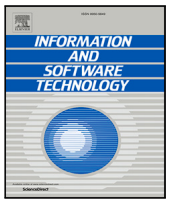
Information and Software Technology 183 (2025) 107743

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof



SemiRALD: A semi-supervised hybrid language model for robust Anomalous Log Detection

Yicheng Sun [a], Jacky Wai Keung [a], Zhen Yang [b,*], Shuo Liu [a], Hi Kuen Yu [a]

[a] Department of Computer Science, City University of Hong Kong, Hong Kong, China
[b] School of Computer Science and Technology, Shandong University, Qingdao, China

ARTICLE INFO

Dataset link: https://github.com/log-detection/SemiRALD

Keywords:
Anomaly log detection
Software reliability
Log parsing
RoBERTa
Bi-LSTM
Semi-supervised learning

ABSTRACT

Context: Deep learning-based Anomalous Log Detection (DALD) tools are critical for software reliability, but current approaches face challenges, including information loss during log parsing, reliance on large labeled datasets, and fragility in low-resource scenarios.
Objective: To overcome the above limitations, we propose SemiRALD, a semi-supervised learning-based robust ALD approach that leverages Large Language Model (LLM) for log parsing, enhancing both flexibility and accuracy. It utilizes a hybrid language model to repeatedly fit the samples with generate pseudo-labels, thereby training DALD models with limited resources and facilitating efficient anomaly detection tasks.
Method: In detail, SemiRALD utilizes ChatGPT and in-context learning for automated log parsing, thereby improving the log integrity during log parsing. Subsequently, it harnesses a semi-supervised learning framework and our proposed hybrid language model to remedy the performance degeneration caused by low-resource restriction in practice. Semi-supervised learning requires only a small amount of labeled data throughout the entire process, while the hybrid language model is built on the architecture of RoBERTa and an attention-based BiLSTM.
Results: Experiments on the HDFS and BGL datasets demonstrate that SemiRALD achieves an average F1-score improvement of 7.3% and 8.2%, respectively, over seven benchmark models. On small-scale datasets (0.1% of the original size), SemiRALD outperforms competitors by 31.4% and 46.0% in F1-score, respectively. Its consistent performance across diverse datasets highlights its generalizability and robustness.
Conclusion: SemiRALD is capable of handling anomaly detection tasks in both large-scale and low-resource datasets, delivering significant advancements in anomaly log detection and offering robust, adaptable solutions to address prevalent challenges in the field of software reliability engineering.

1. Introduction

Modern software systems are highly interconnected, generating vast quantities of log data during operation [1,2]. These logs are critical for monitoring system health, diagnosing failures, and ensuring software reliability. However, identifying anomalies in logs remains a significant challenge due to information loss in log parsing, the scarcity of labeled training data, and the fragility of existing methods in low-resource scenarios [3]. To facilitate troubleshooting, interconnected systems generate logs that aid in identifying and resolving problems [4]. Real-time recording of operational logs within these systems is crucial for ensuring their reliability and high availability [5]. This practice enables developers to assess software stability, monitor system status, and track significant events [6].

Anomalous Log Detection (ALD) tools aim to identify abnormal behavior by analyzing operational logs and enabling timely actions [7–12]. Over the past few years, numerous ALD methods have been proposed, including detection methods of rule-based [13] and deep learning-based [14–16]. Owing to the learning-based characteristic, the latter offers a high degree of flexibility and dynamic approaches across various application scenarios, gradually dominating this field. While previous anomaly log detection models have demonstrated outstanding performance on public datasets [16–18], there are still several limitations and challenges that current research encounters:

Information loss induced by current log parsers. Log parsers are crucial modules for current deep learning-based ALD tools, which extract essential information for analysis. However, most parsers are developed based on manually crafted rules, leading to their restricted

* Corresponding author.
E-mail addresses: yicsun2-c@my.cityu.edu.hk (Y. Sun), Jacky.Keung@cityu.edu.hk (J.W. Keung), zhenyang@sdu.edu.cn (Z. Yang), sliu273-c@my.cityu.edu.hk (S. Liu), hikuenyu2-c@my.cityu.edu.hk (H.K. Yu).

https://doi.org/10.1016/j.infsof.2025.107743
Received 14 July 2024; Received in revised form 1 April 2025; Accepted 1 April 2025
Available online 11 April 2025

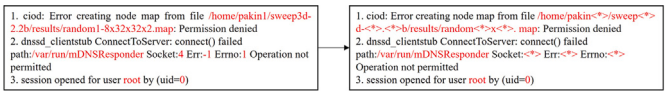


Fig. 1. Examples concerning the parsing error of rule-based Drain parser.

generality to various log formats in the wild and hardly ensuring information integrity after parsing [7]. As shown in Fig. 1, the most commonly used Drain parser mistakenly converts "1" in sub-folder "pakin1" to "<*>" in the first example, as Drain deems "1" here as a digit. Similar examples frequently occur (refer to Fig. 1) because parsing rules of Drain are fixed, but different companies tend to define log formats in diverse manners, posing challenges to the scalability of log examples.

Lack of supervised training examples. Current learning-based anomaly detection approaches often necessitate a large amount of labeled normal or anomalous log data for training. However, supervised training examples are scarce in practice, leading to the unavailability of these models. Typically, supervised methods generally outperform others due to the vast amount of labeled data (e.g., normal or abnormal) used during training. However, these methods are not ideal for industrial applications because they require extensive labeled datasets and are highly sensitive to incorrect labels [19]. This procedure often consumes significant human resources and may lead to labeling errors, which can degrade performance [20]. This makes the effective deployment of supervised methods in industry quite difficult. In contrast, semi-supervised methods require only a small amount of labeled data for training, significantly reducing the cost of manual labeling and dependency on labeled data. Unlike unsupervised methods, semi-supervised approaches utilize some training label information (e.g., feature-label correlations), enabling them to optimize the training objective more effectively [21]. Hence, effectively learning the distinctive features of anomalous logs and making accurate predictions with limited labeled examples presents the second main challenge.

Presenting fragility in low-resource scenarios. As mentioned above, available training examples are limited in practice, especially for some domain-specific scenarios, such as log data in chemical and material engineering. However, although numerous ALD models excel with massive training examples as reported in their respective papers [11,16–18], our experiments reveal their subpar performance on small-scale datasets (e.g., 0.1%, 0.5%, 1%, 5%, 10%, 15% of the full dataset), demonstrating a fragile characteristic. Therefore, enhancing ALD models' robustness in anomalous log detection is the third challenge to be resolved in this paper.

To tackle these challenges, we present a novel log anomaly detection model, **SemiRALD**, a **Semi**-supervised learning-based **R**obust **A**nomalous **L**og **D**etection approach. To enhance the generalization ability to parse various log sequences, we employ ChatGPT for automated log parsing. We have devised prompt templates for log sequences to direct ChatGPT in comprehending log information and parsing logs. We assessed ChatGPT's performance in zero-shot and few-shot scenarios by employing various prompts to investigate its potential as an innovative log parsing approach.

For anomaly detection, SemiRALD introduces a semi-supervised learning framework and leverages our proposed hybrid language model to address performance degradation in practice due to resource constraints. Semi-supervised learning requires only a small amount of labeled data throughout the entire process, while the hybrid language model is built on the architecture of RoBERTa and an attention-based BiLSTM. RoBERTa, a robust large-scale pre-trained language model, employs the encoder section of the Transformer model to effectively capture semantic features during pre-training, demonstrating remarkable adaptability even with small-scale datasets. It encodes the log text, generates context-aware representations, and inputs them into an attention-based Bi-LSTM. The attention mechanism layer highlights crucial information in the log text by assigning varying weights to different parts of the context, while the Bi-LSTM adjusts its focus based on different segments of the log sequence, leveraging both forward and backward information to capture sequential features. By integrating RoBERTa and attention-based Bi-LSTM as a hybrid language model, SemiRALD thoroughly understands log text and captures semantic and contextual information at different levels, thereby enhancing log anomaly detection. Experiments on two popular datasets demonstrate that SemiRALD outperforms seven benchmark models, with average improvements by 7.3% and 8.2% in terms of F1-score, respectively. Particularly on small-scale datasets, which are only 0.1% of the original size, SemiRALD achieves average F1-score improvements of 31.4% and 46.0% compared to the remaining benchmark models. Besides, its consistent and superior performance across the three low-resource scenario datasets also highlights its generalizability. In summary, our primary contributions are as follows:

- We propose the groundbreaking SemiRALD model, which employs semi-supervised learning for model training, requiring only a small amount of labeled data in practice. SemiRALD integrates RoBERTa and an attention-based Bi-LSTM as a hybrid language model. This architecture provides good robustness and enhances the accuracy of log sequence processing.
- To our knowledge, we are the first to utilize ChatGPT for log parsing in anomaly detection and to systematically assess its effectiveness. We crafted suitable prompts for ChatGPT to guide its comprehension and execution of log parsing tasks. Our experiments highlight its superiority to rule-based parsers in preserving log integrity, thereby enhancing the performance of ALD.
- Extensive experiments are conducted to evaluate the effectiveness of SemiRALD across various volume settings of available supervised training examples, demonstrating its superiority against state-of-the-art ALD models, especially in low-resource scenarios.

The remainder of this paper is organized as follows. Section 2 presents the background of anomaly detection. Section 3 elaborates on the proposed approach, covering log parsing using ChatGPT, the hybrid language model, and semi-supervised learning with pseudo-labels. Sections 4 and 5 discuss the experimental design and results. Section 6 addresses the threats to validity. Section 7 reviews related work. Finally, Section 8 summarizes the paper.

2. Background

2.1. Log data

In intricate interconnected systems, substantial log data is often generated during operation. Each log entry documents specific system events and may include fields such as timestamps, severity levels, and message content [22]. A typical log message consists of two parts: (1) **Event template**: describing system events using constant strings or keywords and representing the static aspect of the log message; and (2) **Parameters**: reflecting the state of the particular software runtime and constituting the dynamic aspect of the log message.

The Loghub [23] platform has made a substantial amount of system logs available to researchers, serving as the primary data source. For instance, the HDFS [24] log file was collected from over 200 Amazon EC2 nodes, while the BGL [25] log file was gathered from the Blue Gene/L supercomputer system.

