Automated Software Engineering (2026) 33:12
https://doi.org/10.1007/s10515-025-00548-y

Improving anomaly detection in software logs through hybrid language modeling and reduced reliance on parser

Yicheng Sun[1] · Jacky Keung[1] · Zhen Yang[2] · Shuo Liu[1] · Hi Kuen Yu[1]

Received: 18 January 2025 / Accepted: 30 July 2025 / Published online: 29 September 2025

Abstract

Anomaly detection in software logs is crucial for development and maintenance, allowing timely identification of system failures and ensuring normal operations. Although recent deep learning advancements in log anomaly detection have shown exceptional performance, the reliance on time-consuming log parsers raises concerns about their necessity for quickly identifying anomalies. Standardized preprocessing methods can mishandle or lose important information. Additionally, the significant imbalance between normal and anomalous log data, along with the scarcity of labeled data, presents a persistent challenge in anomaly detection. We first evaluated the impact of omitting a log parser on anomaly detection models. Subsequently, we propose LogRoBERTa, an innovative anomaly detection model that eliminates the need for a parser. LogRoBERTa creates a stable and diverse labeled training set using the Determinantal Point Process (DPP) method, needing only a small amount of labeled data. The hybrid language model is based on RoBERTa's architecture, combined with an attention-based BiLSTM. This setup leverages RoBERTa's strong contextual understanding and BiLSTM's capability to capture sequential dependencies, enhancing performance in complex log sequences. Experiments on four widely used datasets demonstrate that LogRoBERTa outperforms state-of-the-art benchmark models—including three fully supervised approaches—without relying on a dedicated log parser. Furthermore, its consistently strong performance on low-resource datasets highlights its robustness and generalizability across varying data conditions. These results validate the overall effectiveness of LogRoBERTa's design and offer a thorough evaluation of the implications of bypassing a log parser. Additionally, our ablation studies and training set construction experiments further confirm the contributions of each individual component to the model's performance. The study empirically validated that a RoBERTa-based approach effectively handles software log anomaly detection in long and complex log sequences, providing a more efficient and robust solution for omitting a parser compared to existing models.

Extended author information available on the last page of the article

Keywords Anomaly detection · Empirical software engineering · Software log analysis · Log parsing · Hybrid language model · Large language model

1 Introduction

Detecting anomalies in complex software architectures is challenging due to multiple interconnected systems, each relying on interdependence for efficient data processing (Nyyssälä et al. 2022). The intrinsic complexity and large user base of these systems lead to inevitable challenges (Mi et al. 2013; Hashemi and Mäntylä 2024). Therefore, implementing log generation within this intricate network is crucial for effective troubleshooting (Mi et al. 2013). Real-time operational log recording is essential to ensure reliability and high availability (Nyyssölä and Mäntylä 2023), enabling developers to assess software stability, track system status, and monitor significant events (Yu et al. 2023).

Parsing log messages into log events, which represent the constant components of message templates, has been widely studied (Le and Zhang 2022). Recent research efforts (Zhu et al. 2015; He et al. 2016; Chen et al. 2021; Wu et al. 2023; Le and Zhang 2023; Zhang et al. 2023) have focused on leveraging deep learning models for anomaly detection methods based on software logs. The objective is to detect abnormal behavior by analyzing operational logs and taking prompt actions (Zhu et al. 2019). Despite the outstanding performance demonstrated by previous anomaly log detection models on public datasets (Nedelkoski et al. 2020; Zhang et al. 2019; Li et al. 2023), the current research faces various limitations and challenges:

Log parser limitations and deep learning advancements Most log preprocessing methods rely on predefined templates to standardize log keys, which risks losing crucial information during the standardization process and often results in excessive dependence on the compatibility of log parsers (Xie et al. 2023). Additionally, log parsers themselves can be time-consuming, as they often require substantial processing time to handle large-scale log datasets. Previous studies have demonstrated that 9 out of 15 log parsers fail to process all 15 datasets from Loghub-2.0 within a reasonable 12-hour timeframe (Jiang et al. 2024). Furthermore, log parsing based on large language models, such as ChatGPT, requires even more time. For instance, in the Mac dataset, the number of logs per template ranges from 1 to 166, with over 50% of templates containing only a single log (Xu et al. 2024). Additionally, more than 45% of logs contain only 1-2 variables, while logs with more than 6 variables represent less than 3%. With the use of log parsers, recent advances in deep learning models have achieved outstanding performance, with F1-scores exceeding 0.99 (Yu et al. 2024). However, the primary objective of log anomaly detection remains the quick identification of anomalous logs. This raises the question of whether spending extensive time removing variables from log sequences is necessary, given the current state of log anomaly detection techniques.

Excessive data imbalance and reliance on labeled data Interconnected systems typically operate concurrently, generating a vast and dense volume of logs. In software

development, the majority of these logs are normal, with only a small fraction being anomalous. Figure 1 illustrates the log occurrence frequency for two common datasets from the LogHub platform, based on a small sample from a specific time period. In the HDFS dataset (Fig. 1a), logs generated by 203 nodes over a 12-hour period reached a peak frequency of 186,836 entries per minute. In the BGL dataset (Fig. 1b), logs from a single node were collected over three days, with a Maximum frequency of 153,769 entries per hour. However, the proportion of anomalous logs in the small samples of these two datasets is only 2.73% and 6.84%, respectively. Therefore, the key challenge lies in quickly learning the characteristics of anomalous logs for accurate detection. While current research (Sun et al. 2025b; Guo et al. 2021; Lin and Chiang 2022) often uses large amounts of labeled logs to capture detailed features, this

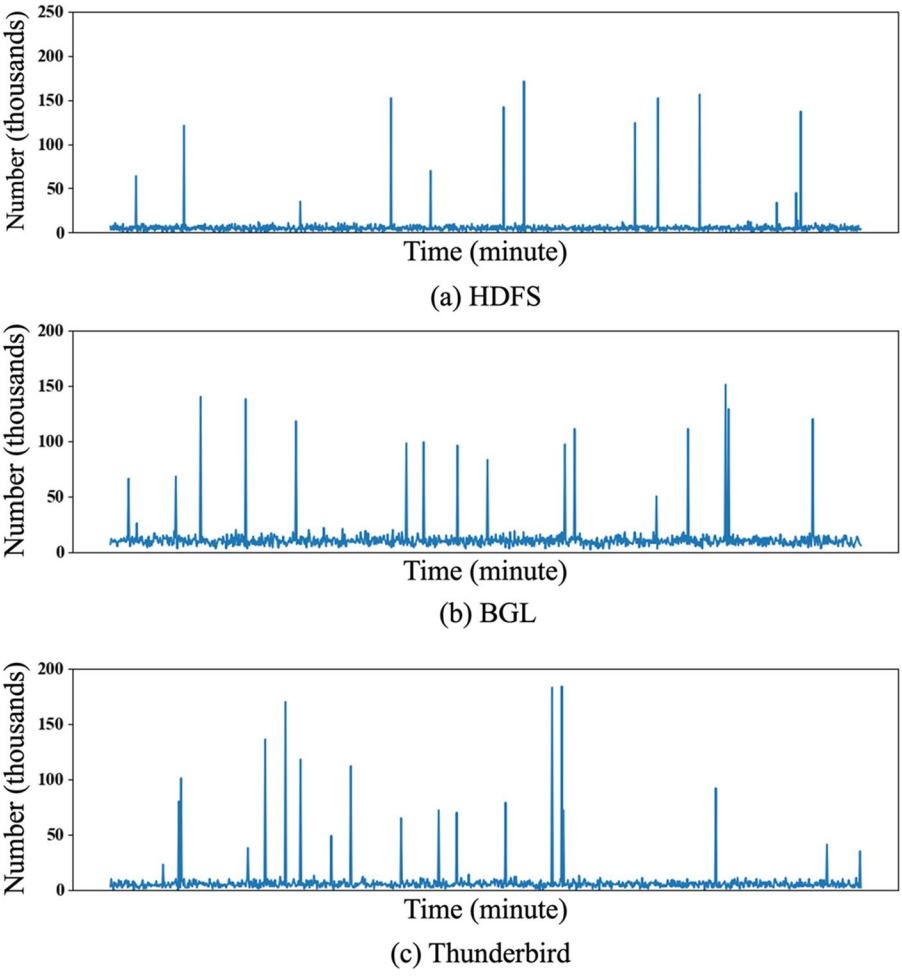


Fig. 1 The log occurrence frequency of three common datasets. **Note:** Due to the varying scales and time spans of the different datasets, we employed different time intervals for analysis. The HDFS dataset was analyzed using minute-long intervals, while the BGL and Thunderbird datasets were analyzed using hour-long intervals

approach has two major limitations: Firstly, labeling logs requires significant manual effort from engineers, making it challenging to obtain sufficient labeled logs in real-world scenarios. Secondly, due to the sheer volume of logs, even though existing anomaly detection models often focus on training with only normal logs or a subset of logs, the total amount of logs used remains large. This still results in substantial time consumption during both labeling and training processes, given the vast log base.

To explore and address these challenges, we evaluated several representative log anomaly detection models, comparing their performance with and without the use of log parsers. In addition, we measured the time saved when log parsers were bypassed. This analysis allowed us to assess how effectively anomaly detection models operate independently of predefined log templates, providing insights into both detection accuracy and efficiency improvements. Additionally, to address the reliance on parsers and the extensive need for labeled logs, we propose **LogRoBERTa**, an innovative framework combining a hybrid language model for **log** anomaly detection based on the **R**obustly **O**ptimized **BERT** **A**pproach. We extracted the core log sequences without extensive parsing to reduce execution time. To support the training of the hybrid language model, we constructed a small yet stable labeled training set using the Determinantal Point Process (DPP) method.

The hybrid language model of amalgamates the characteristics of RoBERTa and Bi-LSTM. Initially, we harness the RoBERTa model to capture semantic features within software log text. RoBERTa (Liu et al. 2019), in contrast to BERT (Devlin et al. 2018), enhances the ability to capture crucial log information through optimized pre-training methods and an improved dynamic masking mechanism. Subsequently, these features are input into a Bidirectional Long Short-Term Memory network (Bi-LSTM) (Huang et al. 2015). Recognizing that Bi-LSTM may not adequately focus on key information in the log context when processing software log text, we introduce an attention mechanism layer before the Bi-LSTM output. The attention mechanism highlights important information within the log text by assigning varying weights to different parts of the context. We compare our model against existing methods using four widely used public log benchmark datasets: HDFS, BGL, Thunderbird, and Spirit. Our model consistently demonstrates outstanding performance, offering valuable assistance to the field of anomaly log detection.

In summary, our contributions are as follows:

- *Introduction of the innovative LogRoBERTa model.* We treat software logs as natural language sequences, employing a pre-trained language model to learn semantic representations of both normal and anomalous logs. By combining the characteristics of RoBERTa and Bi-LSTM and integrating an attention mechanism, our model demonstrates outstanding performance in anomaly log detection.
- *Evaluating the Necessity of Log Parsing.* We are the first to systematically evaluate the necessity of log parsing in log anomaly detection. We investigate the impact of both parsed and unparsed log sequences on the performance of state-of-the-art deep learning models. By examining whether log parsing is essential for achieving high anomaly detection accuracy, we provide new insights into the

trade-off between preprocessing time and model performance.
- *Comprehensive evaluation.* We conducted comprehensive experiments to validate the performance of LogRoBERTa, including a comparative assessment, an ablation study, and evaluations across different scales.

Overall, our contributions signify advancements in anomaly log detection methodologies, offering robust and adaptable solutions to address prevalent challenges in the field.

The remainder of this paper is organized as follows. Section 2 presents the background and related work of anomaly detection. Section 3 elaborates on the proposed approach, including model initialization, log parsing using ChatGPT, pseudo-label generation, log representation, and anomaly detection model building. Sections 4 and 5 discuss the experimental design and results. Section 6 addresses the threats to validity. Finally, Section 7 summarizes the paper.

2 Background and related work

2.1 Log collection

Software logs offer engineers valuable insights into the current system status and the root causes of failures. Consequently, collecting and analyzing logs become essential tasks for both development and operations personnel. However, due to differences in the definition of logs given by different developers, logs often exhibit significant diversity (Tufek and Aktas 2023). Thus, establishing public datasets becomes necessary for evaluating the performance of models developed by various researchers.

The Loghub (Zhu et al. 2023) platform has made a substantial amount of system logs available to researchers, serving as the primary data source. For instance, the HDFS (Xu et al. 2009) log file was collected from over 200 Amazon EC2 nodes, while the BGL (Oliner and Stearley 2007) log file was gathered from the Blue Gene/L supercomputer system. Both datasets originate from production systems, comprising a total of 15,923,592 log messages and 365,298 abnormal samples, with anomalies manually labeled by domain experts.

