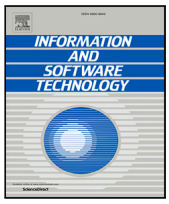
Information and Software Technology 187 (2025) 107869

Contents lists available at ScienceDirect

Information and Software Technology

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



SemiSMAC: A semi-supervised framework for log anomaly detection with automated hyperparameter tuning

Yicheng Sun [a], Jacky Wai Keung [a], Zhen Yang [b,*], Shuo Liu [a], Yihan Liao [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/SMAC-LSTM

Keywords:
Software reliability
Semi-supervised learning
Software log analysis
Anomaly detection
Sequential algorithm configuration
Hyperparameter optimization

ABSTRACT

Context: Logs generated during software operations are critical for system reliability and anomaly detection. However, their diversity, the scarcity of labeled data, and hyperparameter tuning challenges hinder traditional detection methods.

Objective: This paper presents SemiSMAC, a novel semi-supervised framework that leverages the Large Language Model for log parsing and grouping, combined with Sequential Model-based Algorithm Configuration (SMAC) for hyperparameter optimization to enhance anomaly detection.

Method: In this work, we leverage ChatGPT for log parsing and introduce a novel log grouping approach. This grouping process requires only a small number of labeled samples, which ChatGPT uses to generate pseudo-labels for the remaining data, thereby expanding the training set. Furthermore, SemiSMAC utilizes a Sequential Model-based Algorithm Configuration (SMAC) to automatically optimize the hyperparameters of the embedded models. This integration leads to consistent performance improvements, particularly in resource-constrained environments.

Results: SemiSMAC-LSTM, which uses LSTM as the backbone of the SemiSMAC framework, demonstrates superior performance in experiments on four widely used datasets. It outperforms six benchmark models, including three supervised learning models. In low-resource scenarios, SemiSMAC-LSTM exhibits exceptional robustness, showcasing its effectiveness in handling challenging detection tasks.

Conclusion: SemiSMAC demonstrates its potential to revolutionize anomaly detection in both large-scale and low-resource datasets. Its ability to deliver outstanding performance makes it a valuable tool for scalable and automated anomaly detection in real-world applications, paving the way for more reliable and scalable software engineering practices

1. Introduction

Modern software systems continuously generate vast volumes of log data to capture system events, monitor operational states, and support anomaly diagnosis and failure recovery [1]. These logs play a vital role in ensuring system reliability and security. However, analyzing such logs remains a challenging task due to their unstructured format, high variability across systems, and rapid evolution over time [2]. In practice, the lack of clean, high-quality labeled data poses a fundamental barrier to the development of reliable log analysis models [3,4]. Labeling logs is not only time-consuming but also requires extensive domain expertise to correctly identify abnormal patterns, especially in systems with diverse log structures and application-specific semantics. Moreover, many logs contain dynamic variables — such as timestamps, user IDs, and memory addresses — that obscure structural patterns and complicate the parsing process [5,6]. These factors collectively make automated log analysis a labor-intensive, low-reproducibility task that hinders scalability in real-world systems [7].

Despite significant progress in Deep Anomalous Log Detection (DALD) [2,8–10], existing methods face substantial limitations. While supervised models [11,12] can achieve high performance, they require large amounts of labeled data, which is often expensive to obtain and difficult to maintain. Unsupervised methods [13,14] remove the need for labeled data, but often struggle due to the absence of semantic or structural guidance. Semi-supervised approaches [15,16] attempt to strike a balance by leveraging a limited number of labeled instances. However, they continue to face three fundamental limitations that hinder broader adoption in real-world scenarios.