2.2. Log parsing

Log parsing automatically transforms each log message into a structured format by identifying event templates (constant components) and parameters (variable components). For example, a log message

such as "*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*" can be parsed into the event template "*'POST <*>' 200 OK delete empty directory: <*>*" with parameter values [*/webhook/order HTTP/1.0*, */backend/upload/download-/orders/49512/49512*], where "<*>" marks the variable components of the log.

Prior research [7,26] has shown that the effectiveness of log parsing significantly affects the performance of subsequent anomaly detection models. As such, log parsing is a critical step in log analysis, with its accuracy directly influencing the performance of downstream tasks [18,27]. Numerous studies have proposed various log parsing methods [7,28–30], which can be broadly categorized into approaches based on frequent pattern mining [31,32], clustering [33,34], and heuristics [28,35,36].

Among these, heuristic methods, which exploit the inherent characteristics of logs, have demonstrated superior performance in terms of both accuracy [7] and time efficiency [12] compared to other techniques. The most common log parsing method is heuristic-based [37], which extracts log templates according to predefined rules. For example, Drain [28], a heuristic log parsing method that utilizes a fixed-depth parsing tree to encode specially designed parsing rules, is one of the most widely used parsers in log processing.

Recently, the rise of Large Language Models (LLMs), such as ChatGPT [38], has led to their successful application in various software engineering tasks [39], showing promising performance [40]. This has opened up new possibilities for log parsing using LLMs, though further research is needed to fully evaluate their capabilities in this domain.

2.3. RoBERTa

BERT [41] is a model consisting of Transformer encoders, distinguished by its foundation in unsupervised learning from large-scale textual data. The Key innovations of BERT include introducing 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 [42]. BERT's pre-training employs two types of unsupervised learning methods: Masked Language Model (MLM) and Next Sentence Prediction (NSP) [43]. 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 [44], a robustly optimized BERT [41] 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.

2.4. Bi-LSTM

The LSTM network [45], 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 [46]. 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 [47]: 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. The process by which the forget gate determines which information will be retained in the current neural unit can be represented as:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{1}$$

The input gate uses the following equation to determine whether to discard or retain information about the data:

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \\ C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \end{aligned} \tag{2}$$

The output gate determines the output value of the LSTM unit based on the cell state and can be represented as:

$$\begin{aligned} o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t &= o_t * \tanh(C_t) \end{aligned} \tag{3}$$

The weights for input information x are denoted by W_f, W_i, W_c and W_o, while U_i, U_f and U_o represent the weights for the previous moment $h_{(t-1)}$. The biases are represented by b_i, b_f and b_o, and t signifies time. σ is a sigmoid function ranges from 0 to 1.

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 [48] 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 [49]. The 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 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. The update parameters for Bi-LSTM can be represented as:

$$\begin{aligned} L_f &= \sigma(W_1 \cdot x_t + W_2 \cdot L_{f-1} + b_{L_f}) \\ L_b &= \sigma(W_3 \cdot x_t + W_5 \cdot L_{b-1} + b_{L_b}) \\ x_o &= W_4 \cdot L_f + W_6 \cdot L_b + b_{x_o} \end{aligned} \tag{4}$$

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

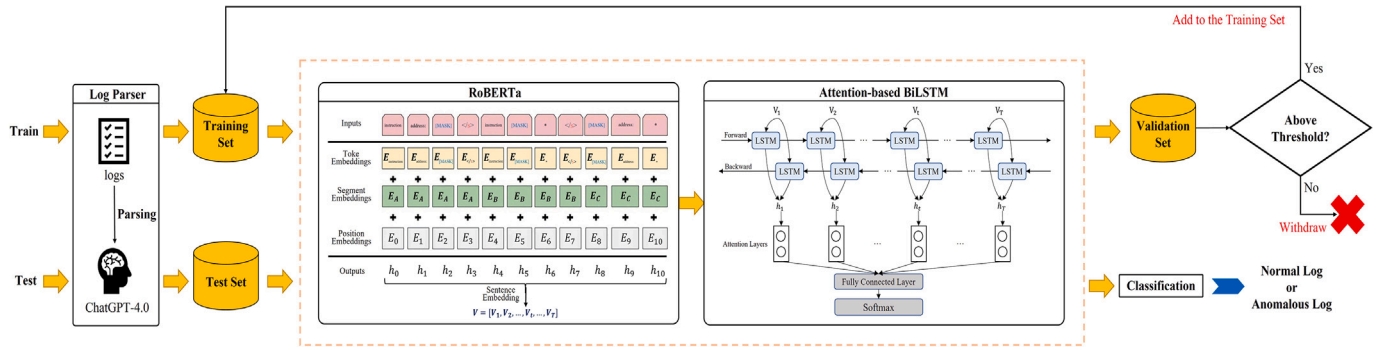


Fig. 2. The overview of SemiRALD.

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. We introduce an innovative log detection model named SemiRALD, as depicted in Fig. 2. At the outset, SemiRALD employs RoBERTa to encode log text and generate context-related representations. These representations are then fed into an attention-based Bi-LSTM. The attention mechanism emphasizes critical information in the log text by assigning different weights to various parts of the context, while the Bi-LSTM adjusts its focus based on different segments of the log sequence, capturing sequential features through the fusion of forward and backward information. By integrating the strengths of each module, SemiRALD effectively identifies anomalous logs.

3.1. Log parsing based on ChatGPT

To accommodate the varied log formats across different software systems, we begin by crafting prompts for the log sequences, incorporating in-context learning. We utilize the ChatGPT API to process these logs, which enables automatic identification and parsing of log entries. We devised a comprehensive prompt template for ChatGPT, outlined in Fig. 3, comprising the following components: (1) Task Description, (2) Extraction and Replacement Rules, (3) Situated Learning, and (4) Target Log. Each section was tailored based on the specific requirements of log sequences, and ChatGPT was employed to refine the prompts, ensuring adaptability to diverse log file formats. Through iterative enhancements, we evaluated ChatGPT-4.0's performance to identify the most effective prompting strategies. Ultimately, we selected two strategies: PT-1, a straightforward prompt, and PT-2, a detailed prompt.

Task Description: This section outlines the log parsing task assigned to ChatGPT, encompassing an introduction, objectives, and anticipated outcomes. Initially, we elucidated the log parsing task to ChatGPT, delineating the expected results. To accomplish this, we employed two prevalent types of ChatGPT instructions: indirect (CoT prompts) and direct instructions. Indirect instructions included phrases like "extract the body of the log" or "process the log sequence" to convey our objectives. Direct instructions were more precise, such as "output the result directly, without explanation".

Extraction and Replacement Rules: This section elucidates the methodology for extracting and processing crucial information from the log text using ChatGPT. We delineated specific guidelines for replacing

Prompt Template	<Task Description> \n <Extraction and Replacement Rules> \n <Situated Learning> \n <Target Log> \n
Prompt Template - Simple (PT-1)	You need to parse the Content of the logs and replace the dynamic variables with templates. Please output the results directly without explanation. Here are some examples: (optional) - Original Log: *{placeholder}* - Replaced Log: *{placeholder}* Please parse the log template from this log message: *{placeholder}*
Prompt Template - Detailed (PT-2)	You need to parse the Content of the logs and replace the dynamic variables with templates. Please output the results directly without explanation. You can refer to these extraction and replacement rules: *{placeholder}* Here are some examples: (optional) - Original Log: *{placeholder}* - Replaced Log: *{placeholder}* Please parse the log template from this log message: *{placeholder}*

Fig. 3. The log parsing prompt template of ChatGPT.

or transforming certain text to align with log parsing requirements. For instance, in PT-2, we stipulated rules for substituting file addresses with "ADDR", IP addresses with "IP", numerical values (e.g., port numbers) with "NUM", dates with "TIME", and usernames with "USER". This approach aids ChatGPT in comprehending the various types of variables present in the log sequences.

Situated Learning: Also recognized as in-context learning, this approach underscores the customization of prior knowledge to optimize ChatGPT's practical usefulness and effectiveness in log text processing. Prior studies have indicated that ChatGPT's performance suffers in the absence of prior knowledge. Hence, presenting examples of original and processed log text can enhance ChatGPT's performance in log anomaly detection. This is referred to as "few-shot" prompting, while the absence of such examples constitutes "zero-shot" prompting.

Target Log: This section involves the input of raw log data into ChatGPT. We feed each log text iteratively into the system, and ChatGPT processes them individually, generating output log text data for use in SemiRALD's anomaly detection experiments. As ChatGPT imposes a maximum length limit on the output tokens for each request, any requests exceeding this limit will result in the termination of the response. Although in our actual experiments, no log text has exceeded the maximum display length, we have implemented additional measures within the program to address potential future log texts that may surpass the maximum length limit. In such instances, ChatGPT will skip processing these log entries and instead provide a log number.

3.2. Anomaly detection model building

3.2.1. Hybrid language model

To learn the representations of log samples, we propose a hybrid language model that first leverages RoBERTa to understand the information within the log sequences. RoBERTa employs dynamic masking, generating a new masking pattern each time a log sequence is input into the model. As depicted in Fig. 2, 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:

$$X = W = T + S + P \tag{5}$$

Word embeddings produce a vector representation h for each token, and these h vectors can be integrated into a sentence-level semantic vector V. 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:

$$V = [V_1, V_2, \ldots, V_t, \ldots, V_T] \tag{6}$$

SemiRALD 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:

$$\begin{aligned} multihead &= \text{concat}(head_1, head_2, \ldots, head_n) \cdot W^O \\ head_n &= \text{Softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \cdot V \end{aligned} \tag{7}$$

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.

3.2.2. Semi-supervised learning with pseudo labels

In log anomaly detection, anomalous logs typically constitute a small portion of the dataset. For example, in the HDFS dataset, only 2.93% of the logs are anomalous, making it challenging to obtain enough labeled anomalous logs for training. Semi-supervised learning, which requires only a small amount of labeled data, has gained prominence for outperforming unsupervised learning. However, unlike other semi-supervised approaches that train solely on normal logs, which have notable shortcomings. First, normal logs typically make up the majority of log data, necessitating a large amount of labeled data to effectively train the model. Second, abnormal logs exhibit much greater variability in format and structure, making it difficult for the model to generalize without learning these diverse patterns. Skipping the learning of such variations in abnormal logs can lead to reduced performance and limited detection capability, and is therefore discouraged.

Our approach addresses these challenges. SemiRALD employs pseudo-labeled semi-supervised learning, requiring only a small amount of labeled data. As shown in Fig. 4, semi-supervised training of SemiRALD involves four steps: (1) model initialization, (2) pseudo-label generation, (3) iterative model training, and (4) test set evaluation.

