

Scalable and Adaptive Log-based Anomaly Detection: A Synergistic Approach

JINYANG LIU, The Chinese University of Hong Kong, Hong Kong
JUNJIE HUANG, The Chinese University of Hong Kong, Hong Kong
ZHIHAN JIANG, The Chinese University of Hong Kong, Hong Kong
YINTONG HUO, Singapore Management University, Singapore
ZHUANGBIN CHEN, Sun Yat-sen University, China
JIAZHEN GU, The Chinese University of Hong Kong, Hong Kong
CONG FENG, Huawei Cloud Computing Technology Co., Ltd., China
MINZHI YAN, Huawei Cloud Computing Technology Co., Ltd., China
MICHAEL R. LYU, The Chinese University of Hong Kong, Hong Kong

System logs are critical for software reliability. While many automated log-based anomaly detection methods exist, they often falter in large-scale cloud systems due to high resource consumption and poor adaptability to evolving logs. In this paper, we present SeaLog, an accurate, lightweight, and adaptive log-based anomaly detection framework that synergistically integrates the efficiency of traditional data-driven methods with the zero/few-shot learning ability of large language models (LLMs). SeaLog consists of a lightweight detection agent and a backbone analyzer. The detection agent utilizes an n-gram probabilistic tree (NPT) for efficient anomaly detection. The backbone analyzer employs an LLM enhanced by in-context learning (ICL) to provide accurate and adaptive predictions, especially for unseen log data. Both components continuously benefit from anomalies confirmed by on-site engineers. We evaluate SeaLog on three public datasets (BGL, Thunderbird, and HDFS) and an industrial dataset, with multiple LLM backbones, including open-source models, to assess its robustness to model choice. SeaLog achieves strong effectiveness across all datasets (F1 scores of 0.949–0.999), runs 2× to 5× faster, and consumes only 5% to 41% of the memory resources. Furthermore, we have successfully deployed SeaLog in Huawei Cloud, from which we share our insights and experiences during this deployment.

CCS Concepts: • **Software and its engineering** → **Maintaining software**.

Additional Key Words and Phrases: Log-based Anomaly Detection, Log Analysis, Large Language Models, Cloud Systems, Software Reliability

Authors' Contact Information: Jinyang Liu, jyliu@cse.cuhk.edu.hk, The Chinese University of Hong Kong, Hong Kong, Hong Kong; Junjie Huang, jjhuang23@cse.cuhk.edu.hk, The Chinese University of Hong Kong, Hong Kong, Hong Kong; Zhihan Jiang, zhjiang22@cse.cuhk.edu.hk, The Chinese University of Hong Kong, Hong Kong, Hong Kong; Yintong Huo, ythuo@smu.edu.sg, Singapore Management University, Singapore, Singapore; Zhuangbin Chen, chenzhb36@mail.sysu.edu.cn, Sun Yat-sen University, China; Jiazhen Gu, jiazhengu@cse.cuhk.edu.hk, The Chinese University of Hong Kong, Hong Kong, Hong Kong; Cong Feng, fengcong5@huawei.com, Huawei Cloud Computing Technology Co., Ltd., China; Minzhi Yan, minzhiyan@huawei.com, Huawei Cloud Computing Technology Co., Ltd., China; Michael R. Lyu, lyu@cse.cuhk.edu.hk, The Chinese University of Hong Kong, Hong Kong, Hong Kong.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM 1557-7392/2026/9-ART
<https://doi.org/10.1145/3846182>

1 Introduction

Cloud computing has surged in popularity in recent years. Large-scale cloud vendors, such as AWS, Azure, and GCP, provide 7×24 services to customers all over the world. Ensuring the reliability of cloud systems is a critical task [5, 7, 14], since a small period of downtime could result in significant financial loss for both cloud vendors and their customers [4]. 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 [33]. 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 [8, 16, 17, 65, 66].

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) [36], support vector machine (SVM) [34], decision tree (DT) [6], and 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 [57, 62] or Bidirectional Encoder Representations from Transformers (BERT) [27], 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 machines) 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 [27, 62] require heterogeneous accelerators (*e.g.*, GPUs) [44, 52] 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 [51]), 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 sufficiently adaptive to deal with diverse and evolving log data. In cloud systems, frequent launches of new software versions 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 [55]. Existing methods typically train models to learn anomaly patterns from historical log data, which can hardly be adjusted to unseen logs, causing performance degradation. To address this problem, a recent study [62] 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 has not been well investigated.

In this paper, we conduct a study on the logs in Huawei Cloud to better understand the characteristics of logs in production cloud systems. We observe that an instance could generate several gigabytes of logs daily, while 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. Specifically, during a one-month-long development cycle of 20 microservices, approximately 14.5% of new logs are introduced on average. When comparing the semantic similarities between two versions of the same microservice, we find that more than 90% of log message pairs share little semantic similarity. This implies that learning semantics from previous logs is

not adaptive enough to handle new logs. Based on these findings, 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, existing “adaptive” attempts (e.g., RobustLog [62]) rely on compute-intensive neural networks to apply learned semantic information to new logs (i.e., not lightweight). Additionally, neither type of method has the few-shot ability for swift adaptation, meaning they cannot handle log data dissimilar to historical data, which is common in real-world scenarios.

Our work. Inspired by the recent development of large language models (LLMs), we propose to *integrate the superior few-shot ability of LLMs and traditional machine-learning methods* in a synergistic manner to meet these demanding requirements simultaneously. This paradigm of integrating lightweight models with LLMs has recently gained traction in the log analysis community [42, 67]. Existing approaches in this paradigm require fine-tuning small language models or training deep detection models. In contrast, SeaLog is the first framework to adopt a zero-training detection agent that learns incrementally from confirmed labels. Specifically, we propose a novel log-based anomaly detection framework, named SeaLog, which consists of two components i.e., (1) a lightweight *detection agent*, and (2) an accurate and adaptive *backbone analyzer*. The backbone analyzer, powered by LLMs, is designed to identify anomalies by directly comprehending the semantics of logs. Since LLMs have been trained on an extensive natural language corpus, it becomes feasible to effectively identify anomalies even for unseen logs, hence achieving adaptiveness. However, solely querying LLMs is too expensive to analyze the entire set of logs, especially considering that each query to LLMs would incur charges based on the length of input data, and it is time-consuming to process a large dataset. To overcome this issue, our framework involves a lightweight detection agent to interact with the backbone analyzer. The agent can filter the majority of normal log data efficiently and forward a limited number of suspicious logs to the backbone analyzer for semantic-based analysis. In doing this, the detection agent can be deployed within resource-constrained instances without incurring voluminous network transmission costs. Additionally, we allow both the detection agent and the backbone analyzer to take human feedback, which further enhances the adaptability of SeaLog to evolving log data in practice.

To evaluate the performance of SeaLog, we conduct extensive experiments on three widely used public datasets and an industrial dataset collected from the production environment of Huawei Cloud. The experimental results demonstrate that compared with state-of-the-art solutions, SeaLog achieves strong performance (F1 scores of 0.949 to 0.999) across all datasets, attaining the highest F1 on all four benchmarks, in the setting where log data do not evolve. In addition, SeaLog retains a high and stable accuracy even if log messages evolve. Moreover, compared with existing methods, SeaLog is 2× to 5× faster and its detection agent consumes only 5% to 41% of the memory. This paper makes the following contributions:

- We propose SeaLog, a synergistic framework fusing the advantages of machine-learning approaches and LLMs for log-based anomaly detection. It comprises a lightweight detection agent that learns incrementally without gradient-based training and a backbone analyzer to meet practical requirements, i.e., accurate, lightweight, and adaptive.
- We conduct extensive evaluations of SeaLog on both public and industrial datasets. The results demonstrate that SeaLog outperforms state-of-the-art methods in processing fixed or evolving log data, runs 2× to 5× faster and only consumes 5% to 41% of the memory resources.
- We have deployed SeaLog in a real-world cloud system (i.e., Huawei Cloud) for twelve months, which demonstrates stable performance (i.e., ~0.9 F1 on average) running in our production environment. We share our experience and open-source SeaLog to benefit the community.

2 Background and Motivation

In this section, we first discuss the background of log-based anomaly detection. Then, we conduct a study to understand the attributes of logs in cloud systems, from which we derive the industrial requirements that guide 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* [8, 17, 28]. In the log parsing step, these methods obtain log templates by 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, 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 [17] 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 has long been recognized as an important problem for software maintenance, their practical application in cloud scenarios is still understudied. Unlike traditional software, cloud systems have a larger scale, generating massive volumes of diverse logs from millions of physical and virtual instances. We investigate the log data characteristics in Huawei Cloud through the following study.

2.2 Characteristics of Log Data in Cloud Systems

In the following study, we aim to understand the characteristics of log data in a typical cloud system, *i.e.*, Huawei Cloud; then summarize practical requirements for log-based anomaly detection methods in cloud systems.

2.2.1 Massive and Distributed Log Data. The huge volume of logs has been widely recognized as an essential challenge in recent studies [38, 51]. Addressing this issue is increasingly challenging within the context of the widely adopted microservice architecture of cloud systems [32]. For instance, an application may comprise hundreds of microservices [41, 64], with each one generating tens of gigabytes of log data per day as it supports a large number of users. Transmitting the complete log data of each microservice to a centralized machine for automated analysis is typically impractical, as it can consume significant network bandwidth and can also lead to time delays in promptly identifying anomalies in the runtime. Therefore, it is crucial to directly conduct log-based anomaly detection in a distributed manner within each instance. However, an instance’s main functionality (*e.g.*, video stream) 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 instances allocated for control plane services (used for management and coordination of various resources) have only ~200MB memory and a single CPU core that can be allocated to an anomaly detector without adversely affecting the main functionality running on the instance. *Therefore, it is crucial for the anomaly detector to be lightweight enough, enabling it to be deployed locally within the instances for continuous monitoring.*

2.2.2 Diverse and Evolving Log Data. Log data can evolve frequently due to software changes. Though previous studies have attempted to address this problem [19], they rely on semantic similarity between unseen and seen logs. However, this may not hold in cloud systems which have significantly diverse log data. To better understand the characteristics of log evolution in Huawei Cloud, we collected log data generated by 20 microservices of the same application from February 1, 2023 to March 1, 2023. Then, we study the following two aspects:

(1) How many log templates are shared across different microservices?

