaleAD within an online setting in RQ2. Based on the results of RQ1, we select the most effective model (i.e., RobustLog and NeuralLog) as strong baselines for this question. To simulate a scenario where new log messages are received, we divide the BGL and Industry datasets into six even chunks (numbered from 0 to 5) in chronological order, each containing log messages with different templates. In practice, on-call engineers must verify suspicious logs reported by an anomaly detector, which naturally provides labeled data allowing us to continue training a model on-the-fly. We train the models using the preceding chunk and evaluate the trained models on the subsequent chunk (e.g., training with chunk 0 and testing with chunk 1), ultimately yielding five evaluation results. Although RobustLog and NeuralLog were not initially designed for processing streaming feedback, we equip them with the capability to perform continuous training by fitting the incoming chunk, enabling a fair comparison with ScaleAD.

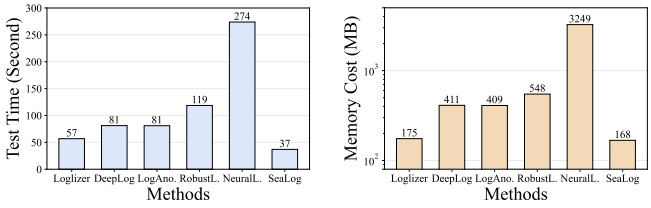
Figure 4 presents the experimental results. Our observations are as follows: (1) ScaleAD consistently achieves the highest F1 scores across all chunks for both datasets, demonstrating its successful adaptation to new chunks and maintaining stable performance. (2) RobustLog's performance significantly declines from chunk 1 to chunk 4 in both datasets, indicating that it experiences catastrophic forgetting [8] when accommodating new chunks. Nevertheless, its performance improves in the 5th chunk on the BGL dataset, as chunks 4 and 5 share considerable data overlap, enabling RobustLog to detect the most recent anomalies. Conversely, the Industry dataset is more complex and has less overlap between chunks, resulting in substantially lower performance for RobustLog compared to other models. (3) NeuralLog exhibits greater stability in performance than RobustLog due to its transformer architecture, which possesses a larger number of parameters capable of memorizing observed data and mitigating the catastrophic forgetting issue.

Answer to RQ2. ScaleAD consistently surpasses the strong baseline methods (RobustLog and NeuralLog) on both BGL and industry datasets within the online setting. The results demonstrate that ScaleAD is able to adapt to evolving logs, thereby meeting the requirements of practical log-based anomaly detection that demands adaptability.



(a) BGL (b) Industry

Figure 5: Anomaly Detection performance w.r.t. the number of queries to experts



(a) Time Efficiency Comparison (b) Space Efficiency Comparison

Figure 6: Experimental results of time and space efficiency comparison

4.4 Impact of Experts' Feedback (RQ3)

In this RQ, we aim to study how the number of queries affects ScaleAD's effectiveness. To achieve this goal, we train ScaleAD with only 50% of the training data for BGL and industry datasets, which mimics the scenario that the model might not be well trained initially. Then, we use the same test set as in RQ1 to evaluate ScaleAD's performance. While testing, we vary the amount of feedback given to ScaleAD by tuning the threshold to issue a query, *i.e.*, θ_{query} in the range of [0, 1] with step size of 0.1.

The results are presented in Figure 5. We can obtain the following observations: (1) The expert is asked less when increasing the threshold θ_{query}. With less feedback, the performance of ScaleAD is degraded. However, even if no feedback is taken (i.e., setting $\theta_{query} = 0$), ScaleAD can still achieve around 0.73 F1 score for BGL and 0.82 for industry data, respectively. In this setting, anomaly detection is only done with the trained TDA. (2) When the query threshold θ_{query} is set to values greater than or equal to 0.8 and 0.5 for the BGL and industry datasets respectively, the number of queries reduces significantly. This indicates that there are only a few samples in the datasets with anomaly scores that exceed these threshold values. The reason for this is that the EVT-based anomaly detection module in ScaleAD tends to conservatively assign high anomaly scores to a small number of anomalous samples. (3) After being trained, TDA only queries fewer than 80 and 190 times on BGL and Industry datasets, respectively. For the industry dataset, it indicates most 6 queries per day can me received from TDA. This indicates that TDA can effectively filter out most of the templates and only interact conservatively with experts.