Model Initialization: We selected a very small portion of labeled logs. Considering the massive quantity of logs recording software anomalies (e.g., Alibaba's cloud computing system generates approximately 30–50 GB or approximately 120–200 million lines of logs per hour), we chose 1‰ of the logs from the dataset to train the initial model, constituting the training set with an equal distribution of normal and abnormal logs. This approach offers several advantages: (1) it simplifies the laborious process of manually labeling logs by engineers, and (2) it avoids class imbalance issues. In the course of model training, we utilized the cross-entropy loss function to gauge the difference between the model's predicted outcomes and the actual labels. The Adam optimization algorithm was employed to iteratively adjust the model parameters, which involved updating the weights within the Bi-LSTM and Attention layers.

Pseudo-label Generation: Utilizing the model trained in the initialization stage, we predicted the classes of the unlabeled log data, resulting in the probability distribution of each log sample's class. Employing a threshold of 0.9, we fed the log data from the unlabeled log data into RoBERTa and Bi-LSTM models, obtaining corresponding predicted class probabilities (normal or abnormal). Samples exceeding the threshold in predicted class probabilities were deemed pseudo-labels, while those below the threshold were discarded. Subsequently, samples with pseudo-labels were retrained alongside the labeled logs for model retraining. We combined the labeled abnormal samples (i.e., the original training set) with the pseudo-labeled data (i.e., samples from the unlabeled log data that exceed the threshold) to create a new augmented training set. This process progressively enlarges the training set while reducing the size of the unlabeled log data.

Model Iterative Training: By integrating the identified high-confidence pseudo-labeled log data with the original labeled log data and utilizing the augmented training set for additional training, SemiRALD continuously adjusted the samples in the unlabeled log data. In each iteration, a new log sample's class probability was determined based on the threshold, replacing the previous log class label in the unlabeled log data. We applied the same optimization algorithm and loss function used during training with labeled logs. With each iteration, the augmented training set was reused for model training, and more high-quality pseudo-labeled log data were progressively introduced until a predefined stopping condition for iteration was reached.

Test Set Evaluation: Following the iterations, we assessed the performance of SemiRALD using a separate test set. This test set was distinct from the data used during training or validation, offering authentic insights into the model's generalization capability. We derived the model's ultimate performance metrics, encompassing accuracy, recall, and F1-score. This phase was pivotal in validating the ultimate practicality of SemiRALD, affirming its efficacy and resilience in real-world scenarios.

4. Experimental setup

4.1. Datasets

We employ two public datasets to evaluate the performance of our algorithm: the HDFS [24] dataset and the BGL [25] dataset. These

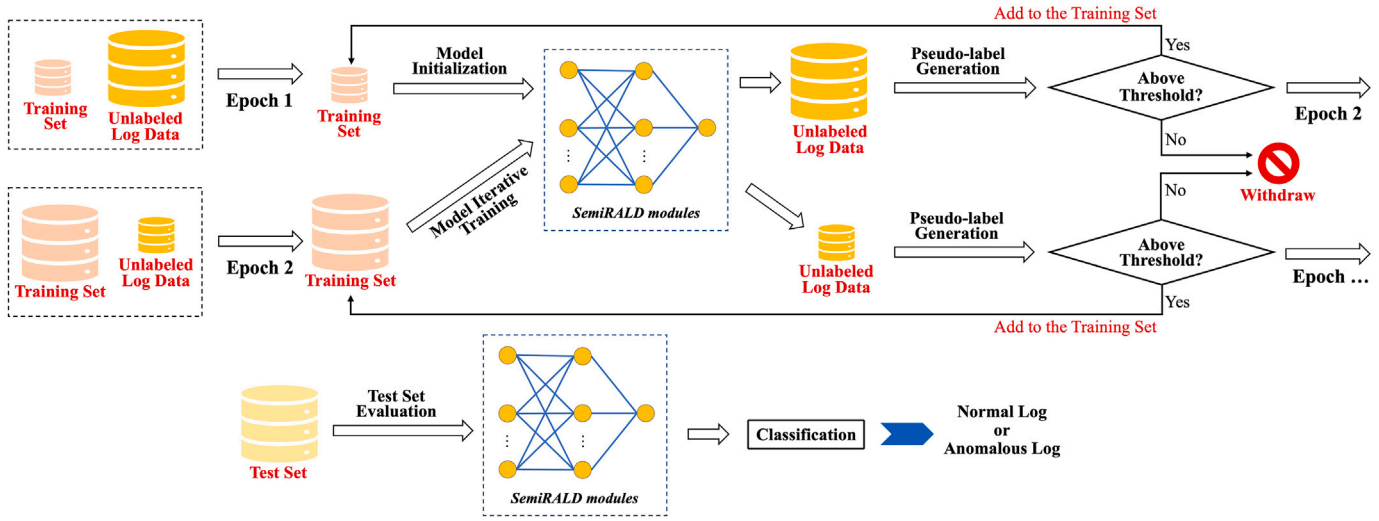


Fig. 4. The semi-supervised training process of SemiRALD.

Table 1
The number of log messages in each dataset.

Dataset	Category	Numbers	Anomalies
HDFS	Distributed file system	11,175,629 (575,061 blocks)	16,838 blocks (2.93%)
BGL	Supercomputer	4,747,963	348,469 (7.34%)
ILMS	Low-resource	54,637	3129 (5.73%)

datasets are labeled with log types and exhibit substantial differences in log formats, allowing us to assess the model's generalization performance. Additionally, we conduct experiments on one low-resource log dataset, the ILMS dataset, which is constructed from our own software. We allocated 80% of the data for training and 20% for testing. The specific distribution of logs is outlined in Table 1.

HDFS dataset is 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 are segmented into 575,061 blocks, forming log message sequences, each of which is 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.

ILMS dataset is from our own development of an intelligent laboratory management system (ILMS) for managing experiment workflows in the chemical and materials fields, specifically designed for low-resource scenarios due to the limited log data available. This system is built using the Java programming language and utilizes Apache Log4j for comprehensive logging capabilities. Log4j enables us to log events in a structured JSON format, improving readability and facilitating automated processing. This format includes essential metadata such as timestamps, log levels, and custom context variables like user IDs and session details (as outlined in the Log4j User's Guide.[1]) Table 2 provides examples of logs at different levels. We continuously collect system logs for 50 h, forming a dataset with a total of 54,637 log entries, referred to as the ILMS dataset. This dataset represents a low-resource scenario, as its log volume is significantly lower compared to interconnected systems. It also features more complex and longer log sequences. According to engineering statistics, the dataset includes 371 distinct log formats. In this dataset, "Info" and "Warning" represent normal logs, while "Error" indicates abnormal logs. Of these, 3,129 (5.73%) log messages are flagged as abnormal.

4.2. Model parameters

To implement SemiRALD, we adopt the version of gpt-4-turbo as the log parser and invoke their API[2] provided by OpenAI. Regarding the model structure and hyper-parameters of SemiRALD, we configure them as below: 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 30 522. 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.

We allocated 80% of the data for training, which was used in the semi-supervised learning process to generate pseudo labels, and 20% for testing. 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, 64 GB 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: What is the performance of SemiRALD against existing DALD approaches?**
  In this study, we compared SemiRALD with seven benchmark models, including an unsupervised learning-based model (DeepLog), supervised learning-based models (HitAnomaly, NeuralLog, LogRobust), a semi-supervised learning-based model (PLELog), and BERT-based models (LAnoBERT and LogBERT). Given the large volume of logs in the HDFS and BGL datasets, we trained our model using only 0.1% of labeled logs. Detailed descriptions of these benchmark models can be found in Section 4.4.

[1] https://logging.apache.org/log4j/2.x/manual/index.html.

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

Table 2
Examples of logs at different levels in our software system.

Info

```
{
  "timestamp": "2023-05-19T10:15:30.123Z",
  "level": "INFO",
  "message": "User 'john.doe' logged in successfully",
  "context": {
    "userId": "**********",
    "ipAddress": "**.***.***.***",
    "sessionId": "**********"
  }
}
```

Warning

```
{
  "timestamp": "2023-05-19T12:30:15.678Z",
  "level": "WARN",
  "message": "Failed to connect to database server",
  "context": {
    "error": "Connection timeout",
    "attemptNumber": 3,
    "lastAttemptTime": "2023-05-19T12:30:00Z"
  }
}
```

Error

```
{
  "timestamp": "2023-05-19T15:20:45.999Z",
  "level": "ERROR",
  "message": "NullPointerException occurred in method 'processOrder'",
  "context": {
    "class": "OrderProcessor",
    "method": "processOrder",
    "exceptionDetails": "java.lang.NullPointerException: Invalid null value"
  }
}
```

Note: Certain parts containing user information are replaced with asterisks (*). The log entry at Error level indicates a *NullPointerException* occurred in the *processOrder* method of the *OrderProcessor* class, attributed to an invalid null value.

it is noteworthy that the experimental outcomes of DeepLog are sourced from the LogBERT article, while the rest are from their respective original research.