2.2 Log parsing

Logs, typically generated through log statements in the source code such as logging.info(), printf(), and Console.WriteLine() (Dai et al. 2020), represent unstructured textual data. Each log entry records specific system events, potentially including fields like timestamps, severity levels, and message content (Makanju et al. 2009). The most common log parsing method is heuristic-based (Zhang et al. 2019), extracting log templates according to heuristic rules. Drain (He et al. 2017), utilizing a fixed-depth parsing tree to encode specially designed parsing rules, is currently one of the most commonly used parsers in log processing (Huo et al. 2023). Log parsing is employed to automatically convert each log message into a structured format by identifying an event template (constant part) associated with parameters (dynamic

variable parts) (Zhu et al. 2019; Le and Zhang 2023). For example, in a software log message like "*2023-11-17 00:04:09.377 INFO: 34.139.166.52:0 - 'POST /webhook/order HTTP/1.0' 200 OK delete empty directory: /backend/upload/download/orders/49512/49512*", the timestamp is "*2023-11-17 00:04:09.377*", the severity level is "*INFO: 34.139.166.52:0 - 'POST/webhook/order HTTP/1.0' 200 OK*", and the message content is "*delete empty directory: /backend/upload/download/orders/49512/ 49512*". After appropriate log parsing, this log message can be split into the event template "*'POST<*>' 200 OK delete empty directory:<*>*", and the parameter values [*/webhook/order HTTP/1.0*, */backend/upload/download/orders/49512/49512*]. Here, "<*>" denotes the position of a parameter.

Despite the widespread use of log parsing for structuring log data, most log preprocessing methods heavily depend on predefined templates, risking information loss and over-reliance on parser compatibility (Xie et al. 2023). Log parsers are also time-consuming, especially with large datasets, and parsing using large language models adds even more time. Despite recent deep learning models achieving high F1-scores, the core objective remains rapid anomaly detection. This raises the question: Is extensive log parsing necessary, given modern detection techniques?

2.2.1 The role of log parsers in historical context

Traditional log parsers can be broadly classified into two categories: rule-based parsers and parsers leveraging large language models (LLMs), such as ChatGPT (Le and Zhang 2023). Rule-based parsers, like Drain (He et al. 2017), rely on predefined templates to extract log structures, whereas LLM-based parsers (Ma et al. 2024; Le and Zhang 2023) attempt to comprehend and analyze logs with greater flexibility. Historically, log parsing was essential due to the limitations of early machine learning models, which often struggled to process unstructured log data. By converting logs into structured templates, these models were better able to identify anomalies. Figure 2 illustrates the performance of the two most commonly used types of parsers. The rule-based parser, specifically Drain, is currently the most widely used. It can be observed that dynamic variables, such as file addresses, IPs, numerical values, timestamps, and usernames, constitute a relatively small proportion of log sequences. Rule-based parsers often misinterpret these dynamic variables. Although LLM-based parsers demonstrate improved parsing accuracy, their processing speed remains significantly slower compared to rule-based parsers.

Moreover, existing research has confirmed that, in most datasets, individual log sequences typically contain a limited number of variables. For example, in the Mac dataset, over 45% of logs contain only 1-2 variables, while logs with more than 6 variables account for less than 3% (Xu et al. 2024). This pattern is also observed in the HDFS and BGL datasets. Such uniformity implies that the majority of log entries exhibit little variation. Contemporary deep learning models, especially hybrid language models, have achieved exceptional performance, with F1-scores exceeding 0.99. The advanced capabilities of models like the Hybrid Language Model further reduce the necessity of log parsers, as they can now detect anomalies directly from raw logs with high precision. Therefore, we posit that log parsers are becoming less essential for log analysis, given the advancements in modern anomaly detection mod-

Original Log Sequence	LLM-based parsers	Rule-based parsers
generating core.385	generating core.<*>	generating core.<*>
ciod: Error creating node map from file /home/pakin1/sweep3d-2.2b/results/random1-8x32x32x2.map: Permission denied	ciod: Error creating node map from file <*>: Permission denied	ciod: Error creating node map from file /home/pakin<*>/sweep<*>d-<*>.<*>b/results/random<*>x<*>.map: Permission denied
HTTP exception thrown: No instances found for any event	HTTP exception thrown: No instances found for any event	HTTP exception thrown: No instances found for any event
storage.BlockManager: Found block rdd_2_4 locally	storage.BlockManager: Found block rdd_2_4 locally storage.BlockManager: Found block <*> locally	storage.BlockManager: Found block rdd_2_4 locally
mod_jk child workerEnv in error state 6	mod_jk child workerEnv in error state 6	mod_jk child workerEnv in error state 6
session opened for user root by (uid=0)	session opened for user <*> by (uid=<*>)	session opened for user root by (uid=0)
ciod: failed to read message prefix on control stream (CioStream socket to 172.16.96.116:33569	ciod: failed to read message prefix on control stream (CioStream socket to <*>	ciod: failed to read message prefix on control stream (CioStream socket to <*>:<*>
Adding protocol org.apache.hadoop.mapreduce.v2.api.MRClientProtocolPB to the server	Adding protocol org.apache.hadoop.<*> to the server Adding protocol <*> to the server	Adding protocol org.apache.hadoop.mapreduce.v2.api. MRClientProtocolPB to the server
authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=61.53.154.93 user=root	authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=<*> user=<*>	authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=<*> user=root

Fig. 2 The performance of the two most commonly used types of parsers. **Note:** Dynamic variables are highlighted in **red**. It can be observed that some logs do not contain dynamic variables, while logs with 1-2 dynamic variables constitute the majority. The rule-based parser (e.g., Drain) often misses or incorrectly interprets many variables. In contrast, the LLM-based parser (e.g., ChatGPT) shows significantly improved accuracy but can only process an average of 2 logs per second, requiring considerable time for large-scale log datasets

els. However, there is currently no systematic study exploring the impact of using or not using a log parser on the performance of anomaly detection models.

2.2.2 The time consumption of log parsers

A significant downside of log parsers is the time they require for processing large-scale datasets. Previous studies have demonstrated that 9 out of 15 log parsers were unable to process all 15 datasets from Loghub-2.0 within a reasonable 12-hour timeframe (Jiang et al. 2024). This extensive processing time becomes particularly problematic in log anomaly detection, where the goal is to rapidly identify abnormal events. In such scenarios, time efficiency is critical for timely response and mitigation (Jiang et al. 2024). While log parsers can provide structured data, their slow speed undermines their utility in real-time anomaly detection. As anomaly detection continues to prioritize both accuracy and speed, the heavy time consumption of log parsers poses a challenge, suggesting that alternative approaches which bypass log parsing may be more appropriate for contemporary needs.

2.3 Log representation

Log parsing generates log event templates, which are then transformed into vectors for log representation (Le and Zhang 2022). Log data must be represented in a format suitable for deep learning models such as LSTM, Bi-LSTM, and Transformer, which can extract semantic information from log data.

Du et al. (2017) proposed a method called DeepLog, utilizing LSTM, which indexes each log event and generates sequential vectors for each log window, modeling system logs as natural language sequences (Hu et al. 2023). LogAnomaly (Meng et al. 2019) employs sequential and quantitative vectors for anomaly detection. LogRobust (Zhang et al. 2019) adopts a pre-trained FastText model to compute semantic vectors for log events and utilizes an Attention-based Bi-LSTM model for anomaly detection. Huang et al. employ the HitAnomaly method to model log template sequences and parameter values, finally using an attention mechanism as a classification model. LogBERT (Guo et al. 2021) utilizes semantic vectors to learn patterns in normal log sequences through masked log message prediction and hypersphere volume minimization.

2.4 Representative anomaly detection models

In the context of this study, an anomaly is defined as a log entry that reflects abnormal system behavior, operational failure, or unexpected conditions that deviate from normal execution flows. These anomalies may manifest in various forms, including explicit error messages (e.g., "File not found", "Permission denied"), critical system warnings, or semantic inconsistencies in domain-specific logs—such as invalid command sequences or mechanical misalignments. Unlike structural noise (e.g., logs arriving out of order), which may occur due to distributed system delays, anomalies in this work are characterized by their association with actual system faults, execution errors, or undesired runtime states that require further investigation or remediation.

As illustrated in Fig. 3, we present representative examples of anomalous logs collected from various modern software and industrial systems. For instance, log messages such as "*Empty file deleted unexpectedly*" or "*Sensor threshold exceeded: 92.7°C*" indicate abnormal system states and are labeled as anomalies. Furthermore, entries like "*Robotic arm deviation exceeds tolerance: +4.5mm*" or "*Component missing at inspection station*" are marked anomalous due to their implications of mechanical malfunction or production sequence violations. These examples demonstrate the diverse range of anomaly types considered in this study, spanning from conventional software-level errors to domain-specific operational failures in complex environments.

The challenge of anomaly detection in software logs is commonly regarded as a binary classification problem (Landauer et al. 2023). Currently, researchers have proposed various methods for detecting abnormal logs from large-scale log datasets. These algorithms can be categorized into unsupervised (Du et al. 2017; Zeufack et al. 2021) , and supervised (Lin and Chiang 2022; Yang et al. 2021; Ying et al. 2021; Zhang et al. 2019) learning approaches. Zhang et al. (2019) introduced LogRobust, which represents log events as semantic vectors and uses an attention-based

[System Error]: FileNotFoundError – File not found: /data/archive/report_2023.log
[Permission Issue]: PermissionError – Access denied for user 'service_account' on /etc/config.yaml
[Null Operation]: Warning – Attempted to delete empty directory: /cache/tmp/session_489
[Service Downtime]: ERROR – Service heartbeat lost for node-23 at 10:42:17
[Positional Deviation]: Robotic arm deviation exceeds tolerance: +4.5mm (max allowed: ±2.0mm)
[Data Synchronization Failure]: ERROR – Data synchronization timeout with remote sensor node-17
[Missing Component]: Inspection failure – Missing part detected at assembly station 3
[Checksum Mismatch]: Completed: ID=28, duration=2.3s, output=checksum mismatch at byte 94
[Memory Usage Spike]: Allocated memory: 1037MB (limit: 1024MB) → succeeded
[Sensor Fault]: Temperature threshold exceeded: 92.7°C in chamber A
[Expired Session Token]: HTTP 200 OK - Response: {"error": "token expired", "retry": false}
[Remote Override]: FeatureToggle: 'auto_backup' set to false via remote override
[Gripper Misalignment]: Gripper alignment delta exceeds threshold: $\Delta\theta$=0.012 rad
[Policy Violation]: Action ignored: policy mismatch between requester and target domain
[Task Preemption]: Task 492 preempted by Task 489 due to scheduler override
[Data Loss Warning]: Result: ok | warnings:1 | dropped_items:12
[Path Resolution Failure]: resolvePath: lookup failed → fallback to NULL

Fig. 3 Examples of anomalous log entries across multiple scenarios. **Note:** Anomalies include checksum mismatches, retry spikes, robotic misalignments, memory overuse, expired tokens, segmentation faults, and other system-level faults. These examples highlight the diversity and semantic complexity of real-world anomalies

Bi-LSTM model to detect anomalies. Huang et al. (2020) were the first to apply the transformer model to the task of log anomaly detection; they proposed HitAnomaly, which used a hierarchical transformer to model sequences of log templates and parameter values and designed an attention mechanism to merge semantic information with parameter values for classification, demonstrating robustness to volatile log data. Du et al. (2017) introduced the DeepLog framework, which treats log information as natural language sequences, autonomously learns log patterns from normal execution sequences, and identifies anomalies by assessing whether incoming log events deviate from the predictions of a stacked LSTM model. Meng et al. (2019) devised a template representation method based on synonyms and antonyms, termed template2vec. This method involves matching incoming log events with existing templates rather than directly flagging them as anomalies, thereby enhancing accuracy. Lin and Chiang (2022); Yang et al. (2021) introduced a semi-supervised learning framework known as PLELog, which relies on probabilistic label estimation. PLELog utilizes the HDBSCAN (McInnes and Healy 2017) clustering method for estimating labels probabilistically on unlabeled log sequences, addressing the challenge of limited labeling. NeuralLog (Le and Zhang 2021) introduces a novel log-based anomaly detection method based on a Transformer classification model that eliminates the need for log preprocessing.

BERT (Devlin et al. 2018), proposed by Google in 2018, stands as a groundbreaking pre-trained language model that has revolutionized natural language processing (NLP) and found widespread application in anomaly log detection (Clark et al. 2019). LogBERT (Guo et al. 2021) learns patterns within normal log sequences through masked log message prediction and hypersphere volume minimization, yet it cannot specifically identify semantic information in abnormal logs. LANoBERT (Lee et al. 2023) employs unsupervised learning using the BERT model to detect anom-

alous logs, although it does not fully consider interactive factors between normal and abnormal logs. BERT-log (Chen and Liao 2022) consists of an event template extractor, log semantic encoder, and log anomaly classifier, automatically detecting log anomalies based on the BERT pre-trained language model. RoBERTa (Liu et al. 2019) represents a significant advancement over the BERT model; however, its extensive exploration in the realm of log anomaly detection remains incomplete.

2.5 RoBERTa

BERT (Devlin et al. 2018) is a model consisting of Transformer encoders, distinguished by its foundation in unsupervised learning from large-scale textual data. BERT comprehensively understands the semantics and syntactic structure of language, showcasing exceptional performance across various NLP tasks (Tenney et al. 2019).

Key innovations of BERT include the introduction of three types of embeddings: Token Embedding, Segment Embedding, and Position Embedding. Token Embedding converts each word in the text into a vector representation in a high-dimensional space. Segment Embedding distinguishes different sentences in the text, aiding the model in understanding and analyzing relationships between sentences. Position Embedding provides positional information for each word in the text (Jawahar et al. 2019). BERT's pre-training employs two types of unsupervised learning methods: Masked Language Model (MLM) and Next Sentence Prediction (NSP). **MLM** randomly masks certain vocabulary from the input text and uses "[MASK]" to replace these words, while **NSP** determines whether two sentences are logically connected, evaluating coherence or interdependence between the sentences.