* 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), yihanliao4-c@my.cityu.edu.hk (Y. Liao).

https://doi.org/10.1016/j.infsof.2025.107869
Received 18 March 2025; Received in revised form 4 August 2025; Accepted 6 August 2025
Available online 16 August 2025

Limitations of Rule-Based Log Parsers. Most existing methods rely on traditional rule-based parsers — such as Drain or Spell [17,18] — for transforming raw logs into structured templates. These parsers are known to suffer from limited accuracy, particularly when handling logs with high variability or embedded dynamic variables. Importantly, recent studies have shown that inaccuracies in the parsing stage can propagate downstream, degrading the effectiveness of subsequent anomaly detection [6]. Despite this, log parsing and anomaly detection are still treated as independent steps in most existing pipelines, making it difficult to optimize them jointly. This disjointed architecture not only introduces inefficiencies but also limits the model's ability to adapt to parsing errors during detection. Therefore, improving the integration between parsing and anomaly detection is crucial to enhancing end-to-end log analysis performance.

Excessive Label Dependence in Semi-Supervised Methods. Although classified as semi-supervised, many existing approaches still rely heavily on large volumes of labeled data — particularly normal logs — for model training. For example, PLELog [15] utilizes 50% of all normal log sequences during training, while LogAnomaly [19] depends on a substantial proportion of labeled normal traces — often covering nearly the entire normal dataset — to achieve competitive performance. Such dependence on extensive human-annotated data contradicts the premise of low-resource applicability and poses a significant barrier to deployment in real-world settings where labeled data is scarce or expensive to obtain.

Limitations of Existing Hyperparameter Tuning. The performance of deep log anomaly detection models is highly sensitive to hyperparameter configurations. However, tuning these parameters efficiently remains a challenge. Most prior work [12,13,20,21] either applies fixed, empirically selected configurations or performs manual or grid-based search. These strategies are computationally expensive and often fail to generalize across datasets and model architectures. Moreover, improper tuning can lead to overfitting, degraded generalization, and inconsistent results across different log environments.

To address these challenges, we propose a unified semi-supervised anomaly detection framework that integrates a ChatGPT-assisted log grouping mechanism with a random forest-based Bayesian Optimization Algorithm (BOA), specifically the Sequential Model-based Algorithm Configuration (SMAC), to automate hyperparameter tuning across diverse backbone architectures and learning paradigms. To further reduce human labeling effort and extend supervision to unlabeled logs, we introduce pseudo-label propagation based on semantic log grouping. This mechanism enables the framework to assign meaningful labels to previously unannotated data, enhancing the quality of supervision without requiring additional manual effort. This integration enhances error tolerance and improves the accuracy of pseudo-labeling. We refer to the resulting framework as SemiSMAC, which can be instantiated with various backbones to form SemiSMAC-<T>, where <T>denotes a specific model architecture—analogous to genericity in programming.

As a semi-supervised learning framework, SemiSMAC only needs a tiny portion of labeled data for initiation. It first adopts ChatGPT to parse log messages to corresponding templates (as shown in Fig. 2) and then incorporates log grouping into the above process. Unlike prior work that treats parsing and detection as separate stages, our approach embeds log grouping directly into the ChatGPT-based parsing process and utilizes the grouping results for pseudo-label generation. This enables parsing and grouping to be performed in a single ChatGPT pass, reducing redundant processing and improving label propagation accuracy. Even if some of the logs cannot be exactly correctly parsed, ChatGPT can group them to their correct clusters based on semantic understanding, thereby improving the error tolerance of log parsing. Besides, parsed logs that are grouped into the same clusters tend to carry similar semantics. Thus, we can conjecture and assign pseudo-labels for those unlabeled logs based on the existence of labeled logs in their groups, thereby expanding supervised examples for DALD. Afterward, we adopt random forest-based BOA to automatically tune hyperparameters of the backbone model in SemiSMAC, thereby enabling the determination of discrete parameters within deep learning models while requiring less time.