- **RQ2: What is the contribution of each individual module within SemiRALD?**
  In this study, we disassembled each module within SemiRALD to evaluate their contribution. RoBERTa is treated as the first module, Bi-LSTM as the second, and the attention mechanism as the third. Since the BGL dataset is more intricate than the HDFS dataset, the differences, in effect, are expected to be more pronounced. Therefore, we selected BGL datasets of varying scales for comparison.
- **RQ3: What is the performance of ChatGPT in log parsing?**
  In this study, we aim to evaluate whether using ChatGPT as the log parser can achieve more information integrity when parsing logs and whether such integrity contributes to the performance of the ALD task.
  First of all, we extensively evaluate the performance of ChatGPT in log parsing with zero-shot and few-shot settings using the HDFS and BGL datasets, where two versions of ChatGPT are involved, i.e., using gpt-4.0-turbo for ChatGPT-4.0 and gpt-3.5-turbo for ChatGPT-3.5. Zero-shot denotes scenarios without providing examples for in-context learning. Few-shot includes 1-shot, 3-shot, and 5-shot configurations in this RQ. Besides, two sets of prompts, including PT-1 and PT-2, are utilized, where PT-1 lacks "extraction and replacement rules". as mentioned in Section 3.1. Fig. 5 is an example of "PT-2 one-shot", where multiple examples constitute a few-shot scenario. For the 1-shot scenario, the most common log messages are selected. For the 3-shot and 5-shot scenarios, log messages with more variable information are chosen. To assess the information integrity after parsing, we adopt Parsing Accuracy (PA) introduced in Section 4.5 as the evaluation metric.
  Secondly, we also compare the effectiveness of the ChatGPT-based parser with that of the most widely used rule-based parser in log parsing, thereby exploring whether the former can contribute to the ALD performance via maintaining information integrity when parsing.
- **RQ4: How does SemiRALD perform in low-resource scenarios against existing DALD approaches?**
  Low-resource scenarios refer to situations where the volume of available data is limited, often due to constraints such as insufficient data generation, restricted access to large-scale datasets, or resource limitations in system logs. These scenarios are commonly found in cases like small-scale industrial software, newly developed systems with a limited user base, experimental applications, or environments where resource constraints prevent large-scale logging. In these cases, models face challenges in training due to the insufficient quantity of labeled or representative data, which hinders the generalization and effectiveness of anomaly detection systems.
  In order to construct a small-scale dataset, we used the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [50] algorithm to identify and select representative samples from the full dataset. Specifically, DBSCAN was employed to cluster the data points based on their density and proximity, ensuring that the selected samples captured the inherent structure of the dataset while minimizing noise. Once the clusters were formed, we reduced the dataset by selecting a proportionate percentage of data from each cluster, based on its size relative to the entire dataset. This approach ensured that we retained a balanced representation of both the large volume of normal logs with diverse formats and the smaller, less frequent abnormal logs. By doing so, we maintained both diversity and relevance, making the dataset suitable for low-resource settings.
  To systematically evaluate performance on low-resource scenarios, we selected varying scales of HDFS and BGL datasets, ranging from 0.1% to 15%. Experiments were conducted at 0.1%, 0.5%, 1%, 5%, 10%, and 15%. Compared to the HDFS dataset, the BGL dataset is larger, with longer log sequences and a more complex structure, rendering log anomaly detection more challenging. Hence, the BGL dataset is better suited for assessing model stability. Additionally, we conducted experiments on the low-resource ILMS dataset, which we constructed from a real-world scenario. Due to the limited volume of our system logs (only 54,637 entries), we experimented using 10%, 50%, and 100% of the data.
  To ensure a comprehensive comparison, we selected representative models from different learning paradigms: DeepLog (unsupervised), PLELog (semi-supervised), and LogRobust (supervised), and compared them with SemiRALD. Notably, in low-resource datasets, the dataset size becomes significantly smaller. If only 0.1% of the logs are labeled, the number of labeled samples may be insufficient for effective model learning. Therefore, we adopted a unified selection strategy: if 0.1% of the labeled logs exceeds

```
You need to parse the Content of the logs and replace the dynamic variables with templates. Please output the results directly without explanation.
You can refer to these extraction and replacement rules:
  - Time Replacement: Replace all timestamps and specific times with "TIME".
  - IP Address Replacement: Replace IP addresses with "IP".
  - File Path Replacement: Replace file paths with "ADDR".
  - Number Replacement: Replace all numbers in log entries with "NUM".
  - User Replacement: Replace all usernames with "USER".
Here are some examples:
  - Original Log: 1134724900 2005.12.16 R43-M1-NC-I:J18-U01 2005-12-16-01.21.40.588712 R43-M1-NC-I:J18-U01 RAS APP FATAL ciod:
Error reading message prefix on CioStream socket to 172.16.96.116:39416, Link has been severed
  - Replaced Log: ciod: Error reading message prefix on CioStream socket to "IP", Link has been severed
Please parse the log template from this log message:
```

Fig. 5. The example of "PT-2 one-shot".

500 entries, we follow the exact selection ratio. Otherwise, we set a minimum threshold of 500 labeled logs. For instance, in the 0.1% scaled BGL dataset, where the number of labeled logs is fewer than 500, we default to selecting 500 labeled logs for training. Conversely, in the 15% scaled BGL dataset, where the number of labeled logs is 712 (greater than 500), we use all 712 logs for training.

- **RQ5: How does SemiRALD perform in terms of efficiency?**
  In this study, we selected HDFS and BGL datasets to investigate the efficiency advantages of SemiRALD. These two datasets were chosen due to their substantial size, which provides a comprehensive evaluation of the time costs associated with log parsing and anomaly detection.
  For the log parsing task, we compared our ChatGPT-based parser with two widely-used parsers, Drain and AEL, which demonstrated the best performance in RQ3. We evaluated the total time required for each parser to completely parse both datasets. For the ChatGPT-based parser, we designed the prompt Target Log to input 800 log sequences at once, allowing the model to parse and return results simultaneously. This parallel processing approach was implemented to maximize efficiency and minimize time costs. For anomaly detection, we selected representative models from different learning paradigms to ensure a comprehensive comparison: DeepLog (unsupervised), PLELog (semi-supervised), and LogRobust (supervised). These models were compared with SemiRALD in terms of both training and testing time. All experiments were conducted on our in-house system, ensuring a controlled environment to assess the computational costs and performance of each model.

4.4. Benchmark models

In this section, we present benchmark models for comparison with SemiRALD 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 [29] 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 [51] 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 [14] 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.

PLELog [20,52] 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.

LAnoBERT [53] is an anomaly detection model based on the BERT architecture, specifically engineered to detect system log anomalies through masked language modeling. By circumventing conventional log parsing methods, LAnoBERT mitigates data loss and preserves the integrity of original log sequences, thereby augmenting detection accuracy. It has demonstrated superior performance on benchmark datasets like HDFS, BGL, and Thunderbird, exhibiting favorable comparisons against both supervised and unsupervised models.

LogRobust [16] 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.

LogBERT [15] 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

In line with [54], we utilize parsing accuracy (PA) as an evaluation metric for the comparative experiments on log parsing. PA metric is the ratio of correctly parsed log messages over the total number of log messages. PA quantifies the effectiveness of automated log parsing and is defined as the ratio of correctly parsed log messages to the total log messages. Post log parsing, each log message is linked with an event template, and a log message is deemed correctly parsed only if its event template matches the actual template.

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.

$$Precision = \frac{TP}{TP + FP} \tag{8}$$

Table 3
The performance of different models on two datasets.

Model	HDFS			BGL		
	P	R	F1	P	R	F1
Unsupervised learning-based						
DeepLog	0.884	0.695	0.773	0.897	0.828	0.861
Supervised learning-based						
HitAnomaly	**1**	0.970	0.980	0.950	0.900	0.920
NeuralLog	0.960	**1**	0.980	0.980	0.980	0.980
LogRobust	0.980	0.960	0.970	0.910	0.770	0.830
Semi-supervised learning-based						
PLELog	0.950	0.963	0.957	0.965	**0.999**	0.982
LAnoBERT	–	–	0.964	–	–	0.875
LogBERT	0.870	0.781	0.823	0.894	0.923	0.908
SemiRALD	0.994	0.991	**0.993**	**0.991**	0.989	**0.990**

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.

$$Recall = \frac{TP}{TP + FN} \tag{9}$$

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.

$$F1 - score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{10}$$

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.

5. Results and analysis

5.1. Effectiveness of SemiRALD (RQ1)

In this section, we conducted a comparison between SemiRALD and seven benchmark models, as outlined in Table 3. 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.

In detail, HitAnomaly, LogRobust, and LAnoBERT demonstrate notably higher performance on the HDFS dataset compared to their performance on the BGL dataset. Conversely, Deeplog, PLELog, and LogBERT exhibit the opposite pattern. NeuralLog, being a supervised model, performs comparably to SemiRALD on both datasets and surpasses other models. Owing to the log format discrepancy between the above two datasets, leading to the log parsers of each model can hardly be generalized to them concurrently. Thus, most ALD models cannot perform consistently well on both datasets above. In contrast, SemiRALD adopts ChatGPT as the log parser with our carefully designed parsing prompt, causing its stable information extraction capability in logs of different formats. HitAnomaly employs a classification-based approach, encoding normal and anomalous data types separately to enhance their distinguishability, ultimately achieving consistent performance on both datasets with F1-scores of 0.980. Notably, SemiRALD, as a semi-supervised approach, even outperforms all three supervised learning models, namely LogRobust, NeuralLog, and HitAnomaly, by 2.3 percentage points, 1.3 percentage points, and 1.3 percentage points on the HDFS dataset, respectively. It also outperforms them by 16.0 percentage points, 1.3 percentage points, and 7.0 percentage points on the BGL dataset in order, showcasing a more competitive advantage.

Answer to RQ1: SemiRALD consistently outperforms current state-of-the-art ALD approaches across diverse datasets, highlighting its impressive generality towards heterogeneous log formats and superior anomaly detection abilities.

5.2. Contribution of each module within SemiRALD (RQ2)

We disassembled each module within SemiRALD to evaluate their contribution. As detailed in Table 4, Line 1 is the full model of SemiALD, 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. Since the BGL dataset is more intricate than the HDFS dataset, the differences, in effect, are expected to be more pronounced. Therefore, we selected BGL datasets of varying scales for comparison, ranging from 1%, 10%, 20%, 50%, to 100% of the dataset size, thereby carrying out more extensive experiments in this section.

Across all configurations, we observed consistent trends concerning dataset size. As the dataset size increased, the F1-scores increased, underscoring the importance of ample data for model performance. Additionally, the performance degradation of Line 5 compared to RoBERTa and BERT-based configurations highlights the superior capabilities of a large-scale pre-trained language model in capturing contextual information and semantic relationships within the data.

Compared to replacing the first module with BERT and removing it, utilizing the RoBERTa module led to average F1-score improvements of 0.9 percentage points and 15.6 percentage points. Introducing the Bi-LSTM module resulted in an average improvement of 7.6 percentage points, and adding an attention layer led to an average improvement of 0.8 percentage points. When the dataset scale was reduced to 1%, SemiRALD demonstrated an approximate 1 percentage point increase compared to Lines 3 and 4 but showed enhancements of 9.4 percentage points and 19.5 percentage points, respectively, when compared to Lines 2 and 5. This highlights the significance of integrating large-scale pre-trained language models. When the dataset size was reduced to 10%, the F1-scores of Line 2 and Line 5 increased by 2.3 percentage points and 5.4 percentage points, respectively, compared to the 1% dataset. As the dataset size was reduced to 20%, all models showed a tendency for F1-scores to stabilize with the increase in dataset size.