RoBERTa (Liu et al. 2019), a robustly optimized BERT pre-training method introduced by Facebook AI in 2019, represents a significant advancement over the BERT model. RoBERTa retains the multi-layer Transformer encoder structure of BERT; however, in this model, key parameters are adjusted, such as an increased number of layers, expanded size of hidden layers, and larger number of attention heads. These features enable RoBERTa to more effectively capture complex linguistic features and patterns.

In terms of pre-training, RoBERTa differs from BERT by relying solely on MLM as its pre-training task. Additionally, RoBERTa introduces Dynamic Masking mode, replicating training data and executing a series of masking strategies. During data processing, RoBERTa employs large-scale and diversified training datasets covering a wider range of text types and larger corpora, significantly enhancing the model's understanding of different text types and its generalization capabilities. Furthermore, RoBERTa employs longer sequence lengths and larger batch sizes during training, further enhancing training efficiency and model performance.

In the context of anomaly detection in software logs, RoBERTa excels at capturing contextual information in software logs and comprehending complex log structures, critical for accurately identifying key information within log sequences. We

believe that RoBERTa, with its superior optimization strategies compared to BERT, will undoubtedly exhibit superior performance in anomaly detection in software logs.

2.6 Bi-LSTM

The LSTM network (Graves and Graves 2012), a variant of RNNs, excels in tasks involving sequential data due to its ability to capture contextual information. It belongs to the family of artificial neural networks (Memory 2010). LSTM networks utilize LSTM units as building blocks in their hidden layers, showcasing effectiveness in capturing long-term dependencies. These units consist of three main components (Kulshrestha et al. 2020): the input gate i_t, the forget gate f_t, and the output gate o_t. The input gate regulates the retention of candidate stage data, determining which information is incorporated into the current internal state. The forget gate decides the degree to which information from previous time steps is forgotten, crucial for managing long sequences effectively. The output gate modulates the flow of information from the current internal state to the external state, controlling the output of the LSTM unit.

In traditional LSTM, information flows unidirectionally, moving solely from the beginning to the end of a sequence. However, when handling complex linguistic structures like log sequences, there exists interdependence between preceding and subsequent information, potentially resulting in the loss of crucial contextual information within log sequences.

Bi-LSTM (Huang et al. 2015) addresses this limitation by introducing a bidirectional structure. It comprises forward and backward LSTMs, responsible for processing the input sequence in both forward (from start to end) and backward (from end to start) directions of information flow (Yildirim 2018). This bidirectional processing strategy enables Bi-LSTM to capture contextual information from both preceding and subsequent texts. In this setup, the forward hidden layer L_f, the backward hidden layer L_b, and the output sequence X_o are employed to update the network. Iterations are performed from t to 1 for backward updates and from 1 to t for forward updates. The update parameters for Bi-LSTM can be represented as:

$$\begin{aligned} L_f &= \sigma(W_1 \cdot x_t + W_2 \cdot L_{f-1} + b_{L_f}) \\ L_b &= \sigma(W_3 \cdot x_t + W_5 \cdot L_{b-1} + b_{L_b}) \\ x_o &= W_4 \cdot L_f + W_6 \cdot L_b + b_{x_o} \end{aligned} \tag{1}$$

where L_f represents the forward pass, L_b represents the backward pass, and X_o denotes the final output layer; W signifies the weight coefficients, while b_{L_f}, b_{L_b}, and b_{x_o} denote the biases. σ is a sigmoid function ranges from 0 to 1. The time step t indicates the current position within the input log sequence, where h_t^f and h_t^b are the hidden state vectors at position t during forward and backward propagations, respectively. Ultimately, these two vectors are concatenated as: $h_t = concat(h_t^f, h_t^b)$, capturing information from both forward and backward directions.

3 Methodology

In this section, we elaborate on the details of our proposed log anomaly detection model.

3.1 Overview

We introduce an innovative log anomaly detection model named LogRoBERTa, as depicted in Fig. 4. At the outset, we employ the Determinantal Point Process (DPP) method (Chen et al. 2018) to construct a diverse and representative training set, minimizing redundancy in log data. DPP optimizes labeling efficiency by selecting a minimal yet comprehensive subset of logs, effectively capturing a wide range of log templates essential for accurate anomaly detection. For robust performance, LogRoBERTa utilizes RoBERTa to encode log text and generate context-related representations. These representations are then processed by an attention-based Bi-LSTM. The attention mechanism highlights critical information in the log text by assigning varying weights to different contextual elements, while the Bi-LSTM captures sequential features by integrating forward and backward information from the log sequence. By combining the strengths of each component, LogRoBERTa effectively identifies anomalous logs without the need for a log parser, even when trained on a limited amount of data.

3.2 Influence of log parsing

Log parsing is commonly used as a preprocessing step to convert unstructured log messages into structured templates by removing dynamic variables such as timestamps, IP addresses, file paths, and numeric values. The resulting abstraction is intended to reduce sequence variability and improve learning efficiency for downstream anomaly detection models. However, in practice, the effectiveness of this step is limited by the accuracy of the parser. As shown in Fig. 2, log parsers often fail to correctly identify all dynamic components or mistakenly treat static tokens as variables. This introduces semantic inconsistencies into the parsed sequences, which may confuse the model during training and degrade detection performance. For example, consider the following log message recorded during a user authentication process:

[2024-11-01 12:08:32] AUTH User admin08: alice IP: 10. 0.0.15 Session: 94a2 Login: web Success: False

This log message contains at least five dynamic fields: timestamp, username, IP address, session ID, and login channel. A well-designed parser should extract a template such as:

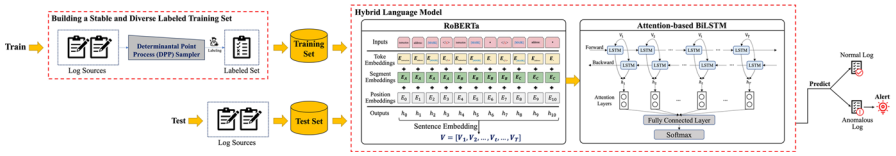


Fig. 4 The overview of LogRoBERTa

[<*>] AUTH User <*>: <> IP: <*> Session: <*> Login: web Success: False

However, many parsers—especially those relying on heuristic rules—tend to remove only surface-level variables (e.g., timestamps and session IDs), while overlooking important contextual fields such as usernames. Moreover, they may mistakenly parse critical Boolean indicators like "Success: False". For instance, some parsers (Jiang et al. 2024) generalize this field into a neutral pattern such as "Success: <*>", thereby discarding the distinguishing value False, which is crucial for identifying login failures. This kind of overly aggressive generalization removes the very feature that separates a failed login attempt from a successful one. As a result, logs reflecting critical system failures or security breaches become indistinguishable from normal activities in the parsed output. This not only increases semantic ambiguity but can also significantly degrade the performance of anomaly detection models, which rely on such discriminative information to detect abnormal behavior accurately. Existing studies have demonstrated that parsing errors can substantially influence the accuracy of log anomaly detection (Xie et al. 2023). While successful removal of irrelevant variables can shorten sequences and facilitate learning, erroneous parsing may remove informative tokens or alter the contextual structure of log messages, thereby reducing the model's ability to capture meaningful patterns. Moreover, the parsing step itself is time-consuming, especially when using LLM-based parsers such as ChatGPT, making it less suitable for real-time or large-scale systems.

To address these limitations, LogRoBERTa is designed to operate directly on raw log sequences without relying on log parsing. Its hybrid architecture—combining the contextual representation capabilities of RoBERTa with the sequential modeling strength of an attention-based BiLSTM—allows it to effectively learn from noisy and variable-length inputs. This observation is further supported by our experimental results, where LogRoBERTa consistently maintains comparable or even superior performance with or without a log parser, highlighting its robustness and practicality for real-world deployment scenarios where log preprocessing can be costly, error-prone, or infeasible.

3.3 Building a stable and diverse labeled training set

To support the training of the hybrid language model, we first extracted the core log sequences without extensive parsing, which significantly reduced the execution time. Given the large scale of the log dataset, manually labeling all entries is impractical. To construct a small yet stable labeled training set, we employed the Determinantal Point Process (DPP) method (Chen et al. 2018), a classical algorithm designed to maximize sample diversity. By selecting a diverse subset of logs, DPP ensures that logs from different templates are represented, reducing the inductive bias associated with imbalanced sample distributions and providing a robust foundation for model training. DPP is particularly effective in handling datasets with high redundancy, as it selects subsets that maximize diversity by favoring dissimilar data points, which is crucial in log anomaly detection where repetitive log sequences are common. Furthermore, DPP allows for a more efficient use of labeling resources by minimizing the number of logs that need to be manually labeled while maintaining a comprehen-

sive representation of the log dataset. This balance between diversity and efficiency makes DPP an ideal choice for creating a robust training set for anomaly detection.

Algorithm 1 Constructing a Stable and Diverse Labeled Training Set Using DPP Method.

Require: Log sequence dataset $X = \{x_i\}_{i=1}^{N}$, Candidate set capacity K
Ensure: Stable and diverse labeled set C
1: Initialize embedding matrix $V = \epsilon(X)$, where ϵ is the embedding function
2: Calculate similarity matrix $L = \text{sim}(V, V)$, where sim represents cosine similarity
3: Initialize empty set $C = \emptyset$
4: **for** $i = 1$ to K **do**
5: Select log sequence x_j that maximizes the marginal gain in dissimilarity:

$$x_j = \arg \max_{x_i \in X \setminus C} \frac{\det(L_{C \cup \{x_i\}})}{\det(L_C)}$$

6: Add x_j to candidate set C: $C = C \cup \{x_j\}$
7: **end for**
8: **Return** C

The detailed algorithm is shown in Algorithm 1. Formally, let the log dataset be represented as a set of log sequences $X = \{x_1, x_2, \ldots, x_N\}$, where each sequence x_i is encoded into a vector representation v_i. These vectors are concatenated to form the embedding matrix $V = \epsilon(X) = [v_1, v_2, \ldots, v_N]$ *(Line 1)*. The similarity between two log sequences x_i and x_j is calculated using cosine similarity *(Line 2)*, yielding the similarity matrix L as follows:

$$L_{ij} = \cos(v_i, v_j) = \frac{v_i \cdot v_j}{\|v_i\| \|v_j\|} \tag{2}$$

The DPP method then selects a subset $C \subset X$ of size K, which maximizes the diversity of the candidate set by maximizing the determinant of the submatrix L_C formed by the selected samples *(Line 4-7)*:

$$\text{argmax}_C \det(L_C) \tag{3}$$

This ensures that the selected subset C is diverse and stable, covering various log sequences effectively. The DPP sampling process is iterative and continues until the predefined set size K is reached, optimizing the total diversity within the selected training set.

Taking the BGL dataset as an example, it contains a total of 4,747,963 log sequences derived from just over 1,800 distinct log templates. These sequences are largely repetitions of the same templates, differing only in their dynamic variables. The Majority of logs contain only 1–2 variables, Making them highly similar in structure. By applying the DPP method, we selected 20,000 labeled logs, representing only 0.42% of the total BGL logs. To ensure the quality of labeled data, we employed a two-round cross-validation annotation process. Specifically, each log entry was independently labeled by two professional software engineers in the first round. In the second round, two additional engineers reviewed the labels and resolved any disagreements

through discussion. All annotators had over five years of experience in software system development and Maintenance. This labeling strategy ensured both consistency and accuracy, and since the DPP method selected only 20,000 representative logs, it significantly reduced the annotation workload compared to other semi-supervised models (Lin and Chiang 2022; Guo et al. 2021) that typically require a much larger volume of labeled data. As a result, LogRoBERTa is provided with a sufficiently diverse and high-quality labeled training set, while avoiding the overhead of labeling large volumes of redundant log sequences.

3.4 Hybrid language model

In this section, we introduce the hybrid language model in LogRoBERTa, which combines the contextual understanding of RoBERTa with the sequential feature extraction capabilities of Bi-LSTM, enabling more accurate anomaly detection, even with limited training data. Unlike BERT, RoBERTa not only optimizes pre-training methods but also enhances the dynamic masking mechanism, thereby capturing critical log information more precisely. Following semantic vectorization, LogRoBERTa utilizes an Attention-based Bi-LSTM neural network for anomaly detection. This enables the model to capture and learn contextual knowledge from log sequences, assigning varying degrees of importance to different types of log sequences. Consequently, the model can identify and emphasize the vocabulary or phrases crucial for understanding log sequences. The fully connected layer serves as the output layer, integrating features from the previous layers and concluding the final log classification.

In detail, after constructing a stable and diverse labeled training set, we obtain the log sequence vector X. To capture the features of the log sequence, LogRoBERTa initially utilizes RoBERTa to comprehend log sequence information. RoBERTa employs dynamic masking, generating a new masking pattern each time a log sequence is input to the model. As depicted in the Fig. 4, log sequences often exhibit considerable repetition, resulting in different masking patterns for identical log sequences. The symbol "</s>" denotes the separator for each log sequence and signifies the end of an individual log sequence. Log sequences are treated as sentence tokens to be evaluated within RoBERTa. A token represents a single input unit obtained after segmenting and adding special markers to the text. Each input word token is initially transformed into a word embedding vector W, achieved by looking up the corresponding word embedded in the vocabulary table. These word embeddings are learned during the model's pre-training, and each word possesses a unique embedding representation. Additionally, the model utilizes positional encoding to maintain the sequential order of words within the sequence.