Answer to RQ3. The performance of ScaleAD improves as the amount of feedback increases. Owing to the EVT-based anomaly detection model in TDA, most normal log messages are filtered out, resulting in only 190 queries in total (6 queries per day) for the industry dataset. Furthermore, even without any feedback, the trained ScaleAD maintains F1 scores of 0.71 and 0.82 on the BGL and Industry datasets, respectively.

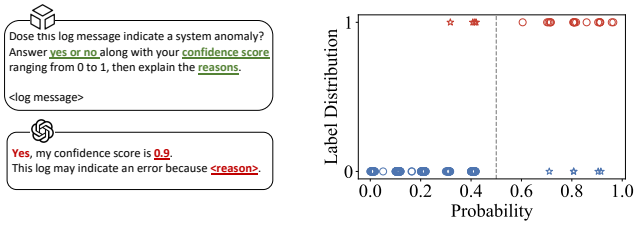
4.5 Time and Space Efficiency (RQ4)

T A practical log-based anomaly detector must be capable of processing a large number of log messages efficiently, while also minimizing memory usage. In this research question, we compare ScaleAD with baseline methods in terms of time and space efficiency. In this RQ, we only evaluate the TDA part because TDA is responsible for online log analysis, while the expert can be deployed remotely.

(1) Time efficiency We measure the time required to perform anomaly detection on the complete BGL test set with the aim of simulating the scenario of deploying these methods in a production environment. For DL-based methods, we set them to evaluation mode without backpropagation. Additionally, since online-generated logs are not pre-parsed, we include the time required to parse raw log messages using Drain [11] for baseline methods. The time required for ScaleAD encompasses all steps from preprocessing to anomaly detection. Note that I/O time is excluded for all methods, and these methods are executed without GPU acceleration, which may not be available when resources are constrained.

Figure 6(a) presents the comparison results. Since LR, IF, and DT present similar efficiency, their values are averaged and denoted as Loglizer in the figure. We can observe that: (1) ScaleAD is the most efficient method, being 2 to 10 times faster than the baseline methods; (2) ML-based methods (Loglizer) demonstrate higher efficiency than DL-based methods but cost more time than ScaleAD. The reason is that Loglizer requires a separate parsing step while ScaleAD conducts log parsing and anomaly detection simultaneously; (3) DeepLog and LogAnomaly exhibit comparable performance, as they both utilize LSTM-based architectures. In contrast, RobustLog and NeuralLog require significantly more time than others, as they adopt networks with a greater number of parameters.

(2) Space efficiency We employ Memory Profiler [30] to track the memory consumption of each method while processing a single log session. Since these methods operate on a single session during online log analysis, we chose one session for the comparison. The comparison emphasizes the maximum memory occupied by each method, as an out-of-memory error may occur if the available memory in an instance is less than what a method requires. Figure 6(b) displays the comparison results. The memory usage trend is similar to the time efficiency trend. ScaleAD demands the least memory during log processing, necessitating only 5% to 41% of the memory consumed by DL-based methods. This is primarily because ScaleAD only needs to store the trie structure in memory, which is typically sparse. Loglizer demonstrates comparable space efficiency to ScaleAD, since it only stores the count vector of a log session, and the subsequent ML-based method (e.g., LR) is memory-efficient. DL-based methods necessitate storing a large number of parameters (e.g., network parameters), resulting in higher memory usage. NeuralLog, which adopts a heavy transformer architecture, consumes the most memory among the methods.



(a) Prompt design for anomaly detection (b) Probability distribution of ChatGPT

Figure 7: A case study of using ChatGPT as an expert

Answer to RQ4. ScaleAD outperforms the selected state-of-the-art baseline methods, being 2 to 10 times faster, and requires only 5% to 41% of the memory consumed by recent deep learning-based methods. The results demonstrate that ScaleAD is efficient concerning both time and space efficiency, thus meeting the requirement of being lightweight for practical anomaly detection.