To answer this question, we obtain the templates of the collected log data by running the log parser Drain [15] following previous studies [51, 62]. Subsequently, we form pairs of every two microservices and compute their

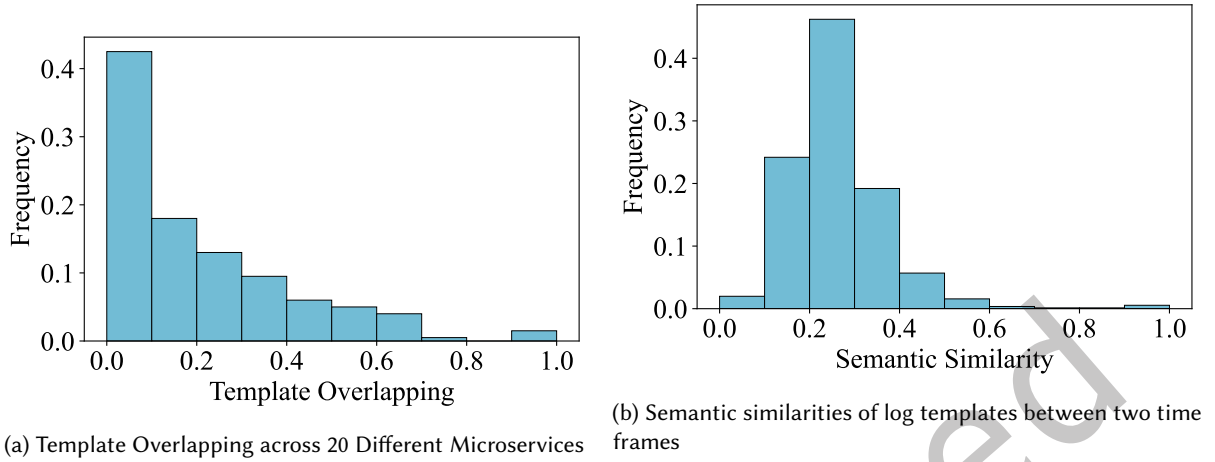


Fig. 1. Statistics of log messages in Huawei Cloud

template overlapping using Jaccard similarity, which is the ratio between the number of common templates and the total number of unique templates in the two microservices. As depicted in Figure 1a, approximately 40% of microservice pairs exhibit no overlapping log templates. Furthermore, the proportion of pairs with a similarity measure of less than 0.5 constitutes roughly 80% of the total pairs. This shows that microservices with different functionalities tend to log differently, though they share a small number of log templates as they may involve some common components while developing.

(2) How similar are the logs in terms of semantics while the software is evolving?

Moreover, we also investigate the evolution of log semantics over time. To achieve this, we partition all log data based on the date of February 15, 2023 and then proceed to calculate the semantic similarity between each pair of log messages before and after this particular time point within each microservice. Specifically, we utilize the OpenAI embedding service [48] to obtain a semantic vector for each log message. Then, we measure the semantic similarity of two log messages in terms of the cosine similarity of their corresponding semantic vectors. We plotted the cumulative distribution function (CDF) of the normalized similarities, which range from 0 to 1, for these log pairs in Figure 1b. The figure illustrates that the majority of log pairs (over 95%) exhibit similarities of less than 0.5. These findings indicate that software updates have the potential to introduce new log messages that differ significantly from historical logs.

In summary, log data in cloud systems exhibits diversity, with different microservices having distinct logs, and undergoes evolution, as new logs display minimal semantic similarities with old ones. Consequently, previous studies focusing on transferring old semantics to new ones may be ineffective. *This underscores the need for a method capable of adaptively accommodating the evolving log semantics in cloud systems.*

Insights: In addition to **accuracy**, a practical log-based anomaly detector needs to be **lightweight** enough for deploying as a plug-in of resource-constrained instances. Furthermore, it needs to be **adaptive** to evolving log semantics, enabling effective identification of anomalies within the dynamic landscape of cloud environments.

3 Methodology

3.1 Overview

Figure 2 provides an overview of the workflow of SeaLog. First, the complete set of raw input logs is processed by a detection agent to identify suspicious log windows. We design a lightweight n-gram probabilistic tree

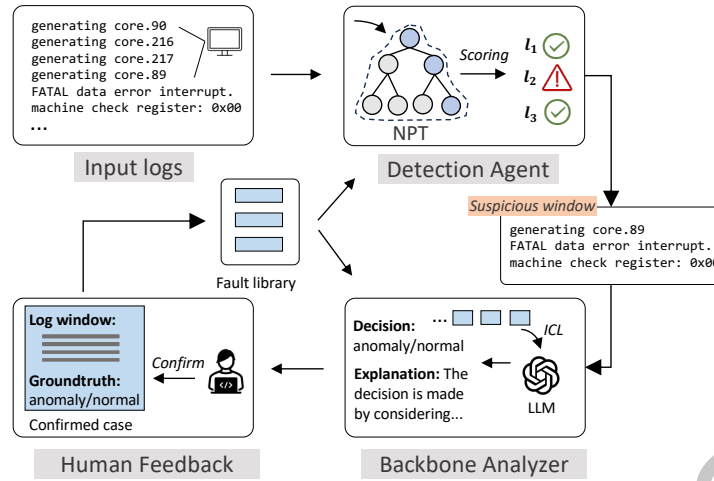


Fig. 2. The overall workflow of SeaLog

(NPT) to serve as the core of the detection agent. Its primary objective is to efficiently filter out the majority of normal logs, thereby leaving a limited number of suspicious log windows for further analysis. Second, a backbone analyzer determines anomalies and provides explanations by leveraging the zero/few-shot capabilities [3] of large language models (LLMs), such as ChatGPT [49], which can comprehend the semantics of log messages and facilitate accurate and adaptive anomaly detection. In addition, we employ in-context learning (ICL) [3, 54] to learn from historical faults for more precise decisions (anomaly/normal) and explanations. Finally, the anomalies are sent to on-site engineers for confirmation, and the confirmed cases (the raw log window and its ground-truth label) are archived into a fault library. These cases serve as continuous feedback to enhance the accuracy of both the detection agent and backbone analyzer, further enhancing the adaptiveness of our framework SeaLog. The synergy between the detection agent and backbone analyzer is pivotal in SeaLog. The detection agent *efficiently* filters out the majority of normal logs, allowing for more *accurate* and *adaptive* analysis through the use of more resource-intensive computation methods (e.g., LLM). This is because only a small number of queries require analysis, leading to decreased bandwidth, token costs, and analysis time.

3.2 Detection Agent

The detection agent is designed to filter the majority of normal logs and retain a small number of suspicious log windows. Towards this goal, the detection agent prioritizes recall over precision, aiming to capture as many suspicious windows as possible. This emphasis on recall is crucial, as any undetected suspicious logs would not be forwarded to the backbone analyzer, leading to the irreversible omission of anomalies. However, this will not result in a substantial increase in the number of logs forwarded to the backbone analyzer, as anomalies are significantly less frequent than normal logs by definition. To achieve this while ensuring efficiency, we propose an n-gram probabilistic tree (NPT) as the core of the detection agent. In the following, we introduce the construction of NPT and anomaly detection based on it.

3.2.1 NPT Construction. The goal of NPT is to efficiently find unusual words or template patterns within system logs, which barely appear when the software system is running in a normal state [60], *i.e.*, suspicious anomalies. Existing anomaly detectors either adopt deep learning-based methods that are not lightweight enough [8, 9, 27], or utilize traditional machine-learning methods that cannot effectively accommodate unseen logs [17]. Consequently, they are unable to achieve the goal of the detection agent due to their limitations.

To address these limitations, we design a simple yet effective structure, namely NPT, which serves the dual purpose of parsing logs and identifying suspicious anomalies. Additionally, it has the capability to incorporate newly labeled data for adaptation (e.g., avoid duplicate false-positive samples to query the backbone analyzer). Particularly, we detect anomalies at the granularity of log windows (e.g., 10 minutes). Given one log window, the NPT is constructed in three phases: preprocessing, signature-based grouping and sequential clustering.

Preprocessing. For an input log message l_i , we follow a common practice [51, 66] to use pre-defined regular expressions to extract parameter fields such as IP address and URL. After that, we tokenize the log message with non-alphanumeric splitters (i.e., any characters that are not letters or numbers), generating log tokens (or words) denoted as $\hat{l}_i = [w_1, w_2, \dots]$.

Signature-based Grouping. We then compute a grouping signature for each log message to roughly group similar logs together. The widely-adopted log parser Drain [15] adopts common prefix words to achieve this, assuming that these logs are more likely to belong to the same template. However, this assumption may not hold true for complex log data in practical scenarios. Instead, we utilize grouping signatures with a less stringent assumption. Each grouping signature comprises three parts: i.e., inherent attributes, top-frequent tokens, and its splitter set. (1) Inherent attributes, such as logging levels (e.g., INFO and DEBUG) and components, are considered because log messages with different levels or originating from different components are more likely to belong to separate templates. (2) Top-frequent tokens, typically the most frequently occurring K tokens within a log message (usually with $K=3$), are included in the log signature. This is because they often represent constant parts of a log template [38], with English stopwords excluded. (3) The splitter set consists of unique non-alphabetical tokens within a log message, representing its format. For example, in the log message “user=root, delay=10, stat=queued” the unique splitters are “=”. We use the splitter set because logs with the same templates often share the same format [18, 24], such as the presence of “=”, indicating a key-value format. Those logs sharing the same grouping signature are sent to the same leaf node in the NPT.