Word embedding is created through the amalgamation of token embedding, segment embedding, and position embedding. Token embedding converts each log sequence token into a 768-dimensional vector denoted as T, segment embedding produces vector S, and position embedding generates vector P. The fusion of these three vectors yields the embedded vector of the log sequence can be represented as:

$$X = W = T + S + P \tag{4}$$

Word embeddings produce a vector representation h for each token, and these h vectors can be integrated into a sentence-level semantic vector $V = [V_1, V_2, ..., V_t, ..., V_T]$. In RoBERTa, the CLS token is employed to encapsulate the semantic information of the entire input sequence. We use the output vector of the CLS token as the semantic vector V for the entire log sequence, thus obtaining the following equation:

$$V = [V_1, V_2, ..., V_t, ..., V_T] \tag{5}$$

LogRoBERTa employs the semantic vector sequence V as the model input and conducts log anomaly detection using an Attention-based Bi-LSTM network. The Bi-LSTM inherits the characteristics of LSTM and can capture both forward and backward contextual information flow from log sequences. Given the limited number of anomalous log sequences, they carry limited features. To better capture key information from anomalous logs, we introduce a "multihead" attention mechanism.

The "multihead" attention consists of eight attention heads, sequentially calculating attention scores between log sequences. To enhance the fitting ability to log sequences, three matrices are utilized within the "multihead" attention. The vector V is multiplied by the weight matrices W_Q, W_K, and W_V, forming three matrices, namely query matrix Q, key matrix K, and value matrix V. For each head, The semantic vector sequence is input into the self-attention function, resulting in a new vector. Their weight values are obtained using Softmax function. The specific formulas are as follows:

$$\begin{aligned} multihead &= \text{concat}(head_1, head_2, \ldots, head_n) \cdot W^O \\ head_n &= \text{Softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \cdot V \end{aligned} \tag{6}$$

During the model training phase, we employ the cross-entropy as the loss function by comparing the predicted output values with the true values derived from the dataset labels. To minimize the cross-entropy loss function, we employ the Adam optimization algorithm to update the parameters of the model, which includes adjusting the weights of both the Bi-LSTM and Attention layers.

Following the attention mechanism layer, we incorporate log features via a fully connected layer. This layer consists of one or more densely connected neural network layers, primarily tasked with synthesizing the outputs from the Bi-LSTM and Attention mechanism layers. Subsequently, a Softmax layer is employed to transform the output of the fully connected layer into a probability distribution. This transformation facilitates more precise predictions across various log categories by the model.

4 Experimental setup

To evaluate the performance of LogRoBERTa, we utilized four widely recognized public datasets: HDFS, BGL, Thunderbird, and Spirit. At the outset, we compared LogRoBERTa with existing software log anomaly detection models to assess its per-

formance. Subsequently, we examined the necessity of log parsing by testing the performance differences of various deep learning models with and without log parsers. Finally, we conducted experiments with LogRoBERTa on the datasets of varying sizes to evaluate its performance, particularly when dealing with smaller-scale data. We also conducted detailed comparative experiments to showcase the contribution of each module.

4.1 Datasets

We employed two public datasets to evaluate the performance of our model: HDFS (Xu et al. 2009), BGL (Oliner and Stearley 2007), Thunderbird (Oliner and Stearley 2007), and Spirit (Oliner and Stearley 2007) datasets. These datasets are labeled with log types and exhibit substantial differences in log formats, enabling us to assess the model's generalization performance. We constructed a training dataset of 20,000 labeled logs using the DPP method and extracted 20% of the remaining data for testing. The specific distribution of logs is outlined in Table 1.

HDFS dataset was generated from over 200 Amazon EC2 nodes, consisting of a total of 11,175,629 log messages. Based on unique block_ids, all log messages were segmented into 575,061 blocks, forming log message sequences, each of which was labeled as normal or abnormal. Among these, 16,838 (2.93%) log blocks are considered to indicate system anomalies.

BGL (Blue Gene/L) dataset is a supercomputing system log dataset collected by Lawrence Livermore National Labs (LLNL). It contains 4,747,963 log messages, each of which is Manually labeled as normal or abnormal. Among these, 348,460 (7.34%) log messages are labeled as abnormal.

Thunderbird dataset is an open log dataset collected from the Thunderbird supercomputer at Sandia National Labs (SNL). The dataset contains Manually labeled normal and abnormal log messages. For computational feasibility, we use a subset of 10,000,000 continuous log messages from the original dataset of over 200 Million messages. Among these, 4,934 (0.49%) are labeled as abnormal.

Spirit dataset is a supercomputing system log dataset collected from the Spirit supercomputer at Sandia National Labs (SNL). The original dataset consists of over 172 Million log messages, each labeled as either normal or abnormal. For this study, we use a subset of 5,000,000 log messages, of which 764,500 (15.29%) are labeled as abnormal.

Table 1 Number of log messages in each dataset used in LogRoBERTa

Data	Category	Log	Anomaly	Test set	
		Message		Normal	Abnormal
HDFS	Distributed file system	11,175,629 (575,061 blocks)	16,838 blocks (2.93%)	115,012 blocks	3,368 blocks
BGL	Supercomputer	4,747,963	348,469 (7.34%)	879,899	69,694
Thunderbird	Supercomputer	10,000,000	4,934 (0.49%)	1,990,200	9,800
Spirit	Supercomputer	5,000,000	764,500 (15.29%)	847,100	152,900

4.2 Model parameters

In the following experiments, LogRoBERTa was instantiated with specific parameter configurations. The hidden size for RoBERTa and input layers were both designated as 768. A learning rate of 2e-5 was selected for training. Regarding batch sizes, we employed 32 for the HDFS dataset and 64 for the BGL dataset. The vocabulary size was defined as 30522. Subsequently, the Bi-LSTM's hidden size was set to 256 with 4 hidden layers and 8 attention heads. Notably, the Maximum sequence length was specified as 128. During the training process, the CrossEntropyLoss function was utilized in conjunction with the Adam optimizer.

To ensure robustness and Mitigate randomness, we conducted each experiment five times and reported the average results. All experiments were performed on a system running Windows 11 with an Intel(R) Core(TM) i7-12700 CPU, 64GB RAM, and an NVIDIA RTX A4000 GPU.

4.3 Research questions

This paper mainly focuses on the following five research questions and designs our experiments accordingly.

- **RQ1: Can the proposed LogRoBERTa model outperform existing models in log anomaly detection?** In this experiment, following the dataset selection strategies adopted in prior research (Le and Zhang 2022), we evaluated the LogRoBERTa model against six benchmark models on four widely used log datasets: HDFS, BGL, Thunderbird, and Spirit. HDFS represents logs from a distributed file system and features a relatively simple and consistent structure. In contrast, BGL, Thunderbird, and Spirit contain more complex and heterogeneous log sequences, often generated by large-scale systems such as supercomputers or embedded systems. These datasets differ significantly in terms of log volume, template diversity, and structural complexity, thus providing a comprehensive basis for assessing the generalization and robustness of anomaly detection models. The selected comparative models encompass various mainstream architectures such as BERT, LSTM, and Transformer, among others. These benchmark models encompass unsupervised learning-based models (DeepLog Du et al. 2017), supervised learning-based models (HitAnomaly Huang et al. 2020, NeuralLog Le and Zhang 2021, LogRobust Zhang et al. 2019), and semi-supervised learning-based models (PLELog Lin and Chiang 2022; Yang et al. 2021, LogBERT Guo et al. 2021). Detailed descriptions of these benchmark models can be found in Section 4.4. Notably, the experimental outcomes of DeepLog are sourced from the LogBERT paper, while the remaining results are from their respective original research.
- **RQ2: How does the omission of a log parser affect the performance of the model?** In this experiment, We evaluated five log anomaly detection models, comparing their performance with and without log parsers. Additionally, we measured the time saved by bypassing log parsers. This analysis helped us as-

sess how effectively anomaly detection models operate without predefined log templates, offering insights into whether log parsers are still necessary given the capabilities of advanced deep learning models.

- **RQ3: Does each module in LogRoBERTa contribute meaningfully to its overall performance?** In this study, we disassembled each module within LogRoBERTa to evaluate their individual contributions. The hybrid language model consists of three components: RoBERTa as the first module, Bi-LSTM as the second, and an attention mechanism as the third. Since the effectiveness of each module may vary depending on the structural complexity of the log data, ablation studies are essential to verify whether each component meaningfully contributes to the model's overall performance. We calculated the average log sequence length across the four datasets and found that the BGL dataset has an average length of 194 tokens, which is significantly longer than those in the other three datasets. Additionally, BGL contains over 1,500 distinct log event templates, indicating a high degree of structural diversity. The anomaly rate in BGL is also 7.34%, considerably higher than that of the Thunderbird dataset (0.49%), and both the average message length and the number of unique log formats surpass those in the Spirit dataset. Based on these observations, we define BGL logs as more complex (i.e., having a larger number of distinct formats and higher anomaly density) and verbose (i.e., containing longer and more detailed log messages). These properties make BGL a more representative and challenging dataset for evaluating log parsing performance. Therefore, we selected BGL datasets of varying scales for detailed comparison and analysis.
- **RQ4: Does LogRoBERTa maintain strong performance in low-resource scenarios?** While log anomaly detection research has traditionally focused on large-scale, high-volume systems, recent studies (Sun et al. 2025a; Yu et al. 2024) emphasize the growing importance of supporting low-resource environments. In practice, many real-world systems—such as laboratory management platforms, embedded systems, early-stage deployments, and small industrial applications—generate significantly fewer logs due to limited usage or isolated operating conditions. Despite their lower volume, anomalies in these systems can be equally or even more critical, as operational failures often lead to disproportionately high costs or disruptions. Moreover, logs in such environments often exhibit greater diversity and structural variability, making them more challenging to parse and analyze effectively. Frequent software updates and heterogeneous workflows further complicate log interpretation, demanding models that can perform reliably with Minimal training data. To this end, we evaluate LogRoBERTa and several benchmark models under low-resource conditions to assess their generalization and robustness in these practical yet understudied scenarios. To systematically evaluate performance in low-resource scenarios, we experimented with varying scales of the HDFS, BGL, Thunderbird, and Spirit datasets, ranging from 0.1% to 15% of the original size. Specifically, samples of 0.1%, 1%, 5%, 10%, and 15% were used. To ensure comprehensive comparisons, we included representative models from different learning paradigms, providing a robust basis for contrasting their performance with LogRoBERTa. In addition, we utilized two low-re-

source datasets provided by our industry collaborators: the Intelligent Laboratory Management System (ILMS) and the Factory Assembly Line Logs (FALL) datasets. For each system, we continuously collected logs over a designated time period, resulting in datasets containing 54,637 and 31,615 log entries, respectively. Although these datasets are relatively small compared to those collected from large-scale interconnected systems, they represent realistic low-resource scenarios due to their Limited log volume. Both datasets exhibit complex and lengthy log sequences, with 371 and 2,290 distinct formats, respectively. The FALL dataset, in particular, presents greater structural diversity, as it includes logs generated by robotic assembly processes, where subtle anomalies—such as positional deviations or production sequence irregularities—are difficult to capture using traditional methods. These characteristics make ILMS and FALL especially valuable for evaluating the robustness of anomaly detection models under real-world low-resource constraints.

- **RQ5: To what extent do different approaches for building a stable and diverse labeled dataset influence LogRoBERTa's effectiveness?** In this study, we investigate whether different methods for constructing stable and diverse labeled training sets affect log diversity and, in turn, impact the anomaly detection performance of LogRoBERTa. To construct representative subsets from large-scale log data, we considered four widely used selection strategies: Determinantal Point Process (DPP) (Chen et al. 2018), k-Center Greedy (Ding et al. 2019), K-Means-based sampling (Likas et al. 2003), and Random Sampling with Stratification (Meng 2013). The DPP method, which is used in LogRoBERTa by default (as described in Section 3.3), aims to maximize sample diversity via a probabilistic repulsion mechanism. The k-Center Greedy method starts from a randomly chosen log entry and iteratively selects the log that is farthest in embedding space from the current selection set, thereby maximizing coverage by minimizing the worst-case proximity. K-Means-based sampling first clusters all log entries into K groups and then selects one representative log from each cluster—typically the one closest to the centroid or, alternatively, the farthest—to approximate diversity. Finally, stratified random sampling divides the dataset based on class labels (normal vs. abnormal) and randomly selects an equal number of samples from each class to create a balanced subset. We evaluate these four methods from two perspectives: (1) log diversity and (2) their impact on LogRoBERTa's anomaly detection performance. For the diversity assessment, we invited two professional software engineers to manually inspect the labeled training sets produced by each method and assess the extent to which different log formats are covered. For anomaly detection performance, we trained LogRoBERTa separately on each of the four labeled subsets and compared the resulting F1-scores. Following the initial assessment by software engineers, we selected the BGL dataset and the low-resource FALL dataset for this evaluation, as both exhibit greater structural complexity and variability, making them suitable benchmarks for analyzing training set diversity.

4.4 Benchmark models

In this section, we present benchmark models for comparison with LogRoBERTa in terms of performance among various log anomaly detection models. The benchmark models were selected based on the use of a log parser and the learning method.

DeepLog (Du et al. 2017) stands as an innovative framework that employs deep learning techniques, notably Long Short-Term Memory (LSTM) networks, to analyze system log data for anomaly detection. It learns the patterns of typical operations from sequential log entries and detects deviations as potential threats or system malfunctions. By emphasizing the sequential aspect of log data, DeepLog accurately predicts and distinguishes between normal and anomalous system behaviors, making it highly effective for security and maintenance tasks in IT systems.