4.6 Case Study

In this section, we study the feasibility of employing ChatGPT as an expert to support on-call engineers in the loop of ScaleAD. Specifically, we aim to measure the extent to which ChatGPT can provide feedback that is as consistent as that of on-call engineers. To achieve this, compare the decisions of ChatGPT and that of experts (*i.e.*, groundtruth) for querying the same set of logs. We randomly select 200 lines of log messages from the production environment of Huawei Cloud, 24 of which indicate a system problem, with the rest being normal logs. We manually remove sensitive information in the logs for privacy issues. The prompt to query ChatGPT is shown in Figure 7(a), which requires ChatGPT to output a confidence score, along with reasons for its decision, which could assist on-call engineers in understanding its output. Note that we have made the queries and responses publicly available in our Github repository.

By comparing the "yes or no" answers of ChatGPT with human labels, we can obtain an f1 score of 0.816, recall of 0.833 and precision of 0.8. Figure 7(b) shows the detailed probability distribution of ChatGPT's output. We observed that most cases were appropriately classified with a 0.5 threshold. However, some false predictions were still present (represented as stars in the figure). After manually inspecting these false predictions, we noticed that (1) ChatGPT may miss some anomalies if log messages lack evident semantics that indicate an error. (2) the normal logs that are wrongly classified as abnormal were subtle anomalies that engineers typically disregard, such as "failed to login, wrong password." Such logs often contain keywords such as "failed," indicating abnormal semantics.

These observations suggest that ChatGPT is powerful in identifying anomalies purely based on the semantics of log messages. It is promising to integrate ChatGPT within the framework of ScaleAD, as evidenced by its high consistency with human decisions (*i.e.*, high F1 score of 0.816). However, integration with more domain knowledge of on-call engineers could further enhance its performance, such as prompt designs that enable ChatGPT to ignore login errors in this case. Since our paper's primary focus is not designing a high-performing expert, we leave this for future work.

5 THREATS TO VALIDITY

We identify the following threats to external and internal validity.

External validity. The type of log-based anomaly is the first external threat. ScaleAD assumes that log templates with low occurrence counts are more likely to be anomalous. While this assumption is generally valid based on our experience and experimental results (e.g., RQ1 and RQ2), there are cases where anomalous log templates can burst within a short period, leading to a high occurrence count. Nevertheless, such scenarios typically indicate a severe system problem and can be easily identified by on-call engineers via other measurements, such as the key performance indicators (e.g., heartbeat). The second external threat is the study's object. Our motivation for a lightweight and adaptive method is driven by real-world scenarios in a single cloud system, namely Huawei Cloud. However, Huawei Cloud is a representative world-leading cloud provider with a vast scale. Besides, we adopt public datasets for evaluation. Hence, the evaluation results should be representative and compelling.

Internal validity. Implementation and parameter settings are the primary internal threats to the validity of our study. To mitigate these threats, we have implemented several measures. First, we have employed peer code review while implementing ScaleAD, and for the baseline methods, we have utilized open-sourced code released by the original paper or highly-rated replications on Github. Second, we have used the hyperparameters recommended by the original authors wherever available. If the hyperparameters were not available, we performed an intensive hyperparameter tuning process, such as grid search, to optimize the models for baseline methods. Third, to ensure the reproducibility of our results, we have made our code and partial data publicly available.

6 RELATED WORK

Prior research related to log-based anomaly detection can be broadly categorized into two classes, *machine learning (ML)-based* methods and *deep learning (DL)-based* methods.

Machine learning-based methods include the use of principal component analysis (PCA) by Xu et al.[37] for mining system problems from console logs and the work of Lou et al.[22], who detects system anomalies by mining invariants among log messages. Lin et al.[19] proposed LogCluster, which recommends representative log sequences for problem identification by clustering similar log sequences. He et al.[12] proposed Log3C, which incorporates system monitoring metrics into the identification of high-impact issues in service systems. Loglizer [13] provides a comprehensive evaluation of using ML-based methods for log-based anomaly detection.

These ML-based methods generally require fewer computation resources for execution, however, they are not tailored for addressing evolving logs as in a complex cloud system. Differently, ScaleAD achieves better efficiency due to our lightweight design and is capable of adapting ever-changing logs.

Deep learning-based methods for log-based anomaly detection have been approached through various methods. One such method, proposed by Du et al.[7], is Deeplog, which utilizes a Long Short-Term Memory (LSTM) network to model log sequences. LogAnomaly[24] further utilizes log count vectors and log semantic vectors to model log sequences more comprehensively. However,

both Deeplog and LogAnomaly are trained in an unsupervised manner, which has been shown to be less effective than supervised models [17]. Typical supervised solutions include a CNN-based approach [23] and GRU-based LogRobust [40]. Since labeled data is usually insufficient due to the labor-intensive nature of manual labeling, the semi-supervised method, PLElog [39], addresses this problem via label probabilistic estimation.