Sequential Clustering. In practice, the proposed grouping signatures can already lead to leaf nodes containing cohesive log messages. Then, we are allowed to directly adopt lightweight sequential clustering [38] in each leaf node to generate more fine-grained *log clusters* effectively. Each log cluster comprises a template and its corresponding mark. The template is derived by analyzing the logs sequentially assigned to the cluster. The mark is an annotation indicating whether this template represents an anomaly or a normal event, as determined by human feedback. The mark is set to null by default if no feedback is given.

Specifically, as shown in Figure 3 (right-hand side), each log message tries matching with existing log clusters. If it cannot match with any log cluster, it forms a new cluster, with the template being the log message itself. Otherwise, the log message is assigned to an existing log cluster, and the log template associated with the cluster is updated. This is achieved by comparing the words of the existing log template with the arriving log message. The differing words are treated as parameters and marked as * in the updated log template.

3.2.2 LRU-based Anomaly Detection. After obtaining a log window, we determine anomalies by measuring the extent of word-level (i.e., unusual words) or template-level (i.e., sequence patterns) rarity of the window. To achieve this efficiently, we propose to extract word-level and template-level LRU (least recently used) sketches to inform decisions. The core idea behind the LRU sketches is to maintain a compact data structure that continuously tracks whether a specific word or template combination has recently become unusual.

LRU-Sketch Computation. We use LRU sketches to capture anomalies at both the word level and template level. For simplicity, we will only explain the word-level LRU sketch and its corresponding anomaly detection procedure, omitting the detailed discussion of the template level, which follows a similar approach. Specifically, the word-level LRU sketch is computed utilizing data from the most recently observed log windows within a specific time period. Each sketch comprises multiple variables detailed in Table 1. Specifically, ng_i^w represents the *n-gram word*, which is defined as a contiguous sequence of n words derived from a log template, denoted as

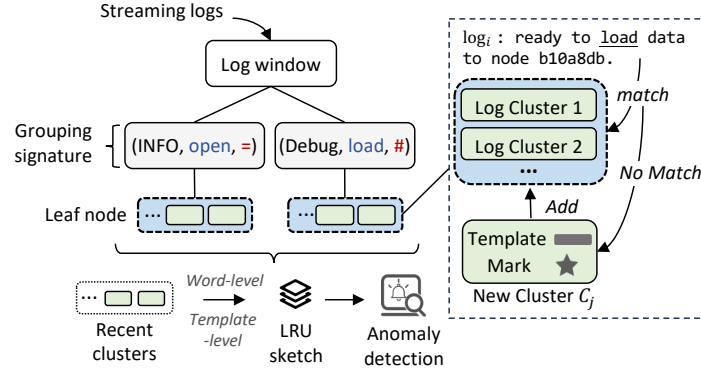


Fig. 3. The structure of n-gram probabilistic tree (NPT)

$ng_i^w = \{w_i, \dots, w_{i+n-1}\}$, and $C(ng_i^w)$ denotes its total count in the most recently seen windows. In addition, we use $C_a(ng_i^w)$ and $C_n(ng_i^w)$ to denote the total occurrences of ng_i^w shown in abnormal windows or normal windows, respectively. We continuously update these statistics in the least recently used manner, e.g., by only considering data observed within the past ten days to mitigate the impact of outdated information.

Suspicious Anomaly Identification. We then identify suspicious anomalies based on the LRU sketch, which is formulated as a binary classification problem (i.e., anomaly or normal). Specifically, we propose a streaming classifier based on Bayes' theorem [2], designed to run efficiently and with minimal parameter requirements, making it memory-friendly. After a log message L is assigned to a log cluster in NPT, we extract its associated statistical data stored in the LRU Sketch based on its n-grams. We then perform the following computation, where we omit the superscript w for simplicity:

$$p(y = 1|L) = p(y = 1) \times \prod_i p(ng_i|y = 1) \quad (1)$$

$$p(y = 0|L) = p(y = 0) \times \prod_i p(ng_i|y = 0), \quad (2)$$

$$p(ng_i|y = 1) = \frac{C_a(ng_i) + \alpha}{\sum_j ng_j + \alpha \times u}, \quad (3)$$

$$p(ng_i|y = 0) = \frac{C_n(ng_i) + (1 - \alpha)}{\sum_j ng_j + (1 - \alpha) \times u}, \quad (4)$$

where y denotes the label for anomaly detection. $p(y = 1)$ and $p(y = 0)$ are the prior probabilities for anomalous or normal log windows, respectively. These probabilities are calculated as the ratio of the number of anomalous (or normal) log windows to the total number of log windows encountered so far. The term $p(ng_i|y = 1)$ is the likelihood of the i -th n-gram indicating an anomaly as described in Equation 3 and Equation 4. The LRU sketches are typically initialized using labeled training data following existing studies [27, 62]. This data can be gathered from historical failures or through fault injection [30, 31]. Furthermore, our LRU-Sketch design can continue to evolve by updating Table 1 after new labeled data (feedback) becomes available.

In practice, unseen n-gram words can appear and their statistics are not recorded for anomaly computation. To mitigate this issue, inspired by Lidstone smoothing [53], we propose to introduce a parameter $\alpha \in [0, 1]$ to compute the likelihood as shown in Equation 3 and Equation 4. Here, u is the number of unique n-gram words (i.e., vocabulary size). In particular, we intentionally set a larger parameter, e.g., $\alpha = 0.8$. In doing this, when dealing with unseen n-gram words, we have both $C_a(ng_i) = 0$ and $C_n(ng_i) = 0$. A large α tends to increase $p(ng_i|y = 1)$ more than $p(ng_i|y = 0)$. Hence, we regard new n-gram words as having a much higher likelihood of indicating

Table 1. Content of Word-level LRU Sketch

Symbol	Content
$C(n_{g_i}^w)$	Total count of n-gram words $n_{g_i}^w$.
$C_a(n_{g_i}^w)$	Total count of n-gram words $n_{g_i}^w$ that are shown in abnormal windows.
$C_n(n_{g_i}^w)$	Total count of n-gram words $n_{g_i}^w$ that are shown in normal windows.

anomalies. This design is crucial because those missed anomalies cannot be revisited by the subsequent backbone analyzer. Therefore, the detection agent must apply a lenient standard to identify suspicious anomalies.

Finally, we average both word-level and template-level anomaly scores to obtain the final anomaly scores, *i.e.*, $\bar{p}(y = 1|L)$ and $\bar{p}(y = 0|L)$. We then normalize each of the anomaly scores within the range of $[0, 1]$ by dividing them by their summation. This normalization process allows for better comparison and more intuitive interpretation of the anomaly scores. Particularly, if a log template has been confirmed as an anomaly or not, *i.e.*, it has a non-null mark value (e.g., 0 or 1), we directly use the mark as the anomaly score. Then, we introduce a querying threshold, denoted as θ_{query} for the detection agent. The detection agent will only raise a query when the normalized anomaly score $\bar{p}(y = 1|L)$ exceeds θ_{query} . This allows us to control the number of queries raised by the detection agent.

3.3 Backbone Analyzer

We have deliberately designed the detection agent to be lightweight to efficiently process large amounts of log data. However, this approach can inevitably result in false positive alarms, particularly when encountering unseen tokens or sequence patterns in the log data. To tackle this issue, we propose utilizing large language models (LLMs) as a backbone analyzer to analyze the detection agent's suspicious results and make more accurate predictions. The reason for adopting LLM is that it has been pre-trained with an extensive corpus (e.g., source code, software documents, and bug reports [63]). This enables LLM to comprehend the log data reported by the detection agent by reasoning with the existing knowledge it has acquired. While directly processing the entire set of log data with LLM is inefficient and costly, the SeaLog framework allows us to leverage the power of LLMs once the detection agent has filtered out the majority of normal log data, assuming that anomalies are rare in most software systems [28].

However, LLMs were originally designed for open-world question answering, rather than specifically for the task of log-based anomaly detection. In order to make LLM suitable for our specific task and to provide it with domain-specific knowledge, we adopt LLM with in-context learning (ICL) as the backbone analyzer of SeaLog. ICL provides LLM with *examples* to make LLM learn from the contextual information in the examples and generate answers according to the format in the examples, which has been proved to enhance the reasoning ability of LLM significantly [3, 13, 23]. Specifically, our LLM-based backbone analyzer has three steps: *example retrieval*, *prompt formulation* and *LLM querying*.

3.3.1 Example Retrieval. Previous studies have shown that examples that are semantically similar to the query are more effective than random ones [13, 54]. We follow this idea in our backbone design. For every query window q , we select k nearest neighbors (k NN, $k=3$ by default), which are the most similar log windows, from a candidate library of annotated training logs. To achieve this, we first embed all log windows in the training data as a library of semantic vectors using the OpenAI Embedding service [48], denoted as $V = \{v_1, v_2, \dots\}$. Similarly, each query q is also embedded as a semantic vector v_q . We then retrieve k examples whose semantic vectors in V are closest to v_q measured by Euclidean distance $d(v_i, v_j) = \sqrt{\sum_u (v_{iu} - v_{ju})^2}$. By doing this, we are able to select k examples that have similar semantics to q , which facilitates LLM's prediction of q .