HitAnomaly (Huang et al. 2020) is a supervised model based on a Transformer architecture. It utilizes a specialized log parser for encoding log information as parameters. Employing a standard log parser, it transforms raw log data into a structured format, which is then analyzed using Transformer encoders. HitAnomaly adopts a classification-based approach, separately encoding both normal and abnormal data types to improve its ability to differentiate between them. This method aims to accurately detect anomalies in system logs by focusing on their unique patterns, making it an invaluable tool for maintaining system integrity and security.

NeuralLog (Le and Zhang 2021) is a supervised classifier model built on the Transformer architecture and incorporates a tokenizer. This strategy empowers the model to learn unique features from raw log data directly without the intermediate step of log parsing, which frequently results in information loss. By integrating a mixture of normal data from the target system and abnormal data from another system, NeuralLog enhances its anomaly detection capabilities, rendering it versatile and efficient for real-world scenarios.

LogRobust (Zhang et al. 2019) is a supervised model built on an attention-based Bi-LSTM architecture. It employs a specialized log parser to preprocess logs, generating TF-IDF and word semantic vectors to extract log features. Both normal and abnormal logs contribute to the training process. The incorporation of attention mechanisms bolsters the model's capacity to pinpoint significant anomalies in log data, thereby enhancing detection accuracy and reliability across various operational contexts.

PLELog (Lin and Chiang 2022; Yang et al. 2021) introduces a semi-supervised log-based anomaly detection technique that employs probabilistic label estimation to efficiently handle unlabeled log data. This innovative approach integrates semantic information from log events into fixed-length vectors, employs HDBSCAN for clustering these vectors, and utilizes an attention-based GRU network to refine the detection process. Empirical evaluations conducted on datasets such as HDFS and BGL showcase PLELog's robust anomaly detection capabilities, thereby diminishing the necessity for extensive manual labeling.

LogBERT (Guo et al. 2021) is a BERT-based model for anomaly detection, employing MLM and DeepSVDD losses during training. It processes log sequences refined by a Drain parser to learn normal log patterns through tasks such as masked log key prediction and hypersphere minimization. This strategy enables LogBERT to

grasp profound contextual relationships within log data, establishing a robust framework for anomaly detection based on deviations from learned log patterns.

4.5 Evaluation metrics

Log anomaly detection is treated as a classification problem, and to assess the effectiveness of models in detecting anomalies, we rely on Precision, Recall, and F1-Score metrics to evaluate model performance. The accuracy metric, which calculates the proportion of all predictions correctly made by the anomaly detection model out of the total number of samples, is not used. This is because, in most datasets, normal logs make up the majority, while anomalous logs constitute only a small portion, which is characteristic of well-qualified software. Consequently, a model could attain a high accuracy score even if it predicts all logs as normal. The definitions of each metric are as follows:

Precision can be regarded as a measure of quality, particularly in binary classification tasks. It quantifies the proportion of correctly identified anomalous log sequences out of all the anomalous log sequences detected by the model. A high precision score signifies that the model is reliable and avoids triggering unnecessary alerts.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{7}$$

Recall evaluates the model's capability to identify all instances of the positive class, representing the percentage of log sequences correctly identified as anomalies out of all anomalies. Recall primarily assesses whether the model can capture all positive samples, and a model with a high recall value indicates that it does not overlook many exceptional cases.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{8}$$

The F1-score represents the harmonic mean of precision and recall. It serves as a metric for assessing the overall effectiveness of a model, particularly when dealing with datasets characterized by imbalanced positive and negative classes. The term "F1" reflects the equal weighting given to precision and recall, making it a commonly used single criterion for evaluating models in the context of anomaly detection.

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{9}$$

Specifically, TP (True Positive) is the count of anomalous log sequences correctly detected by the model. FP (False Positive) is the count of normal log sequences incorrectly identified as anomalies. TN (True Negative) is the count of normal logs that are correctly classified. FN (False Negative) is the count of anomalous log sequences that the model failed to detect. The above four states (TP, FP, TN and FN) can be more effectively visualized and represented using a confusion matrix.

5 Results and analysis

5.1 Effectiveness of LogRoBERTa (RQ1)

In this section, we evaluate the anomaly detection performance of LogRoBERTa by comparing it against six benchmark models across four datasets: HDFS, BGL, Thunderbird, and Spirit, as outlined in Table 2. To summarize, all semi-supervised and fully supervised models outperformed the unsupervised model DeepLog, highlighting the crucial importance of leveraging historical samples in anomaly log detection tasks. Notably, LogRoBERTa, despite not using a parser, demonstrated consistent and outstanding performance across all four datasets, underscoring the advantages of hybrid language models in handling diverse and complex log structures.

In detail, HitAnomaly and LogRobust demonstrate notably better performance on the HDFS dataset compared to the BGL dataset, whereas DeepLog, PLELog, and LogBERT exhibit the opposite trend. Similar patterns are observed in the Thunderbird and Spirit datasets, underscoring the inherent challenge of achieving consistent performance across datasets with diverse log formats and structures. NeuralLog, a supervised model leveraging tokenization and Transformer architecture, performs well but falls short of LogRoBERTa, though it surpasses other models. The discrepancies in log formats across the four datasets hinder the generalization of log parsers. For instance, LogRobust and PLELog, which rely on the Drain parser, show varying performance across these datasets, highlighting the difficulty in maintaining consistent performance for most models.

In contrast, LogBERT, despite utilizing the BERT architecture, does not outperform LogRoBERTa. For example, LogBERT exhibits significant F1-score differences between HDFS (0.823) and BGL (0.908) and performs better on Thunderbird (0.963) than Spirit (0.834). Notably, LogRoBERTa, using only 20,000 labeled logs for training (e.g., just 0.42% of the total BGL dataset), outperforms three supervised models—LogRobust, NeuralLog, and HitAnomaly—by 2.2%, 1.2%, and 1.2% on the HDFS dataset; 16.1%, 1.4%, and 7.1% on the BGL dataset; 30.8%, 3.8%, and 10.8% on the Thunderbird dataset; and 4.1%, 7.6%, and 15.1% on the Spirit dataset. These results demonstrate the competitive advantage of LogRoBERTa, particularly in handling diverse log structures and achieving robust performance across datasets.

Answer to RQ1: Despite substantial disparities in format definitions and structures across the four datasets, LogRoBERTa, trained with only 20,000 labeled logs, consistently outperforms state-of-the-art models, including three supervised learning-based approaches. This demonstrates its remarkable generalization ability to heterogeneous log formats and superior anomaly detection performance.

5.2 Effect of log parser omission (RQ2)

In this section, we evaluate LogRoBERTa alongside four representative log anomaly detection models—DeepLog, LogRobust, PLELog, and LogBERT—comparing their performance both with and without log parsers. These models were selected to represent a diverse range of learning paradigms and architectural designs. Specifically, DeepLog is chosen as a widely used unsupervised baseline. LogRobust represents a

Table 2 The performance of different models on four datasets

Model	HDFS			BGL			Thunderbird			Spirit		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Unsupervised learning-based												
DeepLog	0.884	0.695	0.773	0.897	0.828	0.861	0.484	0.895	0.628	0.438	**1**	0.609
Supervised learning-based												
HitAnomaly	**1**	0.970	0.980	0.950	0.900	0.920	0.950	0.810	0.880	0.920	0.750	0.840
NeuralLog	0.960	**1**	0.980	0.980	0.980	0.980	0.920	**0.990**	0.950	0.919	0.908	0.914
LogRobust	0.980	0.960	0.970	0.910	0.770	0.830	0.610	0.780	0.680	0.985	0.915	0.949
Semi-supervised learning-based												
PLELog	0.950	0.963	0.957	0.965	**0.999**	0.982	0.864	0.941	0.901	0.931	0.767	0.841
LogBERT	0.870	0.781	0.823	0.894	0.923	0.908	0.962	0.965	0.963	0.862	0.807	0.834
LogRoBERTa	0.994	0.991	**0.992**	0.992	0.991	**0.991**	**0.990**	0.986	**0.988**	**0.991**	0.990	**0.991**

fully supervised method that integrates log parsing into its training pipeline. PLELog is included as a semi-supervised model designed to operate with limited labeled data. Additionally, LogBERT is selected to represent BERT-based semi-supervised approaches, allowing for a direct architectural comparison with our proposed model. This selection enables a comprehensive analysis of how different model types respond to the inclusion or omission of log parsers. We assessed five representative parsers on the four datasets outlined in RQ1: four rule-based parsers (Drain He et al. 2017, Spell Du and Li 2016, AEL Jiang et al. 2008, and IPLoM Makanju et al. 2009) and a state-of-the-art LLM-based parser (LiLAC Jiang et al. 2024). In particular, LILAC leverages the GPT-3.5-turbo-0613[1] model. Table 3 showcases the performance of various log parsers across different anomaly detection models on the four benchmark datasets.

Notably, models paired with log parsers do not consistently outperform their counterparts without a parser ("-"). For example, on the BGL dataset, PLELog with either the Drain or Spell parser, LogBERT with the Spell parser, and LogRoBERTa with the Spell parser all exhibited lower F1-scores compared to their parser-free variants. Similar trends were observed across the other three datasets. More notably, LILAC, which leverages ChatGPT as a parser, achieved the best performance across all five models in most cases across the four datasets. On the BGL dataset, for instance, LILAC led to F1-score improvements of 28.3% for PLELog and 4.18% for LogBERT, compared to the average scores obtained using the other four parsers (Drain, Spell, AEL, and IPLoM). Previous work (Sun et al. 2025a) has shown that using ChatGPT for log parsing can significantly improve parsing accuracy (PA), which partially explains the performance gains observed in semi-supervised models like PLELog and LogBERT. A similar trend was found on the Spirit dataset, where LILAC again delivered the best results for three models. However, in scenarios without log parsers, all five models generally outperformed those using certain rule-based parsers, suggesting that such parsers do not always enhance anomaly detection performance. Importantly, LogRoBERTa consistently maintained strong and stable performance across all four datasets, even without the aid of a parser, significantly outperforming other models. These results demonstrate LogRoBERTa's robustness and effectiveness, particularly in handling diverse and complex log formats.

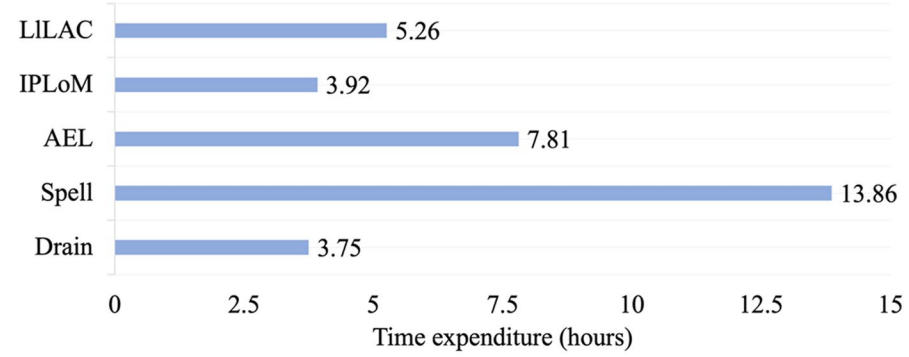
When no log parser is used, the results remain highly competitive, particularly with advanced models. For instance, on the BGL dataset, LogBERT reaches an F1-score of 0.844 without a parser, compared to 0.909 with ChatGPT. Combined with previous findings that some models using specific parsers exhibit lower F1-scores than their parser-free counterparts, these results suggest that while log parsers may offer slight improvements, their overall impact may not justify their use in all cases. Moreover, omitting the log parsing step reduces the total time for model training and evaluation. Figure 5a presents the total parsing time required by different log parsers across the four datasets. Even the fastest parser, Drain, takes approximately 3.75 hours to process all log entries, indicating that parser-based preprocessing can introduce significant delays. This level of time consumption poses a challenge for real-time anomaly detection systems where rapid response is essential. Furthermore, Fig. 5b

[1] https://platform.openai.com/docs/introduction

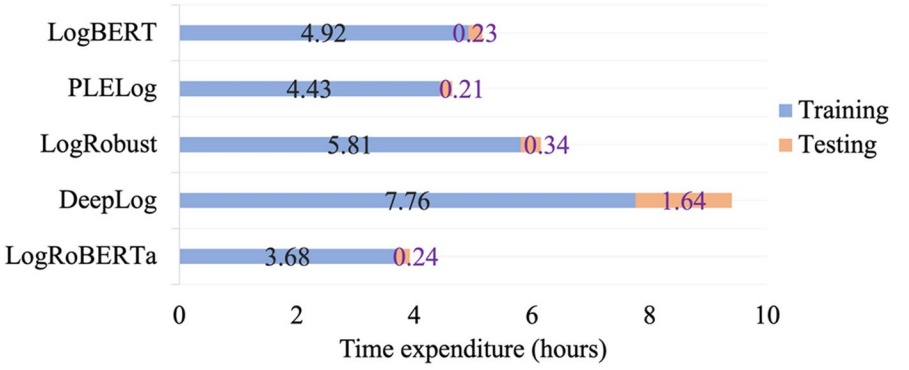
Table 3 The performance of different log parsers on four datasets