In experiments, we use LSTM, a typical sequential deep neural network, as the default backbone of SemiSMAC, namely SemiSMAC-LSTM. Extensive assessments on two popular datasets demonstrate that SemiSMAC-LSTM outperforms six benchmark models, including three supervised learning models, with average improvements of 7.1% and 6.9% in terms of F1-score, respectively. Even in the low-resource data setting of 0.1% scale, SemiSMAC-LSTM shows its robustness and resilience, achieving average increases of 18.7% and 31.2% in F1-scores across the two datasets compared to three representative models, highlighting its effectiveness. In summary, The contributions of this paper can be summarized in three-fold:

- We innovatively exert ChatGPT to unify log parsing and grouping in semi-supervised DALD, effectively improving the error tolerance of log parsing and the accuracy of pseudo-label assignment.
- We are the first to introduce the SMAC method to automatically tune the hyperparameters of backbone models in the DALD field and seamlessly incorporate it into our framework. Various backbones and baselines are examined in this work, demonstrating its effectiveness in the DALD field.
- We conducted comprehensive experiments to validate the performance of SemiSMAC, including comparative assessment, ablation study, and low-resource scenario evaluation.

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

This work is an extension of our COMPSAC 2024 paper [22]. Compared to the preliminary version, we have transitioned the model to semi-supervised learning and innovatively used ChatGPT to unify log parsing and grouping. We have redesigned all experiments and utilized public datasets available on the LogHub [5] platform to facilitate better performance comparisons with existing models (RQ2). Additionally, we have compared the performance of ChatGPT against other parsers (RQ3) and evaluated the model's performance on low-resource datasets (RQ4).

2. Background and related work

2.1. Log parsing

Logs constitute semi-structured text data, typically generated through log statements in source code (e.g., logging.info(), printf(), and Console.WriteLine()) [23]. To ensure the reliability of systems, logs are widely used as they capture key states during system operation [3,24]. As shown in Fig. 1, these logs consist of both static messages and dynamic variables. Each log entry records a specific system event and contains dynamic information such as timestamps, numbers, ports, APIs, addresses, block/task IDs, and usernames.

Log parsing is employed to automatically convert each log message into a structured format by identifying an event template (constant part) associated with parameters (dynamic variable parts) [25,26]. Fig. 1 provides examples of raw log messages and the corresponding log events obtained through parsing. For example, in a software log message like "*2023-11-17 00:04:09.377 INFO: 34.139.166.52:0 - 'POST /webhook/order HTTP/1.0' 200 OK delete empty directory: /backend/upload/download/orders/49512/49512*", the timestamp is "*2023-11-17 00:04:09.377*", the severity level is "*INFO: 34.139.166.52:0 -*

Fig. 1. The example of log messages and log events.
Note: Dynamic variables are highlighted in **red**. Irrelevant details (i.e., timestamps, APIs, and software systems) at the top of the log will be removed directly.

Group ID (G_{truth})	Log ID	Original Log Sequences
1	1	081110 020313 6034 INFO dfs.DataNode$PacketResponder: Received block blk_-5306443381473737599 of size 67108864 from /10.251.195.52
1	2	081110 103655 9020 INFO dfs.DataNode$PacketResponder: Received block blk_-4935573595350235765 of size 67108864 from /10.251.123.195
2	3	081110 022226 6281 INFO dfs.DataNode$DataXceiver: 10.250.19.16:50010 Served block blk_1078000656626961731 to /10.250.19.16
3	4	Jun 15 04:06:18 combo su(pam_unix)[21416]: session opened for user cyrus by (uid=0)
4	5	- 1121383222 2005.07.14 R01-M0-N0-I:J18-U11 2005-07-14-16.20.22.672549 R01-M0-N0-I:J18-U11 RAS APP FATAL ciod: Error creating node map from file /home/pakin1/sweep3d-2.2b/results/random1-8x32x32x2.map: Permission denied
5	6	17/06/09 20:10:48 INFO python.PythonRunner: Times: total = 41, boot = 15, init = 26, finish = 0
5	7	17/06/09 20:10:48 INFO python.PythonRunner: Times: total = 1072, boot = 856, init = 210, finish = 6
5	8	17/06/09 20:10:48 INFO python.PythonRunner: Times: total = 1078, boot = 851, init = 219, finish = 6