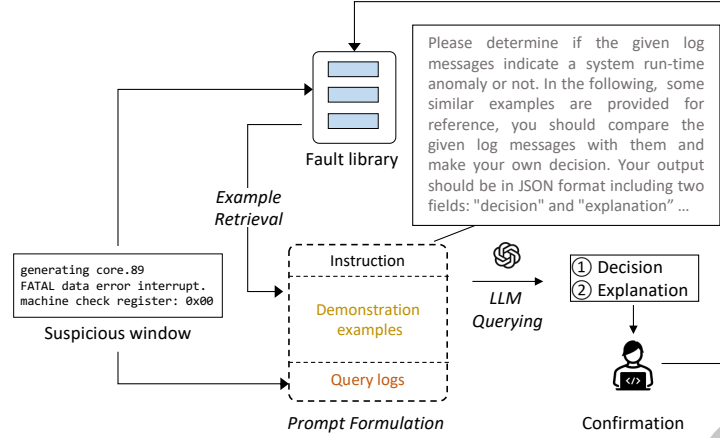


Fig. 4. Processing suspicious windows with backbone analyzer

In practice, computing Euclidean distances directly by comparing v_q with every vector in V can be time-consuming, particularly when V is large and continues to grow as more cases are accumulated during the operations of on-site engineers. To accelerate the retrieval process, we speed up the similarity search based on FAISS (Facebook AI Similarity Search) [25]. FAISS can build indices for vectors in V and partition the vector space into smaller subspaces, allowing for faster search within these subspaces.

3.3.2 Prompt Formulation. The prompt for LLM comprises three components, as depicted in Figure 4. (1) the *instruction*, which defines the log-based anomaly detection task and establishes the desired output format for LLM. Specifically, we regulate the output of LLM using JSON format, enabling precise parsing of LLM's decisions and excluding any irrelevant text generated by LLM. (2) the *examples* that are used to provide demonstrations for LLM. In the example demonstration, we order the examples by placing the more similar examples close to the query. As highlighted by recent studies [13, 54], this approach can facilitate LLM with learning relationships between the examples and the query, thus improving the overall performance of LLM. (3) the *query* that should be predicted by LLM. The log messages for prediction are directly appended after the instructions and examples. Due to space limitations, we include the specific details of the prompt in our replication package.

3.3.3 LLM Querying. We then utilize the constructed prompt to query an LLM (e.g., GPT-3.5). The responses obtained from the backbone include a decision regarding whether the input is classified as an anomaly or normal. Those anomalies are reported to on-site engineers for prompt mitigation. Additionally, the responses provide corresponding explanations that aim to aid on-site engineers in confirming the classifications. After a case is confirmed, the input log window and its confirmed label are stored in the fault library to facilitate future analysis. Additionally, we allow engineers to annotate each log template as an anomaly or not, which is saved in the mark within the log cluster, further enhancing the adaptiveness of SeaLog.

Despite our empirical observations in Section 2.2 suggesting a minimal overlap in semantics among log entries, our methodology essentially differs from prior research [62]. The retrieved examples are instrumental in illustrating the anomaly detection process and the expected structure of the output [13]. Our strategy actively seeks out the most closely related examples to maximize the effectiveness of the ICL process to enhance LLM.

3.4 Deployment

Real-world deployment of SeaLog consists of two phases: offline warming-up and online detection. The *offline warming-up* phase provides the detection agent and backbone with labeled log data for training. Specifically, for

the detection agent, these log data are processed to construct the initial NPT structure and LRU sketches. For the backbone analyzer, these log data are used to build the candidate library. During the *online detection* phase, the NPT continuously processes streaming log messages and raises suspicious logs to the backbone analyzer. The backbone analyzer conducts ICL-based reasoning by referring to cases retrieved from the candidate library. Once a decision is made by the backbone analyzer and confirmed by on-site engineers, it is provided as feedback for future analysis. In Section 4.8, we share more details regarding our practical experience with the real-world deployment of SeaLog. It is worth noting that SeaLog’s detection agent is fundamentally *unsupervised*: it relies on incremental n-gram statistics (NPT and LRU Sketch) that require no labeled data to operate. During warm-up, labeled windows are used only to build the backbone’s candidate library, not to train the agent’s statistical model. During online detection, confirmed labels from the backbone analyzer are incorporated as feedback, making the overall framework *semi-supervised*. This incremental learning design allows the agent to adapt to evolving log patterns without retraining from scratch.

4 Evaluation

We answer the research questions (RQs) in this section.

- RQ1: How effective is SeaLog under the *offline* setting?
- RQ2: How effective is SeaLog under the *online* setting?
- RQ3: How does the number of queries affect the performance of SeaLog?
- RQ4: What is the time and space efficiency of SeaLog?
- RQ5: How does the choice of LLM backbone affect SeaLog’s performance?

4.1 Evaluation Setting

4.1.1 Dataset. We evaluate SeaLog on three widely used public benchmarking datasets and one industrial dataset from the production environment of Huawei Cloud. Table 2 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) [47, 65]. The Thunderbird dataset originates from a Thunderbird supercomputer at Sandia National Labs (SNL) [47, 65]. Although existing studies often use 10 million continuous lines from the Thunderbird dataset for evaluation [27, 28], they do not specify which 10 million logs they employed; therefore, we use the first 10 million logs in this study. The HDFS dataset is collected from Amazon’s EC2 platform running Hadoop-based MapReduce jobs [56, 65]. It contains over 11 million log messages organized by block IDs into sessions, where each session traces the lifecycle of an HDFS block operation. We exclude the Spirit dataset from our evaluation because its anomaly ratio exceeds 80%, which deviates substantially from realistic production scenarios where anomalies are rare events. Furthermore, prior studies that evaluate on Spirit often invert the positive and negative labels to artificially reduce the anomaly ratio [28]. We consider this practice methodologically unsound for semantic-based approaches, as it distorts the contextual meaning that log messages convey. The industrial dataset is collected from the production environment of Huawei Cloud. It contains one month of log data generated by 20 internal microservices that offer various functionalities such as API Gateway, user management, auto-scaling, load balancing, and more. Experienced engineers from our site reliability engineering (SRE) team assist us in labeling the dataset based on their historical diagnostic reports. These reports record the starting times of actual system issues, which facilitate accurately identifying the anomalous log messages manually. After a thorough labeling process, we obtained a log dataset with approximately 1.5 million lines of log messages covering a wide range of system problems. We list the details of 103 types of failures in our replication package [37]. Together, these four datasets span three categories of log sources. (1) *Supercomputer systems* (BGL, Thunderbird) feature stable log formats and hardware-centric faults. (2) *Distributed systems* (HDFS) exhibit anomalies that manifest as missing or reordered block operations within session-based log groups. (3) *Industrial microservice systems*

Table 2. Statistics of evaluation datasets

Dataset	BGL	Thunderbird	HDFS	Industry
# Log messages	4,747,963	10,000,000	11,175,629	1,488,073
# Templates	456	1,504	~47,000	3,241
# Train windows (anomaly ratio)	2,884 (21%)	416 (55%)	460,052 (1.5%)	3,048 (13%)
# Test windows (anomaly ratio)	722 (24%)	105 (30%)	115,013 (1.5%)	933 (18%)

(Industry) are characterized by diverse and rapidly evolving log templates across heterogeneous services. This breadth of evaluation enables us to assess SeaLog’s effectiveness across different anomaly types and log evolution patterns.

Log message grouping. The choice of log grouping strategy is an important design decision for anomaly detection. Prior empirical studies [8, 28, 59] have systematically compared fixed-window, sliding-window, and session-based grouping strategies, and consistently found that time-based windows better preserve temporal correlations among log events than fixed-count windows. In particular, Le and Zhang [28] showed that grouping strategy significantly impacts detection accuracy, and session-based grouping (where available) tends to outperform fixed-window approaches. Based on these findings, SeaLog adopts time-based windows (1 hour) for BGL and Thunderbird, where logs are generated continuously without explicit session identifiers, and session-based grouping by block ID for HDFS, where each block operation forms a natural session. For our industrial deployment, we use 10-minute windows to meet real-time alerting constraints.

4.1.2 Evaluation Metrics. 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) = $\frac{TP}{TP+FP}$, recall (RC) = $\frac{TP}{TP+FN}$, and F1 score = $\frac{2 \times PC \times RC}{PC+RC}$.

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

- **Loglizer (IF/LR/DT)** [17] is a log-based anomaly detection library encompassing a variety of machine learning (ML) methods, including isolation forest (IF), logistic regression (LR), decision tree (DT), etc.
- **DeepLog** [9] 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** [43] aims to detect anomalies by combining log count vectors and log semantic vectors, which also utilizes a forecasting-based mechanism to reflect anomalies.
- **RobustLog** [62] aims to capture the meaning of log messages using word embeddings to handle the challenge of constantly changing log messages. Each log message is encoded as a representation based on word vectors using an attention-based Bidirectional LSTM.
- **NeuralLog** [27] 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.

4.1.4 Implementation Details. We implemented the prototype of SeaLog with approximately 1000 lines of Python code for easy use in Huawei Cloud. We utilize ChatGPT (gpt-3.5-turbo-0301) as the LLM in the backbone analyzer,

Table 3. Experimental results of end-to-end offline anomaly detection

Method	Precision	BGL		Thunderbird			HDFS			Industry		
		Recall	F1 score	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	0.364	0.914	0.521	0.176	0.994	0.299
LR	0.738	0.437	0.549	0.842	0.516	0.640	0.999	0.992	0.995	0.818	0.655	0.727
DT	1.000	0.570	0.726	1.000	0.839	<u>0.912</u>	1.000	0.996	<u>0.998</u>	0.942	0.788	0.858
DeepLog	0.241	0.895	0.380	0.295	1.000	0.456	1.000	0.340	0.508	0.358	0.909	0.513
LogAnomaly	0.268	0.862	0.409	0.307	1.000	0.470	1.000	0.341	0.508	0.360	0.927	0.519
RobustLog	0.942	0.961	<u>0.951</u>	1.000	0.710	0.830	0.998	0.995	<u>0.997</u>	0.984	0.764	0.860
NeuralLog	0.881	0.886	0.883	0.713	0.719	0.715	0.999	0.995	0.997	0.889	0.895	<u>0.887</u>
SeaLog	0.994	0.991	0.993	1.000	0.903	0.949	1.000	0.999	0.999	1.000	0.931	0.964

which is widely recognized for its advanced language understanding capabilities. For the baselines, NeuralLog and Loglizer, we directly utilized the code provided by the respective authors. Additionally, for DeepLog, LogAnomaly, and RobustLog, we utilized the DeepLoglizer library [8] for implementation. We set $k = 3$ and evaluate the settings of the important parameter θ_{query} for SeaLog in Section 4.4. For baseline methods, we utilize the default parameters provided by the original source code. We use the same window setting for the public datasets as in [28], namely, using a fixed window size of one hour without overlapping. For the industrial dataset, we use 10 minutes as the window size for real-time anomaly detection.