Model		BGL						Spirit						HDFS						Thunderbird					
		Drain	Spell	AEL	IPLoM	LILAC	-	Drain	Spell	AEL	IPLoM	LILAC	-	Drain	Spell	AEL	IPLoM	LILAC	-	Drain	Spell	AEL	IPLoM	LILAC	-
DeepLog	P	0.270	0.271	0.271	0.273	0.398	0.310	0.438	0.602	0.413	0.607	0.621	0.528	0.835	0.852	0.903	0.889	0.914	0.849	0.200	0.211	0.200	0.200	0.314	0.187
	R	0.988	0.988	0.988	0.988	0.798	0.568	1	1	1	1	0.994	0.891	0.994	0.981	0.934	0.791	0.975	0.918	1	1	1	1	0.985	0.973
	F1	0.426	0.426	0.425	0.427	**0.531**	0.401	0.609	0.751	0.584	0.755	**0.764**	0.663	0.908	0.912	0.918	0.837	**0.944**	0.882	0.333	0.348	0.333	0.333	**0.476**	0.314
PLELog	P	0.702	0.560	0.661	0.655	0.958	0.879	0.931	0.859	0.863	0.500	0.884	0.737	0.893	0.953	0.967	0.913	0.962	0.958	0.250	1	0.400	1	0.593	0.232
	R	0.791	0.404	0.854	0.433	0.856	0.797	0.767	0.859	0.887	0.662	0.897	0.792	0.979	0.199	0.800	0.781	0.843	0.804	1	0.125	0.500	0.500	0.683	0.461
	F1	0.744	0.476	0.744	0.521	**0.904**	0.836	0.841	0.859	0.875	0.570	**0.890**	0.764	**0.934**	0.329	0.864	0.842	0.899	0.874	0.400	0.222	0.444	**0.667**	0.635	0.309
LogBERT	P	0.897	0.887	0.824	0.831	0.901	0.828	0.862	0.793	0.752	0.774	0.883	0.802	0.870	0.863	0.868	0.845	0.881	0.859	0.962	0.920	0.957	0.929	0.971	0.934
	R	0.907	0.881	0.853	0.859	0.917	0.861	0.807	0.767	0.793	0.803	0.871	0.759	0.781	0.787	0.792	0.763	0.790	0.776	0.965	0.972	0.951	0.954	0.960	0.952
	F1	0.902	0.884	0.838	0.845	**0.909**	0.844	0.834	0.780	0.772	0.788	**0.877**	0.780	0.823	0.823	0.828	0.802	**0.833**	0.815	0.963	0.945	0.954	0.941	**0.965**	0.943
LogRobust	P	0.910	0.849	0.844	0.726	0.902	0.693	0.985	0.947	0.986	0.973	0.980	0.963	0.961	0.938	0.896	0.930	0.973	0.950	0.900	0.805	0.852	0.862	0.883	0.817
	R	0.770	0.988	0.982	0.977	0.926	0.802	0.915	1	1	1	0.942	0.940	1	0.999	1	1	1	0.993	1	0.988	0.982	0.982	0.974	0.969
	F1	0.830	0.914	0.908	0.833	**0.915**	0.743	0.949	0.973	**0.993**	0.986	0.965	0.951	0.980	0.968	0.945	0.964	**0.986**	0.971	0.947	0.887	0.913	0.918	**0.989**	0.887
LogRoBERTa	P	0.992	0.992	0.992	0.991	0.993	0.994	0.992	0.990	0.990	0.990	0.990	0.991	0.995	0.993	0.994	0.991	0.995	0.994	0.990	0.987	0.991	0.982	0.990	0.990
	R	0.992	0.991	0.993	0.993	0.992	0.991	0.989	0.989	0.991	0.988	0.990	0.990	0.990	0.990	0.991	0.986	0.991	0.991	0.982	0.988	0.984	0.980	0.985	0.986
	F1	**0.992**	0.991	**0.992**	**0.992**	**0.992**	**0.992**	0.990	0.990	**0.991**	0.989	0.990	**0.991**	0.992	0.991	0.992	0.988	**0.993**	0.992	0.986	0.987	0.986	0.981	0.987	**0.988**

Note: The highest performance for each model is highlighted in **bold**. The performance without using a log parser is denoted by "-"



(a) Overall time cost of log parsing.



(b) Overall time cost of anomaly detection.

Fig. 5 Computational efficiency of log analysis tasks across four datasets

illustrates the total runtime of five anomaly detection models on the same datasets. Among them, DeepLog exhibits the highest time cost, while LogRoBERTa is the most efficient. In particular, LogRoBERTa achieves the lowest runtime During both the training and testing phases, with average time reductions of 32.9% and 24.1%, respectively, compared to the other four models. Despite its lower computational cost, LogRoBERTa consistently delivers superior detection performance, making it a practical choice for rapid deployment and real-time anomaly detection scenarios.

Interestingly, the LogRoBERTa model demonstrates nearly identical performance with and without the use of a log parser, achieving F1-scores of 0.992, 0.991, 0.993, and 0.988 across the four datasets, respectively. This consistency underscores that for advanced models like LogRoBERTa, log parsers contribute Minimally to detection accuracy. In fact, omitting the log parsing step results in substantial time savings, reducing runtime by 28.3%, 33.6%, 40.8%, and 44.1% on the respective datasets. By directly processing raw log data, LogRoBERTa maintains its exceptional performance while streamlining the pipeline for enhanced efficiency. These findings indicate that for sophisticated models such as LogRoBERTa, log parsing is not essential. The decision to use a parser, as well as the choice of parser, should be guided by the

model's performance and specific application requirements, particularly in scenarios demanding fast or real-time anomaly detection.

Answer to RQ2: LogRoBERTa achieved similar performance with and without a log parser, demonstrating that a parser is not essential for effective anomaly detection. Alongside other benchmark models, it highlights that advanced models can maintain high performance even without a log parser, as evidenced by the minimal differences observed in LogBERT, while also saving significant time. Thus, the necessity and time consumption of using a parser should be reconsidered in the design of future anomaly detection models.

5.3 Contribution of each module within LogRoBERTa (RQ3)

In this section, we disassembled each module within LogRoBERTa to evaluate their contribution. As detailed in Table 4, Line 1 is the full model of LogRoBERTa, Lines 4 and 5 are designated to explore the RoBERTa module's enhancement effects, Line 2 is utilized to assess the Bi-LSTM module's improvement, and Line 3 is employed to investigate the attention mechanism's enhancement effects. Given the complexity of the BGL dataset compared to other datasets, the differences in performance among modules are expected to be more pronounced. Therefore, we selected BGL datasets at various scales—1%, 10%, 20%, 50%, and 100%—to conduct more extensive experiments in this section. It is worth noting that the training set was also scaled down proportionally. For instance, at the 1% scale, LogRoBERTa used only 200 labeled logs, despite there being over 1,000 distinct log templates. This implies that LogRoBERTa may not be able to capture the features of all the different logs.

In summary, each module of LogRoBERTa significantly contributes to the model's performance improvement. Across all configurations, we observed consistent trends related to dataset size: as the dataset size increased, F1-scores also increased, underscoring the importance of sufficient data for model performance. Furthermore, the performance degradation of Line 5, compared to RoBERTa and BERT-based configurations, highlights the superior capabilities of large-scale pre-trained language models in capturing contextual information and semantic relationships within the data.

Table 4 The performance of three modules on the BGL dataset

Module		DataSet Size				
		1%	10%	20%	50%	100%
RoBERTa						
(1) + Bi-LSTM + Attention	F1	0.965	0.986	0.991	0.991	0.991
(2) + Attention	F1	0.838 (-12.7%)	0.861 (-12.5%)	0.888 (-10.3%)	0.893 (-9.8%)	0.902 (-8.9%)
(3) + Bi-LSTM	F1	0.948 (-1.7%)	0.977 (-0.9%)	0.983 (-0.8%)	0.984 (-0.7%)	0.985 (-0.6%)
BERT						
(4) + Bi-LSTM + Attention	F1	0.944 (-2.1%)	0.971 (-1.5%)	0.980 (-1.1%)	0.980 (-1.1%)	0.982 (-0.9%)
(5) Bi-LSTM + Attention	F1	0.739 (-22.6%)	0.774 (-21.2%)	0.792 (-19.9%)	0.809 (-18.2%)	0.801 (-19.0%)

In terms of horizontal comparison, it is worth noting that when the dataset size is reduced to 1%, the labeled training logs do not cover all log variations. This Limitation prevents LogRoBERTa from learning all log features, resulting in a 2.1% decrease in the F1-score compared to the 10% dataset scale. Similarly, Lines 2 to 5 exhibit significant performance declines.

As part of the vertical comparison, we calculated the average F1-score improvements over these five dataset scales to assess the contribution of each model component. Specifically, replacing or removing the first module with BERT, instead of using RoBERTa, results in average F1-score improvements of 1.34% and 20.18%, respectively, with RoBERTa. Introducing the Bi-LSTM module further improves the average F1-score by 10.84%, while adding an attention layer leads to a 0.94% improvement. When the dataset size is reduced to 1%, LogRoBERTa shows an approximately 1.1% higher F1-score than Lines 3 and 4, and improvements of 12.7% and 22.6% over Lines 2 and 5, respectively. This underscores the importance of integrating large-scale pre-trained language models. At the 10% dataset scale, with 2,000 training logs covering all log variations, the F1-scores of Line 2 and Line 5 only increase by 0.2% and 1.4%, respectively, compared to the 1% dataset size, indicating Minimal improvement. As the dataset size increases to 20%, all models tend to show stabilization in F1-scores.

Answer to RQ3: Overall, each component of LogRoBERTa contributes to enhancing log anomaly detection performance, with the RoBERTa module (Module 1) showing the most substantial improvement.

5.4 Exploration in low-resource scenarios (RQ4)

In this section, we first introduce two low-resource datasets, ILMS and FALL, provided by our industry collaborators. In both datasets, log messages labeled as "Info" and "Warning" are treated as normal, while those labeled "Error" are considered anomalies. A training set of 4,000 labeled logs was constructed using the DPP method, and 20% of the remaining logs were reserved for testing. The model parameters were configured as described in Section 4.2. The following are example logs and detailed descriptions of the log contexts across different severity levels in our software systems.

Note: Certain parts containing privacy information are replaced with asterisks (*).
(ILMS) **"level": "Info"**, "context": {"userId": "********", "ipAddress": "***.***.***.***", "sessionId": "*******"}
(FALL) [2024-10-17 08:21:14.152] **INFO**: Arm A executed task ID=TX-**** at Station=7 in 3.12s (X=124.5, Y=88.3, Z=20.0)
(FALL) [2024-10-19 09:45:33.994] **INFO**: Robot C aligned component ID=R*** on fixture B2 (Tolerance=±0.02mm)
(ILMS) **"level": "Warning"**, "context": {"error": "Connection timeout", "attemptNumber": 3, "lastAttemptTime": "2023-05-19T12:30:00Z"}
(FALL) [2024-10-21 09:17:08.381] **WARNING**: Torque on Arm B joint-3 exceeded threshold (5.1Nm > 5.0Nm) during operation ID=PK-***
(FALL) [2024-10-21 11:05:01.728] **WARNING**: Delay in pickup sequence at Buffer Zone 3. Expected=2.0s, Actual=4.9s
(ILMS) **"level": "Error"**, "context": {"class": "OrderProcessor", "method": "processOrder", "exceptionDetails": "java.lang.NullPointerException: Invalid null value"}
(FALL) [2024-10-19 06:36:20.145] **ERROR**: Task ID=QX-**** failed. Position deviation exceeded tolerance (X=127.3mm, Expected=125.0±1.0mm)
(FALL) [2024-05-25 09:22:48.410] **ERROR**: Failed to write result file to /mnt/logs/batch/20240525/QX****/result.json (I/O Error Code=E204)

The **ILMS** dataset was collected from an intelligent laboratory management system deployed in chemical and materials research facilities. Developed in Java and instrumented with Apache Log4j, the system logs events in a structured JSON format to support automated processing and improve interpretability. The logs cover a wide range of operations, including user actions, instrument control commands, job scheduling, and data synchronization. During a 50-hour monitoring period, we recorded 54,637 log entries with 371 distinct formats, of which 5.73% were labeled as anomalies.

The **FALL** dataset was gathered from a production-line monitoring system in an operational factory environment. This system integrates multiple hardware and software components to coordinate tasks such as cutting, assembling, and quality inspection of industrial parts. Log messages in FALL are highly Heterogeneous, often reflecting robotic arm movements, positional deviations, and production sequence irregularities. Due to the precision required in these tasks, the logs are highly contextual and feature significant structural variation. Over a 20-day continuous production period in October 2024, we collected 32,615 raw log entries comprising 2,290 distinct formats, with 9.56% labeled as anomalies.

To systematically evaluate the performance of the proposed LogRoBERTa model under low-resource scenarios, we selected six datasets. Specifically, we sampled varying proportions of four large-scale datasets—HDFS, BGL, Thunderbird, and Spirit—at scales of 0.1%, 1%, 5%, 10%, and 15%. These datasets represent typical interconnected systems. In contrast, our two industrial datasets, ILMS and FALL, contain only 54,637 and 32,615 log entries, respectively, and are inherently low-resource. For these datasets, we conducted experiments at 10%, 50%, and 100% of the full data volume. Additionally, due to the Limited data size, we employed the DPP method to construct a training set of 4,000 labeled logs for LogRoBERTa. To ensure a comprehensive comparison, we selected representative anomaly detection models from different learning paradigms: DeepLog (unsupervised), PLELog (semi-supervised), and LogRobust (supervised). The experimental results are summarized in Fig. 6.