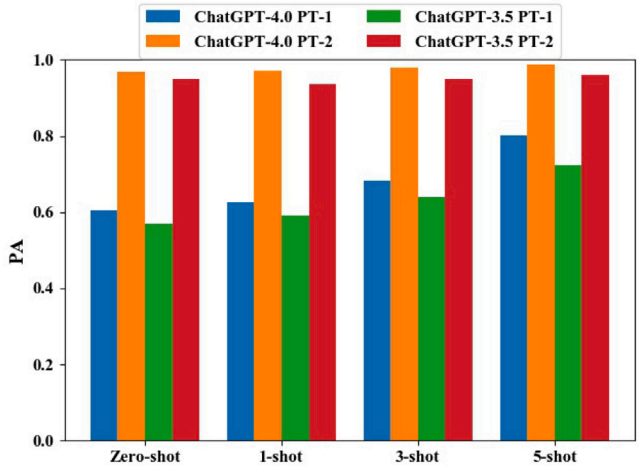
Answer to RQ2: In summary, each module of SemiRALD contributes to improving log anomaly detection performance, with Module 1 showing the most remarkable improvement.

5.3. Analysis on ChatGPT as the log parser (RQ3)

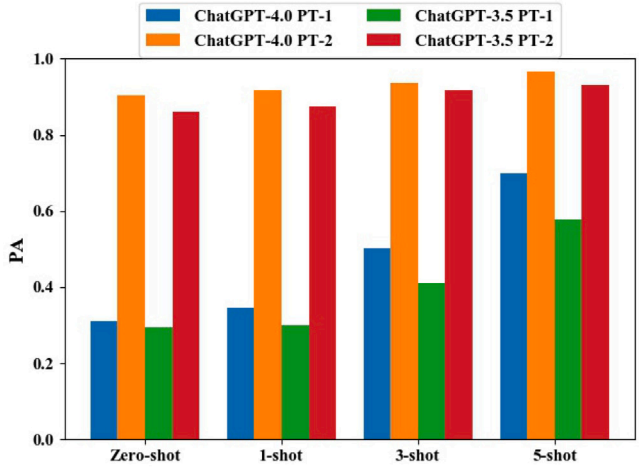
In this section, we extensively evaluate the performance of ChatGPT in log parsing with zero-shot and few-shot settings using the HDFS and BGL datasets, where two versions of ChatGPT are involved. Fig. 6 illustrates the performance comparison in information integrity (evaluated by PA) after log parsing via two versions of ChatGPT. Apparently, ChatGPT's performance in PT-2 surpasses that in PT-1 across both datasets. This underscores the utility of "extraction and replacement rules" in aiding ChatGPT's comprehension of log information. The absence of these rules causes ChatGPT to struggle to filter dynamic

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	0.980	0.987	0.990	0.990	0.990
(2) + Attention	F1	0.888 (−9.39%)	0.909 (−7.09%)	0.917 (−7.37%)	0.918 (−7.27%)	0.920 (−7.07%)
(3) + Bi-LSTM	F1	0.970 (−1.02%)	0.976 (−1.11%)	0.980 (−1.01%)	0.984 (−0.61%)	0.986 (−0.40%)
BERT						
(4) + Bi-LSTM + Attention	F1	0.968 (−1.22%)	0.977 (−1.01%)	0.982 (−0.81%)	0.983 (−0.71%)	0.983 (−0.71%)
(5) Bi-LSTM + Attention	F1	0.789 (−19.49%)	0.848 (−14.08%)	0.850 (−14.14%)	0.850 (−14.14%)	0.830 (−16.16%)



(a) HDFS



(b) BGL

Fig. 6. The performance of ChatGPT on two datasets.

variables from intricate and diverse log data, leading to lower PA. Additionally, ChatGPT exhibits superior performance on the HDFS dataset compared to the BGL dataset. This discrepancy is ascribed to the latter carrying longer and more complex log samples, presenting heightened comprehension hurdles.

With 5-shot prompts, providing examples of pid and date, ChatGPT-4.0's accuracy significantly improved. In contrast, ChatGPT-3.5 often exhibited noise and omitted processing one or two variables in the sequence. This led to an error rate of 4.6% in handling the mentioned log sequence, contributing to its lower PA value, a challenge that ChatGPT-4.0 successfully addressed.

In general, ChatGPT-3.5 tends to miss some dynamic variables within longer and more complex log sequences, leading to its lower scores. In contrast, the performance of ChatGPT-4.0 is considerably better, although it occasionally misses dynamic variables due to noise. Its accuracy far surpasses that of ChatGPT-3.5. Even in zero-shot scenarios, ChatGPT-4.0 improves the accuracy. With an increasing number of few-shot examples, the performance of ChatGPT-4.0 steadily improved.

Furthermore, we evaluate the log parsing performance of our method in comparison to other log parsers across different types of software systems. In addition to the three datasets introduced in Section 4.1, we select four log datasets from distinct software systems — Windows, Android, OpenSSH, and Proxifier — provided by LogHub. These systems span a variety of domains, including distributed systems, operating systems, mobile systems, supercomputers, server applications, and standalone software. For each system, we select 4000 log messages, along with their respective log templates and parameters, covering all log formats, which serve as the ground truth for evaluating the log parsers. We assess the performance of five well-known log parsers: Drain [28], Logram [55], AEL [35], Spell [54], and SPINE [56], in addition to our state-of-the-art ChatGPT-based parser: ChatGPT-4.0 PT2, across these seven datasets.

Table 5 presents the PA of these log parsers. In general, our ChatGPT-based parser demonstrates significant advantages in terms of PA. Although more than half of the log sequences contain only 1–2 dynamic variables, identifying and parsing these variables fully remains challenging. This is especially true for cases where the variable formats are highly diverse or where the context surrounding the variable is complex. The five well-known log parsers consistently show limitations, particularly in their inability to correctly or fully identify dynamic variables. Additionally, their performance varies significantly across different datasets, which compromises their reliability and consistency. This inconsistency further exacerbates the challenges in log parsing, especially in heterogeneous environments with diverse log formats.

In comparison, our ChatGPT-based parser maintains stable performance, irrespective of dataset type or complexity. Unlike existing parsers, which struggle with long addresses (e.g., file paths), usernames, and short port numbers (e.g., uid), our model is able to recognize and parse these dynamic variables effectively. This superior handling of log sequences leads to more accurate log interpretation and improves anomaly detection performance. However, upon closer inspection, we found that ChatGPT sometimes introduces noise during the processing phase. Specifically, for logs with similar structures, there is a possibility that a small number of logs may fail to undergo proper replacement for certain dynamic variables. This could result in minor discrepancies in parsing accuracy, although these instances are relatively rare.

Fig. 7 displays the comparative performance of ChatGPT-4.0 using both PT-1 and PT-2 prompts, alongside a comparison with the widely used and top-performing Drain parser. The incorporation of "extraction and replacement rules" enhances ChatGPT-4.0's performance, whereas the use of few-shot examples yields more consistent performance across various special variables.

While the Drain parser handles simple variables effectively, due to the varied definitions of usernames, Drain is unable to handle

Table 5
A comparison of the parsing accuracy (PA) between the well-known parsers and our ChatGPT-based parser (the last column).

Dataset	Category	Drain	Logram	AEL	Spell	SPINE	Ours
HDFS	Distributed file system	0.595	0.325	0.608	0.374	0.426	**0.922**
BGL	Supercomputer	0.677	0.464	0.439	0.274	0.361	**0.962**
Windows	Operating system	0.493	0.397	0.293	0.202	0.151	**0.968**
Android	Mobile system	0.684	0.378	0.393	0.270	0.184	**0.903**
OpenSSH	Server applications	0.638	0.425	0.246	0.281	0.253	**0.996**
Proxifier	Standalone software	0.392	0.261	0.473	0.425	0.226	**0.931**
ILMS	Low-resource	0.705	0.582	0.624	0.597	0.573	**0.947**
Average		0.598	0.405	0.419	0.346	0.311	**0.948**

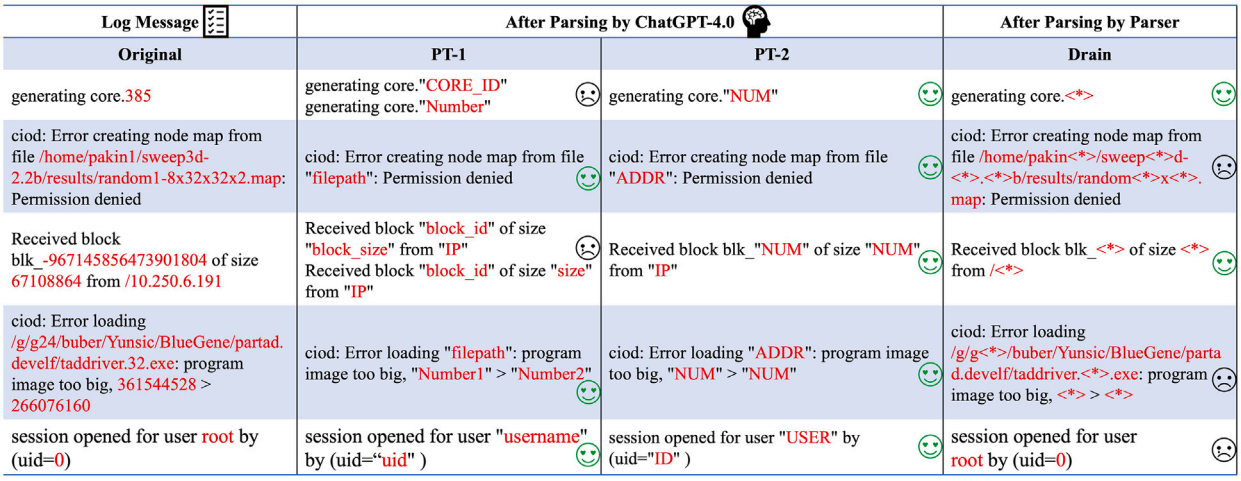


Fig. 7. The performance of ChatGPT-4.0 and Drain.

them. In contrast, ChatGPT can effectively recognize and replace them with "*ADDR*". When encountering addresses such as "*/home/pakin1/sweep3d-2.2b/results/random1-8x32x32x2.map*", Drain's replacement effect is suboptimal and fails to identify the complete address, whereas ChatGPT demonstrates superior capabilities in recognizing file addresses.