DL-based methods utilize different neural network structures (*i.e.*, LSTM and Transformers) to capture patterns from historical log messages. However, these structures are too complex in terms of time and space complexity to be deployed locally in an instance. Additionally, these methods still encounter accuracy degradation when logs change. In contrast, ScaleAD utilizes a TDA for local and efficient anomaly detection, which is continuously improved by a remote expert.

7 CONCLUSION

In this paper, we presented ScaleAD, a scalable and adaptive log-based anomaly detection framework designed to meet the practical requirements of accuracy, lightweight design, and adaptiveness in cloud systems. ScaleAD utilizes a trie-based detection agent (TDA) for lightweight and adaptive anomaly detection in a streaming manner. We also incorporate expert feedback, including utilizing large language models (LLMs) as an expert, to continuously enhance the system's accuracy. Experimental results on two public datasets and an industrial dataset from CloudX showed that ScaleAD is effective, achieving F1 scores between 0.908 and 0.990. Moreover, ScaleAD maintained high and consistent accuracy, even in the presence of evolving log data, while being 2 to 10× faster and requiring 5% to 41% memory resources compared to existing methods. Overall, our proposed ScaleAD framework provides a practical and efficient solution for log-based anomaly detection in cloud systems, while highlighting the significance of combining human expertise and machine learning techniques for improved performance.

REFERENCES

[1] Peter Bodik, Moises Goldszmidt, Armando Fox, Dawn B Woodard, and Hans Andersen. 2010. Fingerprinting the datacenter: automated classification of performance crises. In *Proceedings of the 5th European conference on Computer systems*. 111–124.

[2] Junjie Chen, Xiaoting He, Qingwei Lin, Yong Xu, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2019. An empirical investigation of incident triage for online service systems. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 111–120.

[3] Junjie Chen, Shu Zhang, Xiaoting He, Qingwei Lin, Hongyu Zhang, Dan Hao, Yu Kang, Feng Gao, Zhangwei Xu, Yingnong Dang, et al. 2020. How incidental are the incidents? characterizing and prioritizing incidents for large-scale online service systems. In *Proceedings of the 35th International Conference on Automated Software Engineering (ASE)*. 373–384.

[4] Mike Chen, Alice X Zheng, Jim Lloyd, Michael I Jordan, and Eric Brewer. 2004. Failure diagnosis using decision trees. In *International Conference on Autonomic Computing, 2004. Proceedings.* IEEE, 36–43.

[5] Zhuangbin Chen, Yu Kang, Liqun Li, Xu Zhang, Hongyu Zhang, Hui Xu, Yangfan Zhou, Li Yang, Jeffrey Sun, Zhangwei Xu, et al. 2020. Towards intelligent incident management: why we need it and how we make it. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. 1487–1497.

[6] Zhuangbin Chen, Jinyang Liu, Wenwei Gu, Yuxin Su, and Michael R Lyu. 2021. Experience report: Deep learning-based system log analysis for anomaly detection. *arXiv preprint arXiv:2107.05908* (2021).

[7] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 1285–1298.

[8] Robert M French. 1999. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences* 3, 4 (1999), 128–135.

[9] Supriyo Ghosh, Manish Shetty, Chetan Bansal, and Suman Nath. 2022. How to fight production incidents? an empirical study on a large-scale cloud service. In *Proceedings of the 13th Symposium on Cloud Computing*. 126–141.

[10] Haryadi S Gunawi, Mingzhe Hao, Riza O Suminto, Agung Laksono, Anang D Satria, Jeffry Adityatama, and Kurnia J Eliazar. 2016. Why does the cloud stop computing? lessons from hundreds of service outages. In *Proceedings of the Seventh ACM Symposium on Cloud Computing (SOCC)*. 1–16.

[11] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. 2017. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE international conference on web services (ICWS)*. IEEE, 33–40.

[12] Shilin He, Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, Michael R Lyu, and Dongmei Zhang. 2018. Identifying impactful service system problems via log analysis. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 60–70.