As shown in Fig. 6, as the data size decreases to the range of 15% to 0.1%, the F1-scores of DeepLog and PLELog decline by an average of 19.7%-28.4% and 3.5%-20.5%, respectively, across the HDFS and BGL datasets. Similarly, the fully supervised model LogRobust exhibits a Maximum performance drop of 21.6%. Notably, the performance degradation on the Thunderbird and Spirit datasets is even more pronounced, highlighting the difficulty these models face in generalizing across diverse datasets with varying log structures. On our two low-resource datasets, ILMS and FALL, all baseline models exhibit notable performance drops at the 10% data scale, with LogRobust and PLELog showing the most significant declines. In contrast, LogRoBERTa demonstrates remarkable robustness: although its performance also decreases with reduced data availability, the decline is minimal. Even at the smallest data scale, the average F1-score across the six datasets drops by only 8.5%, underscoring LogRoBERTa's strong generalization capabilities in low-resource anomaly detection scenarios.

In detail, all three models, except LogRoBERTa, exhibit unstable anomaly detection performance on smaller dataset scales. For instance, as illustrated in Fig. 6(a), at a dataset size of 0.1% for the HDFS dataset, LogRoBERTa achieves F1-score increases of 31.1%, 16.5%, and 43.8% compared to DeepLog, PLELog, and LogRo-

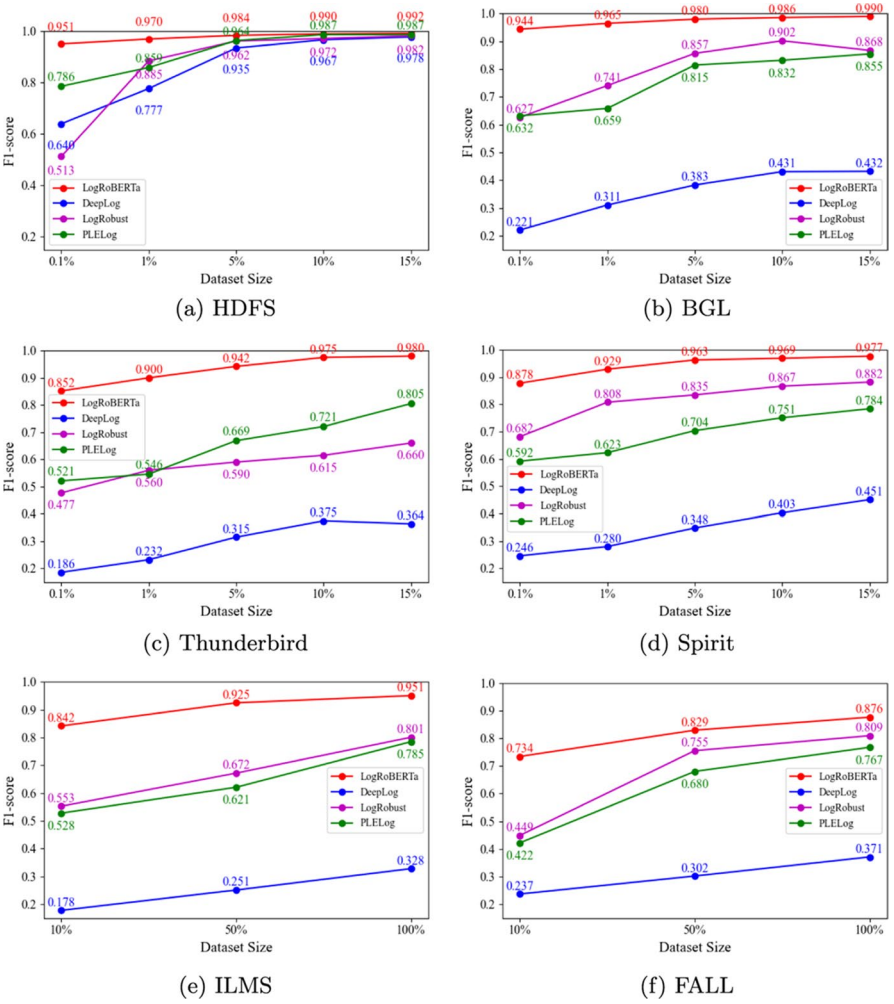


Fig. 6 The performance of models on six low-resource datasets

bust, respectively. As the dataset size grows, all models show the most significant F1-score improvements at the 5% scale, with increases of 29.5% (DeepLog), 17.8% (PLELog), 44.9% (LogRobust), and 3.4% (LogRoBERTa) compared to the 0.1% scale. Upon reaching a dataset size of 10%, the F1-scores of all models stabilize, and the performance gap narrows.

As demonstrated by Fig. 6b, at a dataset size of 0.1% for the BGL dataset, LogRoBERTa achieves an F1-score of 0.944, outperforming PLELog, the second-best performer, by 31.3%. In contrast, LogRobust only attains an F1-score of 0.627, indicating that, without pre-trained language models, even fully supervised models struggle to learn log sequence features in datasets with intricate log formats and extremely limited data. As the dataset size increases, all four models show improved F1-scores, though LogRobust exhibits some fluctuations. For instance, at the 15%

scale, LogRobust's F1-score is 3.4% lower than at 10%. Meanwhile, at 15%, LogRoBERTa reaches an F1-score of 0.990, surpassing PLELog by 13.5%. A similar trend is observed in Figs. 6c and d. At the 0.1% dataset scale, LogRoBERTa achieves an F1-score approximately 45.8% higher than the average of the other three models on the Thunderbird dataset, and 37.1% higher on the Spirit dataset, highlighting its superior predictive accuracy and strong generalization ability in low-resource settings.

Figures 6e and f illustrate the performance of various models on our two low-resource datasets, ILMS and FALL. Due to the limited volume of system logs, labeled data for training is insufficient, resulting in F1-scores at the 10% scale that are, on average, 19.1% and 24.5% lower than those at the 100% scale. Among the baseline models, the unsupervised DeepLog performed the worst under these conditions. Notably, when the data size is reduced to 10%, LogRoBERTa significantly outperforms the other three models, achieving average F1-score improvements of 42.3% on ILMS and 36.5% on FALL. Even as the dataset scale increases to 50% and 100%, LogRoBERTa maintains its performance advantage, reaching F1-scores of 92.5% and 95.1% on ILMS, and 82.9% and 87.6% on FALL, respectively—substantially higher than those of the other models. These results demonstrate LogRoBERTa's strong robustness and generalization ability across diverse real-world software systems.

Answer to RQ4: LogRoBERTa consistently outperforms benchmark models from various learning paradigms, demonstrating robust performance across diverse low-resource data scales. Its superior and consistent results across the six datasets further underscore its strong generalization capabilities.

5.5 Effectiveness of diverse strategies for labeled training set construction (RQ5)

In this section, we investigate whether four widely used selection strategies—DPP, k-Center Greedy, K-Means-based sampling, and Random Sampling with Stratification—affect log diversity and, in turn, influence the anomaly detection performance of LogRoBERTa. These four methods were evaluated from two perspectives. For the diversity assessment, we invited two professional software engineers to manually inspect the labeled training sets generated by each method. To quantify diversity, we computed the coverage ratio (RT), defined as the number of distinct log templates in the selected subset divided by the total number of templates in the dataset. For anomaly detection performance, we trained LogRoBERTa separately on each of the four labeled subsets and compared the resulting F1-scores. Notably, for the BGL dataset, we constructed a training set of 20,000 labeled logs, while for the low-resource FALL dataset, due to its Limited size, we constructed a training set containing 4,000 labeled logs. The final results are summarized in Table 5.

Overall, the results in Table 5 demonstrate that the diversity of the labeled training set (measured by the template coverage ratio, RT) has a direct impact on anomaly detection performance. Both the k-Center Greedy and DPP methods achieved relatively high RT scores on the BGL and FALL datasets, significantly outperforming the other two strategies. Correspondingly, LogRoBERTa trained on these subsets also achieved the highest F1-scores, indicating a strong correlation between template diversity and model effectiveness. In contrast, the stratified random sampling

Table 5 Performance of different subset selection methods on log diversity and anomaly detection

Strategy	BGL				FALL			
	RT	P	R	F1	RT	P	R	F1
k-Center Greedy	93.7%	0.984	0.990	0.987	84.8%	0.871	0.846	0.858
K-Means-based	79.2%	0.970	0.978	0.974	71.5%	0.836	0.815	0.825
Random Sampling with Stratification	52.5%	0.827	0.803	0.815	57.4%	0.760	0.784	0.772
DPP (default)	95.9%	0.992	0.991	0.991	88.2%	0.894	0.859	0.876

method resulted in the lowest RT on both datasets, and the corresponding LogRoBERTa model consistently yielded the poorest detection performance. These findings confirm that the construction strategy of the labeled training set substantially affects downstream anomaly detection results.

Compared to the BGL dataset, the FALL dataset contains a more diverse set of log formats (2,290 in total). Moreover, due to its domain-specific characteristics—such as positional deviations and production sequence irregularities—logs from different formats May share similar patterns. This semantic overlap Makes it more challenging to achieve high template coverage. As a result, even the best-performing DPP method achieved an RT that was 7.7% lower on FALL than on BGL. Interestingly, due to the smaller overall log volume in FALL, stratified random sampling achieved a slightly higher RT than it did on BGL, though it still remained the lowest among the four methods. Despite these dataset-specific differences, both datasets exhibited the same general trend: higher RT scores led to improved anomaly detection performance. For instance, BGL consistently exhibited higher RT values across all sampling strategies, along with higher average F1-scores, reinforcing the positive correlation between log diversity and anomaly detection performance. These findings highlight that maximizing training set diversity is essential for improving the effectiveness of anomaly detection models.

Answer to RQ5: The experimental results demonstrate that the diversity of the labeled training set plays a crucial role in determining model performance. Higher template coverage consistently leads to better anomaly detection results across both balanced and low-resource scenarios.

6 Threats to validity

This section clarifies the threats to log formatting, validity of parser selection, randomness in DPP construction, and system upgrades, respectively.

Log Formatting Logs are produced during software runtime via output statements authored by programmers. Consequently, the definition and output format of logs may differ among programmers across various countries and regions. In certain instances, logs might be generated using the programmer's native language, leading to variations in the performance of the same model when handling logs defined in different languages. Therefore, it is necessary to further evaluate the model's performance in multilingual scenarios.

Validity of Parser Selection While we observed that certain models, such as LogRoBERTa and CNN, demonstrated competitive performance without a log parser, the generalizability of this finding to other datasets or tasks remains uncertain. The selection of specific parsers (Drain, Spell, and ChatGPT) may have influenced the results, and other parsers could yield different outcomes. Additionally, the performance gains from using parsers might vary based on the nature of the log data or the underlying model architecture, potentially impacting results in different contexts. Future work should explore a wider range of parsers, and a more systematic evaluation is needed to determine whether the complex and time-consuming step of log parsing can be omitted in more advanced models.

Randomness in DPP Construction The use of DPP method to construct training dataset introduces an element of randomness that may affect the consistency of our results. While DPP is designed to create diverse and representative samples, the inherent variability in its output could lead to fluctuations in performance metrics. To mitigate this threat, we conducted five independent experiments for each experimental setting and used the mean of the results as the final outcome. This approach enhances the reliability of our findings by accounting for variability and ensuring that our conclusions are based on a robust aggregation of results.

System Upgrades In real-world production settings, log formats may change due to system upgrades, even within the same system. When a large influx of new logs occurs, fine-tuning might be needed to ensure the model remains effective. Additionally, as technology progresses and requirements evolve, the emergence of new log formats is expected, and existing formats may also undergo enhancements and extensions. Our future efforts will involve analyzing log data from diverse datasets to better cater to the needs of anomaly detection tasks.

7 Conclusion

In this paper, we empirically evaluate various log parsers to assess their effectiveness and necessity for anomaly detection across several deep learning models. We also introduce LogRoBERTa, a lightweight and robust anomaly detection model that requires minimal labeled data and integrates RoBERTa with an attention-based BiLSTM to enhance contextual understanding. Extensive experiments demonstrate that LogRoBERTa consistently outperforms existing methods and maintains strong detection performance even without relying on log parsers. In addition, we conduct a comprehensive comparison of five models with and without parsers, revealing that LogRoBERTa achieves comparable results in both settings. This finding challenges the conventional assumption that log parsers are essential and encourages a reevaluation of their necessity and associated time cost in future anomaly detection model design. We also introduce a low-resource evaluation perspective and show that LogRoBERTa exhibits strong generalization and effectiveness even when trained on limited data, highlighting its versatility in both large-scale and resource-constrained scenarios. Furthermore, our ablation studies and experiments on diverse training set

construction strategies validate the effectiveness of each component within LogRoBERTa. Together, these results position LogRoBERTa as a robust and parser-independent solution for log anomaly detection.

In future work, we plan to explore the integration of domain-specific knowledge into LogRoBERTa to enhance its interpretability and adaptability across different software systems. For instance, incorporating structured metadata—such as component tags, service names, or execution stages—into the input representation may help the model better distinguish between contextual variations in logs. Additionally, we aim to extend LogRoBERTa to multilingual and cross-domain log environments by pretraining it on diverse log sources using contrastive learning or adapter-based fine-tuning. This would enable the model to generalize more effectively to unseen log formats and system architectures. We also intend to investigate the development of lightweight variants of LogRoBERTa for deployment in latency-sensitive or resource-constrained settings, potentially through model pruning or knowledge distillation techniques.