4.2 Effectiveness under Offline Setting (RQ1)

In this RQ, we evaluate the effectiveness of SeaLog in an offline setting, where the training and testing sets are fixed. We evaluate the overall performance of SeaLog as well as each component’s contribution to the overall performance.

Overall Performance. Following the setting of previous studies [27, 28], 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. We use the training data for warming up SeaLog as described in Section 3.4 and training the baseline methods. Testing data is used for evaluation. The statistics of training and testing data are shown in Table 2.

The evaluation results are presented in Table 3, where the highest F1 score is emphasized in **bold**, and the second-best score is underlined. We can make the following observations: (1) Unsupervised methods (IF, DeepLog, and LogAnomaly) show significantly lower effectiveness than supervised approaches in terms of F1 score. This is mainly because they assume that all training data is normal. As a result, during the inference phase, these methods classify every window containing unseen log templates as anomalous, 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 (e.g., DT) can still achieve comparable or better performance (on Thunderbird) compared to DL-based methods. The above two observations are consistent with the recent empirical studies [8, 28] that benchmark existing DL-based methods. (3) SeaLog attains the highest F1 score on all four datasets, ranging from 0.949 to 0.999, demonstrating the effectiveness of the collaboration of the detection agent and the backbone analyzer. On HDFS, supervised baselines (DT, RobustLog, NeuralLog) achieve $F1 \geq 0.997$ thanks to rich feature engineering over block traces grouped by session. SeaLog achieves $F1 = 0.999$ (precision 1.000, recall 0.999) on HDFS, demonstrating its effectiveness on session-based distributed system logs where anomalies manifest as deviations in block operation sequences.

Performance of Individual Components. To evaluate the contribution of each component to SeaLog’s overall performance, we created three variants of SeaLog: SeaLog w/o backbone, SeaLog w/ weak backbone and SeaLog w/o in-context learning (ICL). We compared their performance with the original SeaLog in our industry dataset.

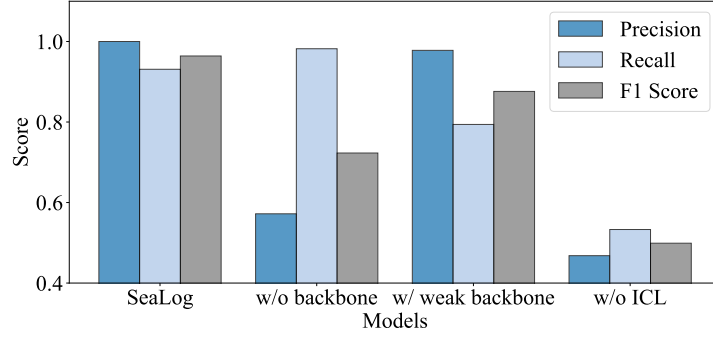


Fig. 5. Effectiveness of individual modules of SeaLog

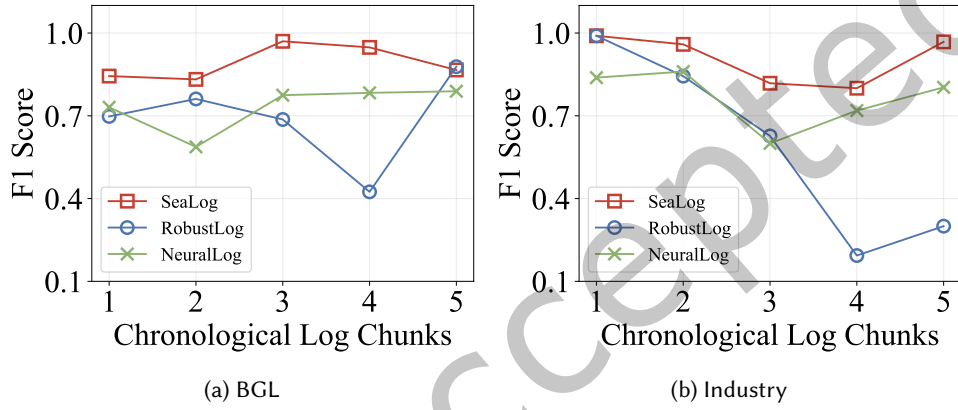


Fig. 6. Experimental results of online anomaly detection

Figure 5 shows the experimental results. We can observe that (1) The detection agent can identify most anomalies but produces a lot of false positives, as evidenced by its high recall (0.982) and low precision (0.572). These false positives are usually caused by the presence of unseen log data, as the detection agent lacks information about these data to make accurate decisions. (2) With a weak backbone, SeaLog w/ weak backbone achieves a higher precision (0.978) but still much lower recall (0.794), indicating that the weak backbone analyzer cannot accurately identify anomalies from the suspicious logs reported by the detection agent. (3) After removing the ICL design in the backbone analyzer, SeaLog w/o ICL shows significantly lower accuracy (e.g., F1 score of 0.499), proving that ICL is essential for providing the LLM with domain-specific knowledge and enhancing its accuracy in log-based anomaly detection. (4) By incorporating feedback from the backbone analyzer, the performance of SeaLog improves significantly, achieving the highest F1 score (0.964). This demonstrates that the detection agent can effectively and adaptively learn from the feedback by updating the statistics it has recorded, thereby accommodating new log data.

Answer to RQ1. SeaLog achieves the best overall performance among all state-of-the-art baselines across four benchmarking datasets, attaining the highest F1 on all four benchmarks. This demonstrates that SeaLog can effectively fuse the results of the detection agent and the backbone analyzer. Additionally, the detection agent achieves superior performance after taking the backbone analyzer’s feedback, highlighting the effectiveness of the feedback mechanism within SeaLog.

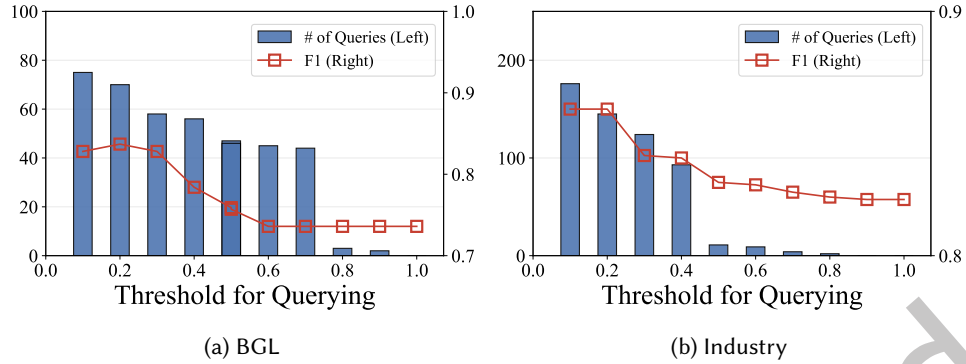


Fig. 7. Anomaly detection performance w.r.t. the number of queries

4.3 Effectiveness under Online Setting (RQ2)

In this RQ, we would like to evaluate whether SeaLog can adapt to evolving log data within an online setting. Based on the results of RQ1, we select the most effective models (i.e., RobustLog and NeuralLog) as strong baselines for this RQ. To simulate a scenario where new log messages are received, we divide the BGL and Industry datasets into six equal chunks (numbered from 0 to 5) in chronological order, each containing log messages with different templates. Note that the results of the Thunderbird dataset are similar to those of the BGL dataset, so Thunderbird is excluded to save space. 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. To mimic this scenario, we train the models using one chunk and then evaluate the trained models on the next chunk (e.g., training with chunk 0 and testing with chunk 1), resulting in five evaluation results. While RobustLog and NeuralLog were not initially designed for receiving streaming feedback, we enable them to perform online training by continuously adapting to the incoming chunk, ensuring a fair comparison with SeaLog.

Figure 6 presents the experimental results. Our observations are as follows: (1) SeaLog 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 [11] 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 shows more consistent performance than RobustLog because of its transformer architecture, which has more parameters to remember observed data and address the issue of catastrophic forgetting.

Answer to RQ2. SeaLog consistently surpasses the strong baseline methods (RobustLog and NeuralLog) under the setting that logs are evolving. The results demonstrate that SeaLog can meet the requirements of being adaptive.

4.4 Impact of Query Number (RQ3)

In this RQ, we aim to study how the number of queries affects the effectiveness of SeaLog. Specifically, we warm up the detection agent with only 50% of the training data for the BGL and Industry datasets, mimicking the scenario that the model might not be well trained initially. Then, we use the same test set as in RQ1 to evaluate

the detection agent. While testing, we vary the number of queries sent to the backbone analyzer by tuning the threshold to issue a query, *i.e.*, θ_{query} in the range of $[0, 1]$ with a step size of 0.1.

The results are presented in Figure 7. We can obtain the following observations: (1) Fewer queries are launched when increasing the threshold θ_{query} . With fewer queries, the performance of SeaLog is degraded. However, even if no query is launched (*i.e.*, setting $\theta_{query} = 1$), the detection agent can still achieve around 0.73 F1 score for BGL and 0.82 for Industry data, respectively. (2) To achieve the highest F1, the detection agent only queries the backbone analyzer fewer than 80 and 190 times on BGL and Industry datasets with millions of log lines, respectively. For the industry dataset, this indicates that at most 6 queries per day can be raised. This shows that the detection agent can filter out most normal cases and involve the compute-intensive backbone analyzer conservatively.