Answer to RQ3: ChatGPT outperforms five well-known parsers in log parsing, demonstrating stronger generality. Moreover, the performance of ChatGPT-4.0 significantly surpasses that of ChatGPT-3.5. Diverse examples further enhance ChatGPT's understanding and processing capabilities of logs.

5.4. Exploration in low-resource scenarios (RQ4)

To systematically evaluate performance on low-resource scenarios, we firstly selected varying scales of HDFS and BGL datasets, ranging from 0.1% to 15%. Experiments were conducted at 0.1%, 0.5%, 1%, 5%, 10%, and 15%. To make a comprehensive comparison, we select representative models, namely DeepLog, PLELog, and LogRobust, from learning paradigms of unsupervised, semi-supervised, and fully supervised, where the experimental results are shown in Table 6.

As can be seen from Table 6, with the reduction of data size to the scale of 15%-0.1%, the performance of DeepLog and PLELog decreases by an average of 19.7–28.4 percentage points and 3.5–20.5 percentage points in terms of F1-score. As for the fully supervised approach, namely LogRobust, its performance also decreases by 21.6 percentage points at most. Regarding SemiRALD, its anomalous log detection performance degenerates as well but is extremely limited. Even though the data size has been shrunk to 0.1%, its performances on Precision, Recall, and F1-score only degrade by 2.6 percentage points, 4.1 percentage points, and 3.5 percentage points, respectively, demonstrating a highly robust characteristic in anomalous log detection.

Besides, except for SemiRALD, the other three models exhibit unstable anomaly detection performance at smaller dataset scales, particularly at 0.1% and 0.5%. For instance, at a dataset size of 0.1% for the HDFS dataset, SemiRALD achieves F1-score increases of 32.0 percentage points, 17.4 percentage points, and 44.7 percentage points compared with DeepLog, PLELog, and LogRobust, respectively. As the dataset size increases, all models experience the most significant improvement in F1-scores when the dataset reaches 5%, with increases of 15.8 percentage points (DeepLog), 10.6 percentage points (PLELog), 7.7 percentage points (LogRobust), and 0.3 percentage points (SemiRALD) compared to the 1% scale. Upon reaching a dataset size of 10%, the F1-scores of all models stabilize.

At a dataset size of 0.1% for the BGL dataset, SemiRALD attains an F1-score of 0.953, surpassing PLELog, the second-best performer, by 32.1 percentage points. In contrast, LogRobust achieves an F1-score of only 0.627, suggesting that without pre-trained language models, even a fully supervised model encounters difficulty in learning log sequence features with extremely limited data. As the dataset size increases, the F1-scores of the four models also improve, but LogRobust shows some fluctuations. At a dataset size of 15%, LogRobust's F1-score is 3.4 percentage points lower than that at 10%. Moreover, at a dataset size of 15%, SemiRALD achieves an F1-score of 0.953, which is 12.2 percentage points higher than that of PLELog.

Fig. 8 illustrates the performance of different models on our developed system log dataset for managing experiment workflows in the chemical and materials fields. Due to the limited volume of our system logs (only 54,637 entries), we conducted experiments using 10%, 50%, and 100% of the data. We observed that when using 10% of the data, SemiRALD significantly outperformed the other three models, with an average F1-score improvement of 47.2%. Even as the dataset scale increased to 50% and 100%, SemiRALD maintained its lead over the other three models, with average F1-scores of 40.4% and 33.1%, respectively, demonstrating the robustness of SemiRALD across different software systems.

Table 6
The performance of different models on two small-scale datasets.

Model		HDFS						BGL					
		0.1%	0.5%	1%	5%	10%	15%	0.1%	0.5%	1%	5%	10%	15%
DeepLog	P	0.942	0.562	0.638	0.879	0.941	0.962	0.124	0.156	0.184	0.237	0.276	0.278
	R	0.485	0.894	0.995	0.997	0.994	0.994	0.987	0.969	0.991	1.000	0.988	0.959
	F1	0.640	0.690	0.777	0.935	0.967	0.978	0.221	0.268	0.311	0.383	0.431	0.432
PLELog	P	0.947	0.910	0.934	0.972	0.974	0.974	0.583	0.564	0.617	0.765	0.786	0.895
	R	0.972	0.690	0.794	0.956	1.000	1.000	0.691	0.587	0.706	0.872	0.883	0.817
	F1	0.786	0.785	0.859	0.964	0.987	0.987	0.632	0.576	0.659	0.815	0.832	0.855
LogRobust	P	0.519	0.673	0.794	0.926	0.945	0.964	0.500	0.506	0.632	0.787	0.851	0.819
	R	0.507	0.646	0.999	1.000	1.000	1.000	0.840	0.856	0.896	0.941	0.959	0.924
	F1	0.513	0.659	0.885	0.962	0.972	0.982	0.627	0.636	0.741	0.857	0.902	0.868
SemiRALD	P	0.955	0.965	0.979	0.983	0.994	0.994	0.978	0.969	0.980	0.979	0.989	0.991
	R	0.966	0.973	0.983	0.987	0.989	0.991	0.933	0.954	0.979	0.984	0.984	0.989
	F1	**0.960**	**0.968**	**0.981**	**0.984**	**0.991**	**0.993**	**0.953**	**0.961**	**0.980**	**0.982**	**0.987**	**0.990**

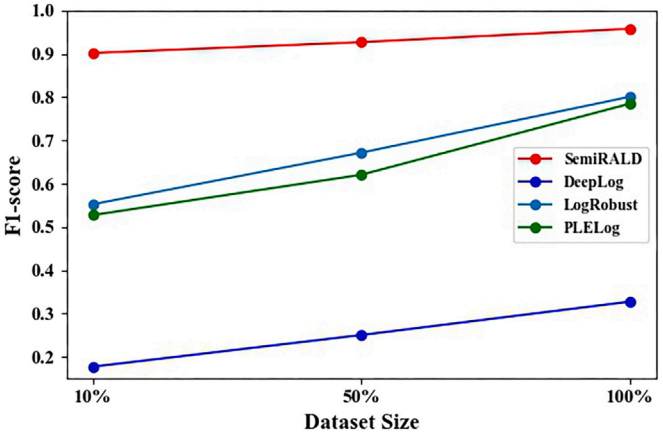
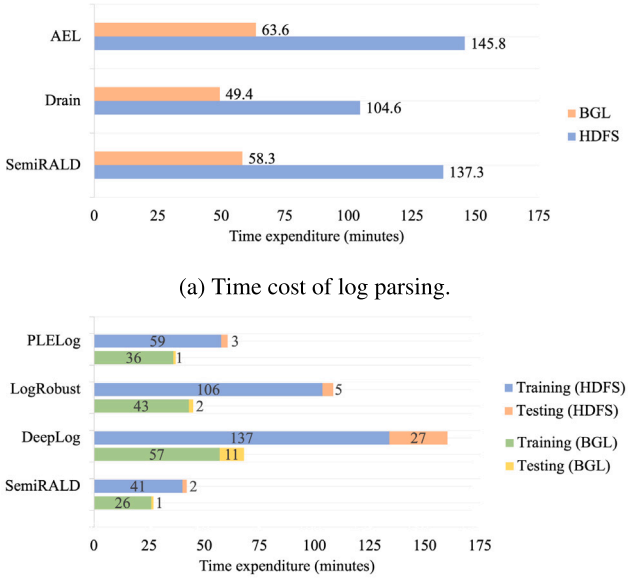


Fig. 8. The performance of different models on our ILMS log dataset.



(a) Time cost of log parsing.

(b) Time cost of log anomaly detection.

Fig. 9. Computational efficiency of SemiRALD on two datasets.

Answer to RQ4: SemiRALD consistently outperforms other ALD approaches of different learning paradigms and demonstrates robustness in diverse data scales. Besides, its consistent and superior performance across the three datasets also highlights its generality capabilities.

5.5. Efficiency of SemiRALD (RQ5)

In this section, we select the HDFS and BGL datasets to investigate the efficiency advantages of SemiRALD. As shown in Fig. 9(a), in the log parsing task, we compare our ChatGPT-based parser with two widely used parsers, Drain and AEL, which demonstrate the best performance in RQ3. As shown in Fig. 9(b), in log anomaly detection, we select representative models from different learning paradigms to ensure a comprehensive comparison: DeepLog (unsupervised), PLELog (semi-supervised), and LogRobust (supervised), and compare them with SemiRALD in terms of both training and testing time.

As shown in Fig. 9, SemiRALD demonstrates an advantage in terms of efficiency. In log parsing, it consumes less time than AEL but more than Drain. Although the average time consumption of SemiRALD is 19.6% higher than that of Drain, its PA accuracy improves by an average of 30.6%. We believe this is a significant advantage because the improvement in accuracy outweighs the relatively higher time cost, making SemiRALD more effective for practical applications, where accuracy is a priority. Additionally, the time consumption issue can be addressed through parallel processing using cluster computing, allowing for faster processing without compromising accuracy.

In anomaly detection, SemiRALD shows a clear leading advantage. The average time expenditure of SemiRALD is reduced by 69.8% compared to DeepLog, 29.3% compared to PLELog, and 55.1% compared to LogRobust. Overall, we find that SemiRALD outperforms the other models in terms of efficiency. For instance, PLELog, which uses the Drain parser and shows relatively lower time consumption, has a total time cost close to that of SemiRALD. However, SemiRALD performs better across both datasets, with an average F1-score improvement of 2.3% and more stable performance. Therefore, we believe that SemiRALD is the most efficient model overall, offering a balance of time efficiency and superior detection accuracy.

Answer to RQ5: SemiRALD demonstrates a leading advantage in time efficiency for anomaly detection, striking a better balance between efficiency and accuracy, and outperforms other models in both log parsing and anomaly detection tasks.

6. Threats to validity

This section clarifies the threats to log formatting, noise in ChatGPT, randomness in dataset scaling selection, 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 even be generated using the programmer's native language. Hence, to accommodate different formats and encompass all dynamic variables, it is imperative to establish distinct 'extraction and replacement rules' tailored to log formats for ChatGPT. This will enhance ChatGPT's accuracy in comprehending logs.

Noise in ChatGPT: In Section 5.3, we investigated the performance of ChatGPT in log parsing. As indicated by the results, ChatGPT inevitably introduces noise during processing. Consequently, some logs may fail to undergo replacement for certain dynamic variables. The introduction of such noise presents a challenge to the subsequent model's anomaly detection predictions. Currently, we are exploring a wider range of prompt templates to mitigate or eliminate the potential for noise.