Log Parsing

Group ID (G_{parsed})	Group ID (G_{truth})	Log ID	Parsed Log Templates
1	1	1	Received block blk_<*> of size <*> from /<*>
1	1	2	Received block blk_<*> of size <*> from /<*>
2	2	3	<*>:<*> Served block blk_<*> to /<*>
3	3	4	session opened for user cyrus by (uid=0)
4	4	5	ciod: Error creating node map from file /home/pakin<*>/sweep<*>d-<*>.<*>b/results/random<*>x<*>x<*>.map: Permission denied
5	5	6	Times: total = <*>, boot = <*>, init = <*>, finish = <*>
6	5	7	Times: total = <*>, boot = <*>, init = <*>, finish = 6
6	5	8	Times: total = <*>, boot = <*>, init = <*>, finish = 6

Fig. 2. The example of log parsing and validating the result.
Note: Dynamic variables are highlighted in different colors: red for numbers, gray for ports, green for addresses, purple for block/task IDs, and orange for usernames. We can observe that some dynamic variables are either unresolved or incorrectly resolved.

'POST/webhook/order HTTP/1.0' 200 OK", and the message content is "*delete empty directory: /backend/upload/download/orders/49512/49512*". After appropriate log parsing, this log message can be split into the event template "*'POST <*>' 200 OK delete empty directory: <*>*", and the parameter values [*/webhook/order HTTP/1.0, /backend/upload/download/orders/ 49512/49512*]. Here, "<*>" denotes the position of a parameter.

Numerous studies in recent years have proposed various log parsers [13,17,25,27]. Log parsing techniques are broadly categorized into methods based on frequent pattern mining [28,29], clustering [30,31], and heuristics [17,32]. Among these, heuristic methods, which leverage the inherent characteristics of logs, have demonstrated superior performance in terms of accuracy [25] and time efficiency [26] compared to other approaches. Fig. 2 demonstrates the log parsing and grouping output produced by Drain, a state-of-the-art rule-based log parser widely adopted in log analysis research. It can be observed that due to the inability of parsers to fully recognize and replace dynamic variables in logs (e.g., in Logs 4, 5, 7, and 8, some entries omitted usernames and numbers, while some incorrectly parsed addresses.), logs that should have been grouped together (e.g., Logs 6, 7, and 8) are incorrectly divided into two groups. Prior research [25,33] has shown that this can affect the performance of subsequent anomaly detection models. Therefore, log parsing is a crucial step in log analysis, and the accuracy of log parsing will impact the performance of downstream tasks [34,35].

Recently, the rise of Large Language Models (LLMs), like ChatGPT [36], has led to their successful application in various software engineering tasks [37], demonstrating satisfactory performance [38]. Many studies [26,39] have demonstrated the feasibility of ChatGPT in log parsing. LogGPT [39] 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. Le et al. [26] utilized ChatGPT as a log parser for extracting log events and parameters, evaluating ChatGPT-3.5's performance in log parsing. DivLog [6] 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 [40] 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. Given this potential, we believe that leveraging ChatGPT for both log parsing and grouping offers high fault tolerance and can address the issue of incorrect grouping as depicted in Fig. 2, thereby enhancing the accuracy of pseudo-label assignment. Hence, utilizing them for log parsing and grouping presents a new possibility, but further research is needed to ascertain the capabilities of LLMs in this domain.

2.2. Log anomaly detection models

Anomaly detection has a long-standing history in the computer vision domain, where deep models such as Deep SVDD [41] directly optimize an anomaly detection objective rather than relying on surrogate tasks like generation or compression. As a theoretically grounded method, Deep SVDD has demonstrated strong performance on standard image benchmarks and in detecting adversarial examples. However, the application of anomaly detection techniques in the context of software logs has emerged relatively recently [1]. Due to the semi-structured and domain-specific nature of log data, methods originally designed for images or tabular data are often not directly transferable, prompting the development of specialized log anomaly detection models.