Threshold Sensitivity Analysis. We further investigate the sensitivity of the detection agent’s performance to the querying threshold θ_{query} . On BGL, we observe that varying θ_{query} from 0.1 to 0.9 produces identical detection results. This is because the detection agent assigns anomaly scores that are highly polarized: all suspicious windows receive scores exceeding 0.99, while normal windows receive scores close to 0. Consequently, any threshold within the range $[0.1, 0.9]$ yields the same set of suspicious windows forwarded to the backbone analyzer. This property simplifies the practical deployment of SeaLog, as the threshold does not require careful tuning.

Answer to RQ3. The performance of the detection agent improves as the amount of feedback increases. Owing to the NPT-based anomaly detection approach, most normal log messages are filtered out, resulting in only 190 queries in total (*i.e.*, 6 queries per day) for the industry dataset. Moreover, the detection threshold θ_{query} is robust: any value in $[0.1, 0.9]$ yields identical results on BGL due to the highly polarized anomaly scores, eliminating the need for careful tuning.

4.5 Time and Space Efficiency (RQ4)

(1) *Time efficiency.* We measure the time required to perform anomaly detection on the complete BGL test set in comparison with all baseline methods. 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 [15] for baseline methods. Note that I/O time is excluded for all methods, and all methods are executed without GPU acceleration, which may not be available when resources are constrained. For SeaLog, we account for the time taken by the detection agent and backbone analyzer. The detection agent’s time includes all steps from preprocessing to anomaly detection. To simulate a real-world deployment scenario, all queries sent to the backbone analyzer are conducted asynchronously, allowing the execution of the detection agent to proceed without being blocked. The backbone analyzer’s time is calculated as the total time needed to process all queries. Note that the communication time between the detection agent and the backbone analyzer is negligible due to the limited number of small-size queries launched.

Figure 8a 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) SeaLog achieves comparable efficiency to Loglizer, being 2 to 5 times faster than other deep learning-based methods. While SeaLog relies on a costly LLM as the backbone for accuracy, it also ensures efficiency because the LLM only needs to process a small number of queries from the detection agent. As shown in Figure 8a, it takes around 10 seconds to process all the queries asynchronously. (2) DeepLog and LogAnomaly with similar LSTM-based architectures exhibit comparable performance. In contrast, RobustLog and NeuralLog require more time than others, as they involve a greater number of parameters. Specifically, the detection agent achieves a throughput of approximately 53,000 log messages per second with a median (p50) latency of 3 milliseconds per log message, demonstrating its suitability for real-time deployment in production environments.

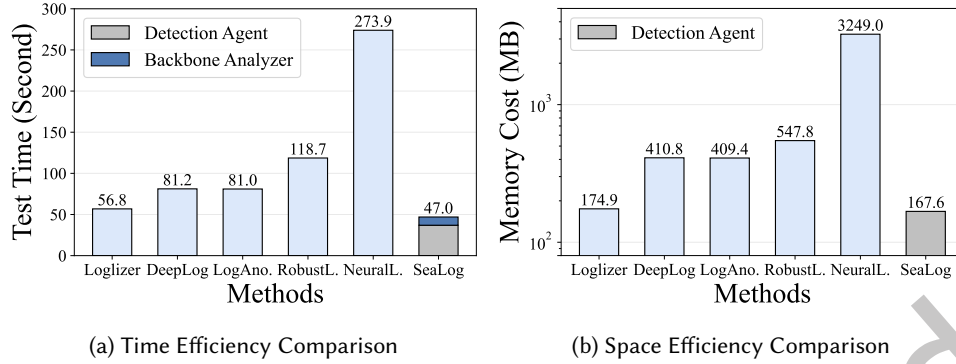


Fig. 8. Experimental results of time and space efficiency comparison

(2) *Space efficiency.* We use Memory Profiler [50] to monitor the memory usage of each method when processing a single log window, as anomaly detection is usually performed on a window-by-window basis. Figure 8b displays the comparison results. SeaLog demands the least memory during log processing, using only 5% to 41% of the memory consumed by DL-based methods. This is primarily because SeaLog only needs to store the NPT structure in memory, which is typically sparse. Loglizer demonstrates comparable space efficiency to SeaLog, since it only stores the count vector of a log window, and the subsequent ML-based method (e.g., LR) is memory-efficient. DL-based methods require 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.

Answer to RQ4. When compared to recent deep learning-based methods, SeaLog is 2 to 5 times faster and requires only 5% to 41% of the memory consumed by the baselines. These results demonstrate that SeaLog meets the requirement of being lightweight for practical anomaly detection.

4.6 Multi-LLM Backbone Evaluation (RQ5)

To investigate the impact of different LLM backbones on SeaLog's performance, we evaluate six LLMs as backbone analyzers on the BGL dataset, including two open-source models (DeepSeek-V3, Qwen3-235B-A22B) and four commercial models (GPT-3.5-Turbo, GPT-4o-mini, GPT-4o, GPT-4.1-mini). We also compare against the Agent-Only baseline (no backbone) and a TrainedBackbone variant that uses a Naive Bayes classifier trained on domain-specific token statistics instead of an LLM.

Table 4 presents the results. We observe that: (1) All commercial LLMs achieve similar F1 (0.598–0.619), while the open-source Qwen3-235B-A22B significantly outperforms them (F1=0.740), demonstrating that open-source models can match or exceed commercial alternatives. (2) The TrainedBackbone (F1=0.890) substantially outperforms all LLM backbones, confirming that domain-specific token statistics remain highly effective when sufficient labeled data is available. (3) Open-source models achieve performance competitive with or superior to that of commercial models, enabling fully on-premise deployment without sacrificing accuracy.

Nondeterminism Analysis. Since LLMs are known to produce nondeterministic outputs, we assess the reproducibility of LLM-based detection by repeating the full evaluation pipeline three times using GPT-4o-mini with temperature=0. The F1 scores across the three runs are 0.597, 0.603, and 0.601, yielding a mean of 0.600 and standard deviation of 0.003. Only 2 out of 721 windows received different predictions across runs, confirming that the LLM backbone produces highly deterministic results at temperature=0.

¹OpenAI: <https://openai.com/api/pricing/>; DeepSeek: https://api-docs.deepseek.com/quick_start/pricing; Alibaba Cloud: <https://www.alibabacloud.com/help/en/model-studio/model-pricing>.

Table 4. Multi-LLM backbone comparison on BGL (721 test windows, 314 backbone calls)

Backbone	P	R	F1	p50 (s)	Cost (\$)
Agent-Only	0.432	1.000	0.603	–	–
TrainedBackbone	0.816	0.978	0.890	–	–
GPT-3.5-Turbo	0.430	0.978	0.598	1.35	0.50
GPT-4o-mini	0.436	0.978	0.603	1.70	0.15
GPT-4o	0.454	0.941	0.612	2.55	2.61
GPT-4.1-mini	0.475	0.890	0.619	2.26	0.42
DeepSeek-V3 [†]	0.431	0.963	0.596	3.0	0.30
Qwen3-235B-A22B [†]	0.623	0.912	<u>0.740</u>	8.4	0.34

[†] Open-source model; deployment cost depends on infrastructure.

Cost estimated from official API pricing (Feb. 2026)¹.

Answer to RQ5. Open-source models are competitive with commercial LLMs, with Qwen3-235B-A22B achieving the best LLM F1 (0.740). The TrainedBackbone remains the strongest overall (F1=0.890), confirming that domain-specific approaches outperform general-purpose LLMs. These results show that SeaLog is robust to the choice of LLM backbone and supports both commercial and open-source models.

4.7 Case Study

Having quantitatively evaluated SeaLog’s effectiveness and efficiency, we now qualitatively examine how the detection agent and backbone analyzer collaborate in practice. We conduct a case study on 30 representative windows from the BGL test set. We categorize these windows into three groups based on the agent’s and backbone’s decisions:

False Positives Filtered (FP_filtered). These windows are flagged as suspicious by the detection agent but correctly identified as normal by the backbone analyzer. For example, windows containing hardware error correction logs (e.g., “ddr error(s) detected and corrected”) receive high anomaly scores from the agent due to rare n-grams like “EDRAM error.” However, the backbone analyzer, leveraging similar training examples retrieved via TF-IDF, correctly classifies these as routine corrective events.

True Positives Confirmed (TP_confirmed). These are genuine anomalies detected by both the agent and the backbone. Typical examples include kernel fatal errors and machine check interrupts, where both the statistical rarity captured by the agent and the semantic severity recognized by the backbone align. For instance, the backbone explains one such window as: “*This log window shows repeated errors related to message codes not matching expected values and multiple severed links, suggesting a potential issue with network communication or system stability.*”

False Negatives Missed (FN_missed). A small number of anomalous windows are missed when the backbone incorrectly overrides the agent’s correct suspicion. This occurs when the retrieved training examples are predominantly normal despite containing similar keywords, leading the backbone to a false negative decision. In one such case, the backbone states: “*This log window indicates normal system behavior as it shows no critical input interrupts and consistent interrupt handling without any reported errors or anomalies,*” despite the window containing an actual fault that the agent correctly flagged (score = 1.0).

This case study demonstrates the complementary strengths of the two components: the detection agent provides high recall through statistical rarity detection, while the backbone analyzer improves precision through semantic understanding. The primary failure mode is backbone misjudgment when retrieved examples are not representative.