Randomness in Dataset Scaling Selection: In low-resource scenarios, the random selection of dataset scales can introduce variability, potentially influencing the results. To address this, we employ the DBSCAN method to generate diverse samples that capture a wide range of log formats. While DBSCAN is designed to create diverse and relevant samples, the inherent variability in its output may still lead to fluctuations in performance metrics. To minimize this effect, we conducted five independent experiments for each experimental setting and reported the mean of the results as the final outcome.

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. Related work

7.1. ChatGpt for log parsing

The advent of Large Language Models (LLMs) such as ChatGPT [57] has demonstrated significant potential across various domains, including conversational interactions [58], language understanding [59], and text extraction [60]. Recent studies have also explored the potential of ChatGPT in log analysis and parsing. Cheng et al. [57] conducted end-to-end data analysis using ChatGPT-4.0 across databases from various fields. Han et al. [61] assessed ChatGPT-3.5's performance in log parsing, focusing on its ability to extract log events and parameters. LogGPT [62] introduces a novel approach to log-based anomaly detection, leveraging ChatGPT-3.5's language interpretation capabilities. The results show promising performance and strong interpretability, providing valuable insights into the potential of prompt-based models for log data analysis. However, LogGPT's performance on the BGL dataset is suboptimal, underscoring the need for further improvements to this approach.

The continued development of large language models has led to innovations such as LogPPT [12], which introduces a prompt-based few-shot learning approach for log parsing. This method captures log templates and parameters using minimally labeled data, outperforming traditional techniques in both effectiveness and efficiency across several public log datasets. DivLog [63] utilizes in-context learning with LLMs to enhance log parsing by leveraging diverse, small samples and training-free prompt construction, achieving state-of-the-art performance in accuracy and template precision across multiple datasets. Additionally, LILAC [64] integrates LLMs with an adaptive parsing cache and ICL, significantly improving template accuracy, parsing efficiency, and reducing reliance on LLM queries. This advancement provides a robust solution to the challenges of log analysis. Despite the growing interest in applying ChatGPT to log parsing, research in software engineering remains limited. A comprehensive investigation into the effectiveness of ChatGPT for automatic log parsing is still lacking, particularly regarding how ChatGPT-4.0 compares to its predecessor, ChatGPT-3.5, and other log parsing models.

7.2. Log anomaly detection models

The challenge of anomaly detection in software logs is commonly regarded as a binary classification problem [65]. Currently, researchers have proposed various methods for detecting abnormal logs from large-scale log datasets. These algorithms can be categorized into unsupervised [29,66], and supervised [16,20,52,67] learning approaches. Zhang et al. [16] introduced LogRobust, which represents log events as semantic vectors and uses an attention-based Bi-LSTM model to detect anomalies. Huang et al. [51] 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. While supervised methods are capable of accurately diagnosing system conditions, their primary limitation lies in the requirement for a substantial volume of normal and abnormal data during the training phase. Consequently, researchers have also explored unsupervised and semi-supervised approaches. Du et al. [29] 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. [68] 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. Yang et al. [20,52] introduced a semi-supervised learning framework known as PLELog, which relies on probabilistic label estimation. PLELog utilizes the HDBSCAN [69] clustering method for estimating labels probabilistically on unlabeled log sequences, addressing the challenge of limited labeling. NeuralLog [14] introduces a novel log-based anomaly detection method based on a Transformer classification model that eliminates the need for log preprocessing.

BERT [41], 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 [70]. LogBERT [15] 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 [53] employs unsupervised learning using the BERT model to detect anomalous logs, although it does not fully consider interactive factors between normal and abnormal logs. BERTlog [71] 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 [44] represents a significant advancement over the BERT model; however, its extensive exploration in the realm of log anomaly detection remains incomplete.

8. Conclusion

In this paper, we introduced SemiRALD, a novel semi-supervised framework for robust anomalous log detection. By integrating LLM for accurate log parsing and a hybrid language model combining RoBERTa with attention-based Bi-LSTM, SemiRALD addresses key challenges in anomaly detection, including data scarcity and log format diversity. Extensive experiments have demonstrated the effectiveness of SemiRALD. Future research will focus on further exploring the scalability and adaptability of SemiRALD to larger and more complex datasets, as well as investigating its applicability to different domains within the realm of reliability engineering. We also intend to conduct comprehensive studies on the interpretability of the detected anomalies, providing clearer insights and actionable recommendations to system administrators.

CRediT authorship contribution statement

Yicheng Sun: Writing – review & editing, Writing – original draft, Visualization, Software, Resources, Methodology, Formal analysis, Data curation, Conceptualization. **Jacky Wai Keung:** Validation, Supervision, Resources, Project administration, Investigation, Funding acquisition. **Zhen Yang:** Writing – review & editing, Visualization, Validation, Supervision, Methodology. **Shuo Liu:** Visualization, Validation. **Hi Kuen Yu:** Visualization, Project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is partially supported by the General Research Fund of the Research Grants Council of Hong Kong and the research funds from the City University of Hong Kong (6000796, 9229109, 9229098, 9220103, 9229029).

Data availability

We release our source code publicly at https://github.com/log-detection/SemiRALD.

References

[1] Jesse Nyyssälä, Mika Mäntylä, Martín Varela, How to configure masked event anomaly detection on software logs? in: 2022 IEEE International Conference on Software Maintenance and Evolution, ICSME, IEEE, 2022, pp. 414–418.

[2] Simon Gökstorp, Jakob Nyberg, Yeongwoo Kim, Pontus Johnson, György Dán, Anomaly detection in security logs using sequence modeling, in: NOMS 2024-2024 IEEE Network Operations and Management Symposium, IEEE, 2024, pp. 1–9.

[3] Yicheng Sun, Jacky Keung, Jingyu Zhang, Hi Kuen Yu, Wenqiang Luo, Shuo Liu, Unveiling hidden anomalies: Leveraging SMAC-LSTM for enhanced software log analysis, in: 48th IEEE International Conference on Computers, Software, and Applications (COMPSAC 2024): Digital Development for a Better Future, 2024.

[4] Haibo Mi, Huaimin Wang, Yangfan Zhou, Michael Rung-Tsong Lyu, Hua Cai, Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems, IEEE Trans. Parallel Distrib. Syst. 24 (6) (2013) 1245–1255.

[5] Jesse Nyyssölä, Mika Mäntylä, Efficiency of unsupervised anomaly detection methods on software logs, 2023, arXiv preprint arXiv:2312.01934.

[6] Boxi Yu, Jiayi Yao, Qiuai Fu, Zhiqing Zhong, Haotian Xie, Yaoliang Wu, Yuchi Ma, Pinjia He, 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, IEEE Computer Society, 2023, pp. 392–404.

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

[8] Jieming Zhu, Pinjia He, Qiang Fu, Hongyu Zhang, Michael R Lyu, Dongmei Zhang, Learning to log: Helping developers make informed logging decisions, in: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Vol. 1, IEEE, 2015, pp. 415–425.

[9] Shilin He, Jieming Zhu, Pinjia He, Michael R. Lyu, Experience report: System log analysis for anomaly detection, in: 2016 IEEE 27th International Symposium on Software Reliability Engineering, ISSRE, IEEE, 2016, pp. 207–218.

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

[11] Xingfang Wu, Heng Li, Foutse Khomh, On the effectiveness of log representation for log-based anomaly detection, Empir. Softw. Eng. 28 (6) (2023) 137.

[12] Van-Hoang Le, Hongyu Zhang, Log parsing with prompt-based few-shot learning, in: 2023 IEEE/ACM 45th International Conference on Software Engineering, ICSE, IEEE, 2023, pp. 2438–2449.

[13] Lingzhi Wang, Nengwen Zhao, Junjie Chen, Pinnong Li, Wenchi Zhang, Kaixin Sui, Root-cause metric location for microservice systems via log anomaly detection, in: 2020 IEEE International Conference on Web Services, ICWS, IEEE, 2020, pp. 142–150.

[14] Van-Hoang Le, Hongyu Zhang, Log-based anomaly detection without log parsing, in: 2021 36th IEEE/ACM International Conference on Automated Software Engineering, ASE, IEEE, 2021, pp. 492–504.

[15] Haixuan Guo, Shuhan Yuan, Xintao Wu, Logbert: Log anomaly detection via bert, in: 2021 International Joint Conference on Neural Networks, IJCNN, IEEE, 2021, pp. 1–8.

[16] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, 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, 2019, pp. 807–817.

[17] Sasho Nedelkoski, Jasmin Bogatinovski, Alexander Acker, Jorge Cardoso, Odej Kao, Self-attentive classification-based anomaly detection in unstructured logs, in: 2020 IEEE International Conference on Data Mining, ICDM, IEEE, 2020, pp. 1196–1201.

[18] Zhenhao Li, Chuan Luo, Tse-Hsun Chen, Weiyi Shang, Shilin He, Qingwei Lin, Dongmei Zhang, Did we miss something important? studying and exploring variable-aware log abstraction, in: 2023 IEEE/ACM 45th International Conference on Software Engineering, ICSE, IEEE, 2023, pp. 830–842.

[19] Van-Hoang Le, Hongyu Zhang, Log-based anomaly detection with deep learning: How far are we? in: Proceedings of the 44th International Conference on Software Engineering, 2022, pp. 1356–1367.

[20] Lin Yang, Junjie Chen, Zan Wang, Weijing Wang, Jiajun Jiang, Xuyuan Dong, Wenbin Zhang, Semi-supervised log-based anomaly detection via probabilistic label estimation, in: 2021 IEEE/ACM 43rd International Conference on Software Engineering, ICSE, IEEE, 2021, pp. 1448–1460.

[21] Xiaoxue Ma, Jacky Keung, Pinjia He, Yan Xiao, Xiao Yu, Yishu Li, A semisupervised approach for industrial anomaly detection via self-adaptive clustering, IEEE Trans. Ind. Inform. 20 (2) (2024) 1687–1697.

[22] Marta Catillo, Antonio Pecchia, Umberto Villano, AutoLog: Anomaly detection by deep autoencoding of system logs, Expert Syst. Appl. 191 (2022) 116263.

[23] Jieming Zhu, Shilin He, Pinjia He, Jinyang Liu, Michael R Lyu, Loghub: A large collection of system log datasets for ai-driven log analytics, in: 2023 IEEE 34th International Symposium on Software Reliability Engineering, ISSRE, IEEE, 2023, pp. 355–366.