8 Supplementary information

The related source code and experiments are publicly available at https://github.com/log-detection/LogRoBERTa.

Acknowledgements This work is supported in part by the General Research Fund of the Research Grants Council of Hong Kong and the Research Funds of the City University of Hong Kong (6000796, 9229109, 9229098, 9220103, 9229029).

Author Contributions Y.S. and J.K. conceptualized the study and led the data collection process. Z.Y., Y.S., and S.L. conducted the experiments and contributed to the data analysis. Y.S. wrote the main manuscript text. Y.S. prepared all the tables. H.Y. prepared all the figures. All authors reviewed and approved the final manuscript.

Data Availability The source code is provided in the Supplementary Information of the manuscript.

Declarations

Competing Interests The authors declare no competing interests.

References

Chen, S., Liao, H.: Bert-log: Anomaly detection for system logs based on pre-trained language model. Appl. Artif. Intell. **36**(1), 2145642 (2022)

Chen, Z., Liu, J., Gu, W., Su, Y., Lyu, M.R.: Experience report: Deep learning-based system log analysis for anomaly detection. arXiv preprint arXiv:2107.05908 (2021)

Chen, L., Zhang, G., Zhou, E.: Fast greedy map inference for determinantal point process to improve recommendation diversity. Adv. Neural Inf. Process. Syst. **31** (2018)

Clark, K., Khandelwal, U., Levy, O., Manning, C.D.: What does bert look at? an analysis of bert's attention. arXiv preprint arXiv:1906.04341 (2019)

Dai, H., Li, H., Chen, C.S., Shang, W., Chen, T.H.: Logram: Efficient log parsing using nn-gram dictionaries. IEEE Trans. Softw. Eng. **48**(3), 879–892 (2020)

Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
Ding, H., Yu, H., Wang, Z.: Greedy strategy works for k-center clustering with outliers and coreset construction. arXiv preprint arXiv:1901.08219 (2019)
Du, M., Li, F., Zheng, G., Srikumar, V.: Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In Proceedings of the 2017 ACM SIGSAC conference on computer and communications security, pp. 1285–1298. (2017)
Du, M., Li, F.: Spell: Streaming parsing of system event logs. In 2016 IEEE 16th International Conference on Data Mining (ICDM), pp. 859–864. IEEE (2016)
Graves, A., Graves, A.: Long short-term memory. Supervised sequence labelling with recurrent neural networks, pp. 37–45. (2012)
Guo, H., Yuan, S., Wu, X.: Logbert: Log anomaly detection via bert. In 2021 international joint conference on neural networks (IJCNN), pp. 1–8. IEEE (2021)
Hashemi, S., Mäntylä, M.: Onelog: towards end-to-end software log anomaly detection. Autom. Softw. Eng. **31**(2), 37 (2024)
He, S., Zhu, J., He, P., Lyu, M.R.: Experience report: System log analysis for anomaly detection. In 2016 IEEE 27th international symposium on software reliability engineering (ISSRE), pp. 207–218. IEEE (2016)
He, P., Zhu, J., Zheng, Z., Lyu, M.R.: Drain: An online log parsing approach with fixed depth tree. In 2017 IEEE international conference on web services (ICWS), pp. 33–40. IEEE (2017)
Hu, C., Sun, X., Dai, H., Zhang, H., Liu, H.: Research on log anomaly detection based on sentence-bert. Electronics **12**(17), 3580 (2023)
Huang, Z., Xu, W., Yu, K.: Bidirectional lstm-crf models for sequence tagging. arXiv preprint arXiv:1508.01991 (2015)
Huang, S., Liu, Y., Fung, C., He, R., Zhao, Y., Yang, H., Luan, Z.: Hitanomaly: Hierarchical transformers for anomaly detection in system log. IEEE Trans. Netw. Serv. Manag. **17**(4), 2064–2076 (2020)
Huo, Y., Su, Y., Lee, C., Lyu, M.R.: Semparser: A semantic parser for log analytics. In 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), pp. 881–893. IEEE (2023)
Jawahar, G., Sagot, B., Seddah, D.: What does BERT learn about the structure of language?. In ACL 2019-57th Annual Meeting of the Association for Computational Linguistics (2019)
Jiang, Z.M., Hassan, A.E., Flora, P., Hamann, G.: Abstracting execution logs to execution events for enterprise applications (short paper). In 2008 The Eighth International Conference on Quality Software, pp. 181–186. IEEE (2008)
Jiang, Z., Liu, J., Chen, Z., Li, Y., Huang, J., Huo, Y., He, P., Gu, J., Lyu, M.R.: Lilac: Log parsing using llms with adaptive parsing cache. Proc. ACM Softw. Eng. **1**(FSE), 137–160 (2024)
Jiang, Z., Liu, J., Huang, J., Li, Y., Huo, Y., Gu, J., Chen, Z., Zhu, J., Lyu, M.R.: 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, pp. 223–234. (2024)
Kulshrestha, A., Krishnaswamy, V., Sharma, M.: Bayesian bilstm approach for tourism demand forecasting. Ann. Tour. Res. **83**, 102925 (2020)
Landauer, M., Onder, S., Skopik, F., Wurzenberger, M.: Deep learning for anomaly detection in log data: A survey. Mach. Learn. Appl. **12**, 100470 (2023)
Le, V.H., Zhang, H.: Log parsing with prompt-based few-shot learning. arXiv preprint arXiv:2302.07435 (2023)
Le, V.H., Zhang, H.: Log-based anomaly detection with deep learning: How far are we?. In Proceedings of the 44th international conference on software engineering, pp. 1356–1367. (2022)
Le, V.H., Zhang, H.: Log-based anomaly detection without log parsing. In 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 492–504. IEEE (2021)
Lee, Y., Kim, J., Kang, P.: Lanobert: System log anomaly detection based on bert masked language model. Appl. Soft Comput. **146**, 110689 (2023)
Li, Z., Luo, C., Chen, T.H., Shang, W., He, S., Lin, Q., Zhang, D.: Did we miss something important? studying and exploring variable-aware log abstraction. In 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), pp. 830–842. IEEE (2023)
Likas, A., Vlassis, N., Verbeek, J.J.: The global k-means clustering algorithm. Pattern Recogn. **36**(2), 451–461 (2003)
Lin, Y., Chiang, Y.Y.: A Semi-Supervised Learning Approach for Abnormal Event Prediction on Large Network Operation Time-Series Data, in 2022 IEEE International Conference on Big Data (Big Data), pp. 1024–1033. IEEE (2022)

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V.: Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692 (2019)
Ma, Z., Chen, A.R., Kim, D.J., Chen, T.H., Wang, S.: Llmparser: An exploratory study on using large language models for log parsing. In Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, pp. 1–13. (2024)
Makanju, A.A., Zincir-Heywood, A.N., Milios, E.E.: Clustering event logs using iterative partitioning. In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 1255–1264. (2009)
McInnes, L., Healy, J.: Accelerated hierarchical density based clustering. In 2017 IEEE International Conference on Data Mining Workshops (ICDMW), pp. 33–42. IEEE (2017)
Memory, L.S.T.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (2010)
Meng, W., Liu, Y., Zhu, Y., Zhang, S., Pei, D., Liu, Y., Chen, Y., Zhang, R., Tao, S., Sun, P., et al.: Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In IJCAI, vol. 19, pp. 4739–4745. (2019)
Meng, X.: Scalable simple random sampling and stratified sampling. In International conference on machine learning, pp. 531–539. PMLR (2013)
Mi, H., Wang, H., Zhou, Y., Lyu, M.R.T., Cai, H.: Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems. IEEE Trans. Parallel Distrib. Syst. **24**(6), 1245–1255 (2013)
Nedelkoski, S., Bogatinovski, J., Acker, A., Cardoso, J., Kao, O.: Self-attentive classification-based anomaly detection in unstructured logs. In 2020 IEEE International Conference on Data Mining (ICDM), pp. 1196–1201. IEEE (2020)
Nyyssälä, J., Mäntylä, M., Varela, M.: How to Configure Masked Event Anomaly Detection on Software Logs?. In 2022 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 414–418. IEEE (2022)
Nyyssölä, J., Mäntylä, M.: Efficiency of unsupervised anomaly detection methods on software logs. arXiv preprint arXiv:2312.01934 (2023)
Oliner, A., Stearley, J.: What supercomputers say: A study of five system logs. In 37th annual IEEE/IFIP international conference on dependable systems and networks (DSN'07), pp. 575–584. IEEE (2007)
Sun, Y., Keung, J.W., Yang, Z., Liu, S., Yu, H.K.: Semirald: A semi-supervised hybrid language model for robust anomalous log detection. Inf. Softw. Technol. 107743 (2025a)
Sun, Y., Keung, J.W., Yang, Z., Liu, S., Liao, Y.: SemiSMAC: A semi-supervised framework for log anomaly detection with automated hyperparameter tuning. Inf. Softw. Technol. 107869 (2025b)
Tenney, I., Das, D., Pavlick, E.: Bert rediscovers the classical nlp pipeline. arXiv preprint arXiv:1905.05950 (2019)
Tufek, A., Aktas, M.S.: On the provenance extraction techniques from large scale log files. Concurr. Comput. Pract. Experience **35**(15), e6559 (2023)
Wu, X., Li, H., Khomh, F.: On the effectiveness of log representation for log-based anomaly detection. Empir. Softw. Eng. **28**(6), 137 (2023)
Xie, X., Jian, S., Huang, C., Yu, F., Deng, Y.: LogRep: Log-based Anomaly Detection by Representing both Semantic and Numeric Information in Raw Messages. In 2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE), pp. 194–206. IEEE (2023)
Xu, W., Huang, L., Fox, A., Patterson, D., Jordan, M.I.: Detecting large-scale system problems by mining console logs. In Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, pp. 117–132. (2009)
Xu, J., Yang, R., Huo, Y., Zhang, C., He, P.: DivLog: Log Parsing with Prompt Enhanced In-Context Learning. In Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, pp. 1–12 (2024)
Yang, L., Chen, J., Wang, Z., Wang, W., Jiang, J., Dong, X., Zhang, W.: Semi-supervised log-based anomaly detection via probabilistic label estimation. In 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pp. 1448–1460. IEEE (2021)
Yildirim, Ö.: A novel wavelet sequence based on deep bidirectional lstm network model for ecg signal classification. Comput. Biol. Med. **96**, 189–202 (2018)
Ying, S., Wang, B., Wang, L., Li, Q., Zhao, Y., Shang, J., Huang, H., Cheng, G., Yang, Z., Geng, J.: An improved knn-based efficient log anomaly detection method with automatically labeled samples. ACM Trans. Knowl. Discov. Data (TKDD) **15**(3), 1–22 (2021)

Yu, B., Yao, J., Fu, Q., Zhong, Z., Xie, H., Wu, Y., Ma, Y., He, P.: Deep Learning or Classical Machine Learning? An Empirical Study on Log-Based Anomaly Detection, in 2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE), pp. 392–404. IEEE Computer Society (2023)
Yu, B., Yao, J., Fu, Q., Zhong, Z., Xie, H., Wu, Y., Ma, Y., He, P.: 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, pp. 1–13. (2024)
Zeufack, V., Kim, D., Seo, D., Lee, A.: An unsupervised anomaly detection framework for detecting anomalies in real time through network system's log files analysis. High-Confidence Comput. **1**(2), 100030 (2021)
Zhang, L., Xie, X., Xie, K., Wang, Z., Lu, Y., Zhang, Y.: An efficient log parsing algorithm based on heuristic rules. In Advanced Parallel Processing Technologies: 13th International Symposium, APPT 2019, Tianjin, China, August 15–16, 2019, Proceedings 13, pp. 123–134. Springer (2019)
Zhang, X., Xu, Y., Lin, Q., Qiao, B., Zhang, H., Dang, Y., Xie, C., Yang, X., Cheng, Q., Li, Z., et al.: 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, pp. 807–817. (2019)
Zhang, B., Zhang, H., Le, V.H., Moscato, P., Zhang, A.: Semi-supervised and unsupervised anomaly detection by mining numerical workflow relations from system logs. Autom. Softw. Eng. **30**(1), 4 (2023)
Zhu, J., He, P., Fu, Q., Zhang, H., Lyu, M.R., Zhang, D.: Learning to log: Helping developers make informed logging decisions. In 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol. 1, pp. 415–425. IEEE (2015)
Zhu, J., He, S., He, P., Liu, J., Lyu, M.R.: Loghub: A large collection of system log datasets for ai-driven log analytics. In 2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE), pp. 355–366. IEEE (2023)
Zhu, J., He, S., Liu, J., He, P., Xie, Q., Zheng, Z., Lyu, M.R.: Tools and benchmarks for automated log parsing. In 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), pp. 121–130. IEEE (2019)

Authors and Affiliations

Yicheng Sun[1] · Jacky Keung[1] · Zhen Yang[2] · Shuo Liu[1] · Hi Kuen Yu[1]

✉ Zhen Yang
zhenyang@sdu.edu.cn

Yicheng Sun
yicsun2-c@my.cityu.edu.hk

Jacky Keung
Jacky.Keung@cityu.edu.hk

Shuo Liu
sliu273-c@my.cityu.edu.hk

Hi Kuen Yu
hikuenyu2-c@my.cityu.edu.hk

[1] Department of Computer Science, City University of Hong Kong, Hong Kong, Hong Kong

[2] School of Computer Science and Technology, Shandong University, Qingdao, China