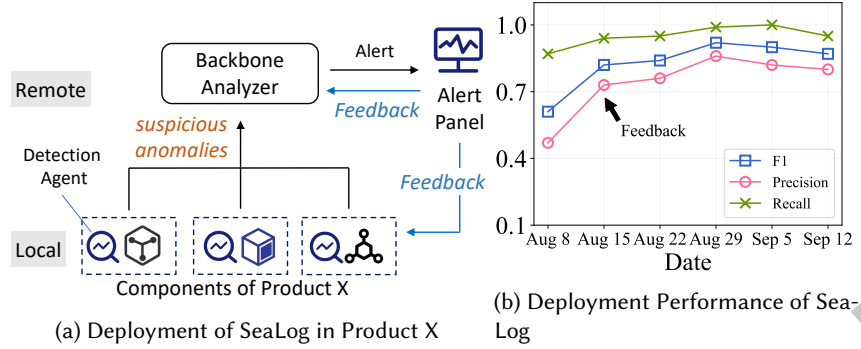


Fig. 9. Practical application of SeaLog in Huawei Cloud

4.8 Industrial Deployment Experience

We share our experience deploying SeaLog in Product X, a unified data center management platform. This platform offers multiple functionalities, including streamlined service orchestration, robust security measures, and access control mechanisms. Product X comprises multiple components, each consisting of *hundreds of microservices*. In the early stage of Product X, we conduct log analysis in a centralized manner, where all log data are transmitted to a high-performance server for manual analysis or utilizing tools like log parsers [66] to assist in the analysis process. This approach was suitable when the product had a smaller scale, generated a limited amount of log data, and had a smaller number of issues to address. However, with the deployment of the product and the increased customer number, the requirements for log analysis become more demanding (as described in Section 2.2).

To meet these practical requirements, we have adopted SeaLog for monitoring Product X. Currently, 135 microservices within Product X have been equipped with SeaLog over a period of *twelve months*. The deployment of SeaLog in Product X is illustrated in Figure 9a. The detection agent's time and space efficiency allows for local deployment alongside each microservice. Once a suspicious anomaly is detected, the agent forwards it to a centrally deployed backbone analyzer for detailed analysis. For confidentiality reasons, during the online deployment, we utilize an internal LLM maintained by Huawei Cloud. Finally, if the LLM agrees that these anomalies are true, these anomalies together with explanations are raised as alerts to a central alert panel, enabling on-site engineers to take prompt mitigation measures. After the on-site engineers confirm the reported anomalies, these anomalies serve as valuable feedback to continuously improve the overall framework. During the past twelve months, SeaLog has resulted in less than 10% false positives, and the interpretable results have been well recognized by the on-site engineers for convenient confirmation.

To assess the performance of SeaLog in deployment, we gathered log data generated by these microservices over a five-week period (August 8, 2023 to September 12, 2023). Then, we compared the prediction results of SeaLog with manually annotated ground truth provided by on-site engineers. The results are shown in Figure 9b. In the first week, we solely deployed the detection agent, which resulted in a high recall but relatively low precision. This outcome was anticipated as we aimed to set a lenient threshold to capture most anomalies, regardless of the cost of increased false positive predictions. Starting from the second week, we deployed the feedback mechanism and the backbone analyzer. After that, the overall performance of SeaLog improved, indicating the effectiveness of the feedback loop and the backbone analyzer.

Ideally, LLMs would provide direct feedback to the local agent and avoid the need for human intervention. However, even with advanced LLMs like ChatGPT, correct predictions cannot always be guaranteed. Therefore, we still involve on-site engineers for confirmation in our real-world deployment. However, with the assistance of the LLM-based backbone analyzer, the manual effort required for confirmation has been significantly reduced. This is particularly beneficial for newly employed engineers who may not be familiar with the system yet. Looking

ahead, we believe that the proposed SeaLog framework can benefit from advancements in LLMs to automate the operation process of large-scale cloud systems.

4.9 Practical Deployment Considerations

Based on our multi-LLM evaluation and deployment experience, we discuss several practical considerations for deploying SeaLog.

Cost efficiency. The detection agent filters the majority of normal logs, resulting in only 314 backbone calls out of 721 test windows on BGL. Based on official API pricing (February 2026), the total cost per evaluation ranges from \$0.15 (GPT-4o-mini) to \$2.61 (GPT-4o), while DeepSeek-V3 costs only \$0.30. This demonstrates that SeaLog’s agent-backbone architecture effectively controls LLM costs.

Privacy preservation. The detection agent runs entirely locally, processing all log data without transmitting raw logs externally. Only the suspicious windows (a small subset) are forwarded to the backbone analyzer, which can be deployed on-premise for privacy-sensitive environments.

Nondeterminism. Our experiments show that LLM backbone nondeterminism is minimal at temperature=0, with an F1 standard deviation of only 0.003 across three runs. Only 2 out of 721 windows received different predictions, indicating that the framework produces highly reproducible results.

Choice of backbone. Our multi-LLM evaluation reveals that open-source models are competitive with commercial LLMs. Notably, Qwen3-235B-A22B achieves $F1=0.740$ on BGL, outperforming all commercial GPT variants ($F1=0.598-0.619$). This demonstrates that open-source backbones can be deployed on-premise without sacrificing accuracy. Meanwhile, the TrainedBackbone using domain-specific Naive Bayes achieves significantly higher F1 (0.890), suggesting that a lightweight trained backbone can outperform general-purpose LLMs when sufficient labeled data is available. We recommend the TrainedBackbone for air-gapped or cost-sensitive deployments. LLM backbones are better suited for environments where log patterns evolve frequently and labeled data is scarce.

Learning paradigm. The detection agent operates in a fully unsupervised manner, relying on incremental n-gram statistics that require no labeled data. The backbone analyzer with ICL is weakly supervised, as it retrieves labeled examples for in-context learning. When the feedback loop is enabled, confirmed anomalies from engineers are incorporated into the training set, making the overall framework semi-supervised. This flexibility allows SeaLog to start with zero labels and progressively incorporate expert feedback over time.

4.10 Threats to Validity

We identify the following threats to the validity of our study.

The cloud system under study. Our motivation for a lightweight and adaptive method is driven by real-world scenarios in a single cloud system. However, Huawei Cloud is a representative world-leading cloud provider with a vast scale. Besides, our findings in Section 2.2 are consistent with previous studies reported by other cloud vendors *e.g.*, Azure [51]. Additionally, we also adopt public datasets for evaluation. Hence, the evaluation results should be compelling.

The LLMs under study. We use an LLM as a backbone analyzer in the SeaLog framework. It is important to note that the performance and characteristics of LLMs can differ between implementations. We evaluate six LLMs spanning commercial (GPT-3.5-Turbo, GPT-4o-mini, GPT-4o, GPT-4.1-mini) and open-source (DeepSeek-V3, Qwen3-235B-A22B) models to ensure broad coverage. SeaLog is designed as a general framework, allowing LLMs to be replaced as necessary, thus enabling SeaLog to benefit from advancements in LLM technology.

Implementation and parameter settings. We have implemented several measures to mitigate these threats. First, we have employed peer code review while implementing SeaLog, and for the baseline methods, we have utilized open-sourced code released by the original authors or highly rated replications on GitHub. Second, we

have used the hyperparameters recommended by the original authors wherever available. Third, to ensure the reproducibility of our results, we have made our code and partial data publicly available.

5 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 [59].

Machine learning-based methods include the use of principal component analysis (PCA) by Xu et al. [56] for mining system problems from console logs and the work of Lou et al. [39], who detect system anomalies by mining invariants among log messages. Lin et al. [35] proposed LogCluster, which recommends representative log sequences for problem identification by clustering similar log sequences. He et al. [16] proposed Log3C, which incorporates system monitoring metrics into the identification of high-impact issues in service systems. Loglizer [17] provides a comprehensive evaluation of using ML-based methods for log-based anomaly detection. Fu et al. [12] detect execution anomalies by learning a finite state automaton from normal training log templates.

The ML-based methods typically require fewer computational resources for execution; however, they are not tailored to handle evolving logs in complex cloud systems. In contrast, SeaLog demonstrates superior efficiency due to our lightweight design and is capable of adapting to ever-changing logs.

Deep learning-based methods for log-based anomaly detection have been approached through various methods [26]. One such method, proposed by Du et al. [9], is DeepLog, which utilizes a Long Short-Term Memory (LSTM) network to model log sequences. LogAnomaly [43] 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 [28]. Supervised solutions leverage labeled anomalous logs to train a neural network for anomaly detection. Typical models include CNN-based approaches [40], GRU-based approaches [62], graph-based approaches [21, 45], attention-based approaches [20, 46] and Transformer-based approaches [28, 29].

Since labeled data is usually insufficient due to the labor-intensive nature of manual labeling, researchers have explored various methods to mitigate the impacts of limited labels. PLELog [58] is a semi-supervised method that learns probabilistic label estimation from normal and abnormal logs. AcLog [10] leverages deep active learning by incorporating human knowledge to augment the quality of training logs. HiLog [22] integrates human knowledge to augment anomaly detection models. MetaLog [61] incorporates the abundant historical labeled logs of mature systems to improve new systems' anomaly detection with a meta-learning paradigm.

DL-based methods utilize different neural network structures (e.g., 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, SeaLog utilizes an NPT for local and efficient anomaly detection, which is continuously improved by a remote expert.

LLM-lightweight model collaboration. Recent studies have explored the collaboration between LLMs and lightweight models for log-based anomaly detection, a paradigm that SeaLog also follows. AdaptiveLog [42] employs a fine-tuned BERT as the small language model and ChatGPT as the LLM, routing uncertain samples from BERT to ChatGPT for re-evaluation. This adaptive selection reduces LLM costs by approximately 73% while achieving strong performance (F1=0.989 on BGL). CoLA [67] uses a trained small detection model (SDM) as a filter to select potentially anomalous instances, which are then analyzed by an LLM expert. The LLM's decisions are fed back to refine the SDM in a continuous learning cycle. Compared with these approaches, SeaLog distinguishes itself in two key aspects: (1) *Zero-training detection agent*: Unlike AdaptiveLog's BERT fine-tuning or CoLA's SDM training, SeaLog's NPT-based detection agent requires no gradient-based training and learns incrementally from confirmed labels, enabling truly streaming deployment. (2) *Lightweight design*: The detection agent uses

$O(1)$ per-log scoring with LRU-bounded memory, making it suitable for resource-constrained environments where GPU-dependent models cannot be deployed. All three frameworks share the “small model screens, LLM confirms” paradigm, validating the architectural choice of SeaLog while highlighting the unique advantages of a training-free approach.