[13] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. 2016. Experience report: System log analysis for anomaly detection. In *2016 IEEE 27th international symposium on software reliability engineering (ISSRE)*. IEEE, 207–218.

[14] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. 2020. Loghub: a large collection of system log datasets towards automated log analytics. *arXiv preprint arXiv:2008.06448* (2020).

[15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[16] Van-Hoang Le and Hongyu Zhang. 2021. Log-based anomaly detection without log parsing. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 492–504.

[17] 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*. 1356–1367.

[18] Yinglung Liang, Yanyong Zhang, Hui Xiong, and Ramendra Sahoo. 2007. Failure prediction in ibm bluegene/l event logs. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*. IEEE, 583–588.

[19] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. 2016. Log clustering based problem identification for online service systems. In *Proceedings of the 38th International Conference on Software Engineering Companion*. 102–111.

[20] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation forest. In *2008 eighth IEEE international conference on data mining*. IEEE, 413–422.

[21] Jinyang Liu, Jieming Zhu, Shilin He, Pinjia He, Zibin Zheng, and Michael R Lyu. 2019. Logzip: extracting hidden structures via iterative clustering for log compression. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 863–873.

[22] Jian-Guang Lou, Qiang Fu, Shengqi Yang, Ye Xu, and Jiang Li. 2010. Mining Invariants from Console Logs for System Problem Detection.. In *USENIX annual technical conference*. 1–14.

[23] Siyang Lu, Xiang Wei, Yandong Li, and Liqiang Wang. 2018. Detecting anomaly in big data system logs using convolutional neural network. In *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. IEEE, 151–158.

[24] 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.

[25] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[26] Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heinz, and Dan Roth. 2021. Recent advances in natural language processing via large pre-trained language models: A survey. *arXiv preprint arXiv:2111.01243* (2021).

[27] Jayashree Mohan, Amar Phanishayee, Janardhan Kulkarni, and Vijay Chidambaram. 2022. Looking Beyond {GPUs} for {DNN} Scheduling on {Multi-Tenant} Clusters. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 579–596.

[28] Adam Oliner and Jon Stearley. 2007. What supercomputers say: A study of five system logs. In *37th annual IEEE/IFIP international conference on dependable systems and networks (DSN'07)*. IEEE, 575–584.

[29] OpenAI. 2022. Introducing ChatGPT. https://openai.com/blog/chatgpt

[30] Fabian Pedregosa and Philippe Gervais. 2022. Memory Profiler. https://github.com/pythonprofilers/memory_profiler

[31] Alban Siffer, Pierre-Alain Fouque, Alexandre Termier, and Christine Largouet. 2017. Anomaly detection in streams with extreme value theory. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data*

Mining. 1067–1075.

[32] Carmine Vassallo, Sebastian Proksch, Harald C Gall, and Massimiliano Di Penta. 2019. Automated reporting of anti-patterns and decay in continuous integration. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 105–115.

[33] Xuheng Wang, Xu Zhang, Liqun Li, Shilin He, Hongyu Zhang, Yudong Liu, Lingling Zheng, Yu Kang, Qingwei Lin, Yingnong Dang, et al. 2022. SPINE: a scalable log parser with feedback guidance. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1198–1208.

[34] Yaohui Wang, Guozheng Li, Zijian Wang, Yu Kang, Yangfan Zhou, Hongyu Zhang, Feng Gao, Jeffrey Sun, Li Yang, Pochian Lee, et al. 2021. Fast outage analysis of large-scale production clouds with service correlation mining. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 885–896.

[35] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. 2022. MLaaS in the wild: Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, 945–960.

[36] Wei Xu. 2010. *System problem detection by mining console logs*. University of California, Berkeley.

[37] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. 2009. Largescale system problem detection by mining console logs. In *Proceedings of SOSP*, Vol. 9. Citeseer, 1–17.

[38] Lin Yang, Junjie Chen, Zan Wang, Weijing Wang, Jiajun Jiang, Xuyuan Dong, and Wenbin Zhang. 2021. Plelog: Semi-supervised log-based anomaly detection via probabilistic label estimation. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 230–231.

[39] 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. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1448–1460.

[40] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, et al. 2019. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 807–817.

[41] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R Lyu. 2019. Tools and benchmarks for automated log parsing. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 121–130.