[24] Wei Xu, Ling Huang, Armando Fox, David Patterson, Michael I Jordan, Detecting large-scale system problems by mining console logs, in: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, 2009, pp. 117–132.

[25] Adam Oliner, Jon Stearley, What supercomputers say: A study of five system logs, in: 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN'07, IEEE, 2007, pp. 575–584.

[26] Zanis Ali Khan, Donghwan Shin, Domenico Bianculli, Lionel Briand, Guidelines for assessing the accuracy of log message template identification techniques, in: Proceedings of the 44th International Conference on Software Engineering, 2022, pp. 1095–1106.

[27] Zeyang Ma, An Ran Chen, Dong Jae Kim, Tse-Hsun Chen, Shaowei Wang, Llmparser: An exploratory study on using large language models for log parsing, in: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, 2024, pp. 1–13.

[28] Pinjia He, Jieming Zhu, Zibin Zheng, Michael R. Lyu, Drain: An online log parsing approach with fixed depth tree, in: 2017 IEEE International Conference on Web Services, ICWS, IEEE, 2017, pp. 33–40.

[29] Min Du, Feifei Li, Guineng Zheng, Vivek Srikumar, Deeplog: Anomaly detection and diagnosis from system logs through deep learning, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 1285–1298.

[30] Min Du, Feifei Li, Spell: Online streaming parsing of large unstructured system logs, IEEE Trans. Knowl. Data Eng. 31 (11) (2018) 2213–2227.

[31] Meiyappan Nagappan, Mladen A. Vouk, Abstracting log lines to log event types for mining software system logs, in: 2010 7th IEEE Working Conference on Mining Software Repositories, MSR 2010, IEEE, 2010, pp. 114–117.

[32] Risto Vaarandi, Mauno Pihelgas, Logcluster-a data clustering and pattern mining algorithm for event logs, in: 2015 11th International Conference on Network and Service Management, CNSM, IEEE, 2015, pp. 1–7.

[33] Keiichi Shima, Length matters: Clustering system log messages using length of words, 2016, arXiv preprint arXiv:1611.03213.

[34] Liang Tang, Tao Li, Chang-Shing Perng, LogSig: Generating system events from raw textual logs, in: Proceedings of the 20th ACM International Conference on Information and Knowledge Management, 2011, pp. 785–794.

[35] Zhen Ming Jiang, Ahmed E Hassan, Parminder Flora, Gilbert Hamann, Abstracting execution logs to execution events for enterprise applications (short paper), in: 2008 the Eighth International Conference on Quality Software, IEEE, 2008, pp. 181–186.

[36] Yintong Huo, Yuxin Su, Cheryl Lee, Michael R. Lyu, Semparser: A semantic parser for log analytics, in: 2023 IEEE/ACM 45th International Conference on Software Engineering, ICSE, IEEE, 2023, pp. 881–893.

[37] Lin Zhang, Xueshuo Xie, Kunpeng Xie, Zhi Wang, Ye Lu, Yujun Zhang, 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, Springer, 2019, pp. 123–134.
[38] Konstantinos I. Roumeliotis, Nikolaos D. Tselikas, Chatgpt and open-ai models: A preliminary review, Futur. Internet 15 (6) (2023) 192.
[39] Jules White, Sam Hays, Quchen Fu, Jesse Spencer-Smith, Douglas C Schmidt, Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design, in: Generative AI for Effective Software Development, Springer, 2024, pp. 71–108.
[40] Shuzheng Gao, Xin-Cheng Wen, Cuiyun Gao, Wenxuan Wang, Hongyu Zhang, Michael R Lyu, What makes good in-context demonstrations for code intelligence tasks with llms? in: 2023 38th IEEE/ACM International Conference on Automated Software Engineering, ASE, IEEE, 2023, pp. 761–773.
[41] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, 2018, arXiv preprint arXiv:1810.04805.
[42] Ganesh Jawahar, Benoît Sagot, Djamé Seddah, What does BERT learn about the structure of language? in: ACL 2019-57th Annual Meeting of the Association for Computational Linguistics, 2019.
[43] Mikhail V. Koroteev, BERT: a review of applications in natural language processing and understanding, 2021, arXiv preprint arXiv:2103.11943.
[44] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, Veselin Stoyanov, Roberta: A robustly optimized bert pretraining approach, 2019, arXiv preprint arXiv:1907.11692.
[45] Alex Graves, Alex Graves, Long short-term memory, in: Supervised Sequence Labelling with Recurrent Neural Networks, Springer, 2012, pp. 37–45.
[46] Long Short-Term Memory, Long short-term memory, Neural Comput. 9 (8) (2010) 1735–1780.
[47] Anurag Kulshrestha, Venkataraghavan Krishnaswamy, Mayank Sharma, Bayesian BILSTM approach for tourism demand forecasting, Ann. Tour. Res. 83 (2020) 102925.
[48] Zhiheng Huang, Wei Xu, Kai Yu, Bidirectional LSTM-CRF models for sequence tagging, 2015, arXiv preprint arXiv:1508.01991.
[49] Özal Yildirim, A novel wavelet sequence based on deep bidirectional LSTM network model for ECG signal classification, Comput. Biol. Med. 96 (2018) 189–202.
[50] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, Xiaowei Xu, DBSCAN revisited, revisited: why and how you should (still) use DBSCAN, ACM Trans. Database Syst. 42 (3) (2017) 1–21.
[51] Shaohan Huang, Yi Liu, Carol Fung, Rong He, Yining Zhao, Hailong Yang, Zhongzhi Luan, Hitanomaly: Hierarchical transformers for anomaly detection in system log, IEEE Trans. Netw. Serv. Manag. 17 (4) (2020) 2064–2076.
[52] Yijun Lin, Yao-Yi Chiang, 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), IEEE, 2022, pp. 1024–1033.
[53] Yukyung Lee, Jina Kim, Pilsung Kang, Lanobert: System log anomaly detection based on bert masked language model, Appl. Soft Comput. 146 (2023) 110689.
[54] Min Du, Feifei Li, Spell: Streaming parsing of system event logs, in: 2016 IEEE 16th International Conference on Data Mining, ICDM, IEEE, 2016, pp. 859–864.
[55] Hetong Dai, Heng Li, Che-Shao Chen, Weiyi Shang, Tse-Hsun Chen, Logram: Efficient log parsing using n n-gram dictionaries, IEEE Trans. Softw. Eng. 48 (3) (2020) 879–892.
[56] Xuheng Wang, Xu Zhang, Liqun Li, Shilin He, Hongyu Zhang, Yudong Liu, Lingling Zheng, Yu Kang, Qingwei Lin, Yingnong Dang, et al., SPINE: a scalable log parser with feedback guidance, in: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2022, pp. 1198–1208.
[57] Liying Cheng, Xingxuan Li, Lidong Bing, Is gpt-4 a good data analyst? 2023, arXiv preprint arXiv:2305.15038.
[58] Nuo Chen, Yan Wang, Haiyun Jiang, Deng Cai, Ziyang Chen, Jia Li, What would harry say? building dialogue agents for characters in a story, 2022, arXiv preprint arXiv:2211.06869.
[59] Zhen Yang, Fang Liu, Zhongxing Yu, Jacky Wai Keung, Jia Li, Shuo Liu, Yifan Hong, Xiaoxue Ma, Zhi Jin, Ge Li, Exploring and unleashing the power of large language models in automated code translation, Proc. ACM Softw. Eng. 1 (FSE) (2024) 1585–1608.
[60] Alberto Olmo, Sarath Sreedharan, Subbarao Kambhampati, GPT3-to-plan: Extracting plans from text using GPT-3, 2021, arXiv preprint arXiv:2106.07131.
[61] Xiao Han, Shuhan Yuan, Mohamed Trabelsi, Loggpt: Log anomaly detection via gpt, in: 2023 IEEE International Conference on Big Data (BigData), IEEE, 2023, pp. 1117–1122.
[62] Jiaxing Qi, Shaohan Huang, Zhongzhi Luan, Shu Yang, Carol Fung, Hailong Yang, Depei Qian, Jing Shang, Zhiwen Xiao, Zhihui Wu, Loggpt: Exploring chatgpt for log-based anomaly detection, in: 2023 IEEE International Conference on High Performance Computing & Communications, Data Science & Systems, Smart City & Dependability in Sensor, Cloud & Big Data Systems & Application, HPCC/DSS/SmartCity/DependSys, IEEE, 2023, pp. 273–280.
[63] Junjielong Xu, Ruichun Yang, Yintong Huo, Chengyu Zhang, Pinjia He, Divlog: Log parsing with prompt enhanced in-context learning, in: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, 2024, pp. 1–12.
[64] Zhihan Jiang, Jinyang Liu, Zhuangbin Chen, Yichen Li, Junjie Huang, Yintong Huo, Pinjia He, Jiazhen Gu, Michael R Lyu, Lilac: Log parsing using llms with adaptive parsing cache, Proc. ACM Softw. Eng. 1 (FSE) (2024) 137–160.
[65] Max Landauer, Sebastian Onder, Florian Skopik, Markus Wurzenberger, Deep learning for anomaly detection in log data: A survey, Mach. Learn. Appl. 12 (2023) 100470.
[66] Vannel Zeufack, Donghyun Kim, Daehee Seo, Ahyoung Lee, An unsupervised anomaly detection framework for detecting anomalies in real time through network system's log files analysis, High- Confid. Comput. 1 (2) (2021) 100030.
[67] Shi Ying, Bingming Wang, Lu Wang, Qingshan Li, Yishi Zhao, Jianga Shang, Hao Huang, Guoli Cheng, Zhe Yang, Jiangyi Geng, An improved KNN-based efficient log anomaly detection method with automatically labeled samples, ACM Trans. Knowl. Discov. Data (TKDD) 15 (3) (2021) 1–22.
[68] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, et al., Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs, in: IJCAI, Vol. 19, 2019, pp. 4739–4745.
[69] Leland McInnes, John Healy, Accelerated hierarchical density based clustering, in: 2017 IEEE International Conference on Data Mining Workshops, ICDMW, IEEE, 2017, pp. 33–42.
[70] Kevin Clark, Urvashi Khandelwal, Omer Levy, Christopher D Manning, What does bert look at? an analysis of bert's attention, 2019, arXiv preprint arXiv:1906.04341.
[71] Song Chen, Hai Liao, Bert-log: Anomaly detection for system logs based on pre-trained language model, Appl. Artif. Intell. 36 (1) (2022) 2145642.