6 Conclusion

In this paper, we presented SeaLog, a lightweight and adaptive log-based anomaly detection framework designed for practical cloud systems. SeaLog utilizes a detection agent for lightweight anomaly detection in a streaming manner. We also incorporate large language models (LLMs) as a backbone analyzer to provide more accurate and adaptive analysis. Experimental results on three public datasets (BGL, Thunderbird, and HDFS) and an industrial dataset from Huawei Cloud showed that SeaLog is effective, achieving F1 scores between 0.949 and 0.999, attaining the highest F1 on all four benchmarks. Our multi-LLM evaluation, including open-source models that enable fully on-premise deployment, demonstrates that SeaLog is robust to the choice of LLM backbone, and a domain-specific trained backbone can serve as a cost-effective alternative. Moreover, SeaLog maintained high and consistent accuracy, even in the presence of evolving log data *i.e.*, adaptive. Additionally, SeaLog is $2\times$ to $5\times$ faster and requires only 5% to 41% memory resources compared to existing methods *i.e.*, lightweight. With the advantages of SeaLog, we have deployed this framework in our production environment for twelve months. Our analysis shows that SeaLog can continuously maintain a high performance (~ 0.9 F1 score) over time. To benefit the community, we share our experience in deploying SeaLog in Huawei Cloud and make SeaLog [37] publicly available.

Acknowledgments

The work described in this paper was supported by two grants from the Research Grants Council of the Hong Kong Special Administrative Region, China: (1) No. SRFS2425-4S03 of the Senior Research Fellow Scheme, and (2) No. CUHK 14209124 of the General Research Fund.

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] George EP Box and George C Tiao. 1973. *Bayesian inference in statistical analysis*. Addison-Wesley.
- [3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.
- [4] 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.
- [5] 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.
- [6] 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.
- [7] 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.
- [8] 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).
- [9] 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.

- [10] Chiming Duan, Tong Jia, Ying Li, and Gang Huang. 2023. AcLog: An Approach to Detecting Anomalies from System Logs with Active Learning. In *2023 IEEE International Conference on Web Services (ICWS)*. 436–443.
- [11] Robert M French. 1999. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences* 3, 4 (1999), 128–135.
- [12] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. 2009. Execution anomaly detection in distributed systems through unstructured log analysis. In *2009 ninth IEEE international conference on data mining*. IEEE, 149–158.
- [13] Shuzheng Gao, Xin-Cheng Wen, Cuiyun Gao, Wenxuan Wang, Hongyu Zhang, and Michael R. Lyu. 2023. What Makes Good In-context Demonstrations for Code Intelligence Tasks with LLMs?. In *Proceedings of the 38th International Conference on Automated Software Engineering (ASE)*. 761–773.
- [14] 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.
- [15] 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.
- [16] 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.
- [17] 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.
- [18] Junjie Huang, Zhihan Jiang, Zhuangbin Chen, and Michael R Lyu. 2025. No More Labelled Examples? An Unsupervised Log Parser with LLMs. In *Proceedings of the ACM International Conference on the Foundations of Software Engineering (FSE)*.
- [19] Yintong Huo, Cheryl Lee, Yuxin Su, Shiwen Shan, Jinyang Liu, and Michael R Lyu. 2023. EvLog: Identifying Anomalous Logs over Software Evolution. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 391–402.
- [20] Peng Jia, Shaofeng Cai, Beng Chin Ooi, Pinghui Wang, and Yiyuan Xiong. 2023. Robust and Transferable Log-based Anomaly Detection. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 64:1–64:26.
- [21] Tong Jia, Pengfei Chen, Lin Yang, Ying Li, Fanjing Meng, and Jingmin Xu. 2017. An approach for anomaly diagnosis based on hybrid graph model with logs for distributed services. In *2017 IEEE international conference on web services (ICWS)*. IEEE, 25–32.
- [22] Tong Jia, Ying Li, Yong Yang, Gang Huang, and Zhonghai Wu. 2022. 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*. 3081–3089.
- [23] Zhihan Jiang, Jinyang Liu, Zhuangbin Chen, Yichen Li, Junjie Huang, Yintong Huo, Pinjia He, Jiazhen Gu, and Michael R Lyu. 2024. LILAC: Log Parsing using LLMs with Adaptive Parsing Cache. In *Proceedings of the ACM International Conference on the Foundations of Software Engineering (FSE)*.
- [24] Zhihan Jiang, Jinyang Liu, Junjie Huang, Yichen Li, Yintong Huo, Jiazhen Gu, Zhuangbin Chen, Jieming Zhu, and Michael R Lyu. 2024. A Large-Scale Evaluation for Log Parsing Techniques: How Far Are We?. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*.
- [25] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data* 7, 3 (2021), 535–547.
- [26] Max Landauer, Sebastian Onder, Florian Skopik, and Markus Wurzenberger. 2023. Deep learning for anomaly detection in log data: A survey. *Machine Learning with Applications* 12 (2023), 100470.
- [27] 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.
- [28] 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.
- [29] Van-Hoang Le and Hongyu Zhang. 2024. PreLog: A Pre-trained Model for Log Analytics. *Proc. ACM Manag. Data* 2, 3 (2024), 163.
- [30] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, and Michael R Lyu. 2023. Eadro: An end-to-end troubleshooting framework for microservices on multi-source data. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1750–1762.
- [31] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, Yongqiang Yang, and Michael R Lyu. 2023. Heterogeneous anomaly detection for software systems via semi-supervised cross-modal attention. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1724–1736.
- [32] James Lewis and Martin Fowler. 2014. Microservices a definition of this new architectural term. <http://martinfowler.com/articles/microservices.html>
- [33] Yichen Li, Yintong Huo, Zhihan Jiang, Renyi Zhong, Pinjia He, Yuxin Su, Lionel Briand, and Michael R Lyu. 2023. Exploring the effectiveness of llms in automated logging generation: An empirical study. *arXiv preprint arXiv:2307.05950* (2023).
- [34] 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.

- [35] 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.
- [36] 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.
- [37] Jinyang Liu et al. 2026. Replication Package of SeaLog. <https://github.com/JinYang88/SeaLog>
- [38] 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.
- [39] 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.
- [40] 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.
- [41] Shutian Luo, Huanle Xu, Chengzhi Lu, Kejiang Ye, Guoyao Xu, Liping Zhang, Yu Ding, Jian He, and Chengzhong Xu. 2021. Characterizing microservice dependency and performance: Alibaba trace analysis. In *Proceedings of the ACM Symposium on Cloud Computing*. 412–426.
- [42] Lipeng Ma, Weidong Yang, Yixuan Li, Ben Fei, Mingjie Zhou, Shuhao Li, Sihang Jiang, Bo Xu, and Yanghua Xiao. 2025. AdaptiveLog: An Adaptive Log Analysis Framework with the Collaboration of Large and Small Language Model. *arXiv preprint arXiv:2501.11031* (2025).
- [43] 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 *IJCAL*. 4739–4745.
- [44] 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.
- [45] Animesh Nandi, Atri Mandal, Shubham Atreja, Gargi B. Dasgupta, and Subhrajit Bhattacharya. 2016. Anomaly Detection Using Program Control Flow Graph Mining From Execution Logs. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 215–224.
- [46] Sasho Nedelkoski, Jasmin Bogatinovski, Alexander Acker, Jorge Cardoso, and Odej Kao. 2020. Self-attentive classification-based anomaly detection in unstructured logs. In *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1196–1201.
- [47] 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.
- [48] OpenAI. [n. d.]. OpenAI Embeddings. <https://platform.openai.com/docs/guides/embeddings>
- [49] OpenAI. 2022. Introducing ChatGPT. <https://openai.com/blog/chatgpt>
- [50] Fabian Pedregosa and Philippe Gervais. 2022. Memory Profiler. https://github.com/pythonprofilers/memory_profiler
- [51] 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.
- [52] 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.
- [53] Wikipedia. [n. d.]. Lidstone Smoothing. https://en.wikipedia.org/wiki/Additive_smoothing
- [54] Junjielong Xu, Ruichun Yang, Yintong Huo, Chengyu Zhang, and Pinjia He. 2024. DivLog: Log Parsing with Prompt Enhanced In-Context Learning. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (ICSE)*.
- [55] Wei Xu. 2010. *System problem detection by mining console logs*. University of California, Berkeley.
- [56] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. 2009. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP)*. 117–132.
- [57] 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.
- [58] 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.
- [59] Boxi Yu, Jiayi Yao, Qiulai Fu, Zhiqing Zhong, Haotian Xie, Yaoliang Wu, Yuchi Ma, and Pinjia He. 2024. Deep Learning or Classical Machine Learning? An Empirical Study on Log-Based Anomaly Detection. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 403–415.
- [60] Guangba Yu, Pengfei Chen, Yufeng Li, Hongyang Chen, Xiaoyun Li, and Zibin Zheng. 2023. Nezha: Interpretable Fine-Grained Root Causes Analysis for Microservices on Multi-modal Observability Data. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 553–565.

- [61] Chenyangguang Zhang, Tong Jia, Guopeng Shen, Pinyan Zhu, and Ying Li. 2024. MetaLog: Generalizable Cross-System Anomaly Detection from Logs with Meta-Learning. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) (ICSE '24). Article 154, 12 pages.
- [62] 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.
- [63] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223* (2023).
- [64] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Wenhai Li, and Dan Ding. 2021. Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study. *IEEE Transactions on Software Engineering* 47, 2 (2021), 243–260.
- [65] Jieming Zhu, Shilin He, Pinjia He, Jinyang Liu, and Michael R Lyu. 2023. Loghub: A Large Collection of System Log Datasets for AI-driven Log Analytics. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 355–366.
- [66] 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.
- [67] Xuhang Zhu, Xiu Tang, Sai Wu, Jichen Li, Haobo Wang, Chang Yao, Quanqing Xu, and Gang Chen. 2025. CoLA: Model Collaboration for Log-based Anomaly Detection. *Proceedings of the VLDB Endowment* 18, 11 (2025), 3979–3987.