To ensure system reliability, logs are well-suited for tasks because they record key states during system operation. Currently, a substantial number of data-driven methods have been introduced that automatically detect anomalies in systems by analyzing software logs. Traditional machine-learning techniques can mine hidden information from various data sources. He et al. [24] conducted an evaluation of the performance of six machine learning-based anomaly log detection algorithms; however, these algorithms exhibited issues such as low efficiency and limited flexibility. Limitations encountered in machine learning for log data anomaly detection include the requirement for substantial amounts of labeled data to train models effectively, the challenge of generalizing across various system behaviors, susceptibility to adversarial attacks simulating normal patterns, and complexities in deciphering intricate model decisions. Furthermore, evolving systems can render acquired patterns obsolete, thus demanding ongoing retraining and adaptation.

To address these issues and tackle the common reliance of machine learning-based approaches on manually designed feature representations for log data, deep learning-based algorithms for log anomaly detection have been introduced. LogRobust [12] and HitAnomaly [20] are notable deep learning algorithms in the supervised learning category. In detail, Zhang et al. [12] introduced LogRobust, which represents log events as semantic vectors and uses an attention-based Bi-LSTM model to detect anomalies. Huang et al. [20] 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. Although supervised methods provide accurate diagnoses of system conditions, their main limitation is the need for a substantial volume of normal and abnormal data during the training phase. Therefore, researchers have also explored unsupervised and semi-supervised approaches [42,43]. Du et al. [13] employed DeepLog to treat log information as natural language sequences, autonomously learn log patterns from normal execution sequences, and identify anomalies by assessing whether incoming log events deviate from the predictions of a stacked LSTM model. NeuralLog [44] introduces a novel log-based anomaly detection method based on a Transformer classification model that eliminates the need for log preprocessing. Yang et al. [15,16] introduced a semi-supervised learning framework known as PLELog, which utilizes the HDBSCAN [45] clustering method for estimating labels probabilistically on unlabeled log sequences, addressing the challenge of limited labeling. LogBERT [21] 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. LogGPT [46] is a GPT-based framework for log anomaly detection that models log sequences as language and predicts the next log entry. It further incorporates reinforcement learning to fine-tune the model specifically for the anomaly detection task, achieving superior performance over prior methods. SemiRALD [7] integrates the strengths of RoBERTa and Bi-LSTM, achieving robust performance across multiple datasets and positioning itself as a more representative model.

Supervised methods typically achieve superior performance compared to other approaches, largely due to the availability of extensive labeled datasets (e.g., normal vs. abnormal logs) used during training. However, their practical applicability in industrial settings is limited, as they require substantial human effort to construct high-quality labeled datasets and are highly sensitive to label noise [1,47,48]. The manual labeling process is not only time-consuming but also prone to errors, which can negatively impact model performance [16]. In contrast, semi-supervised methods require only a small amount of labeled data, significantly reducing the reliance on manual annotation. By leveraging partial label information (e.g., feature-label correlations), these methods can better guide the training process compared to unsupervised approaches, thereby improving detection accuracy [49]. Furthermore, deep learning-based anomaly detection models often involve numerous hyperparameters, such as learning rate, hidden layer size, and the number of training iterations. The large search space formed by these hyperparameters makes manual tuning extremely challenging [50]. Traditional tuning methods typically depend on expert knowledge and trial-and-error strategies, which are not only inefficient but may also fail to identify optimal combination of hyperparameters.

2.3. Sequential model-based algorithm configuration (SMAC)

Sequential Model-Based Optimization (SMBO) [51] is a powerful technique for optimizing complex, expensive, and black-box objective functions. It iteratively builds a surrogate model, such as Gaussian Processes or Tree-structured Parzen Estimators, to approximate the objective function and guides the search for optimal solutions by maximizing acquisition functions like Expected Improvement. Bayesian Optimization Algorithm (BOA) [52], a popular variant of SMBO, leverages probabilistic models to capture uncertainty and systematically explore the search space, proving highly effective in hyperparameter tuning for neural networks [53]. Neural networks encompass various hyperparameters, which are frequently determined based on empirical knowledge [54]. Identifying the most appropriate parameters for the model through empirical methods can be challenging. These hyperparameters significantly impact the runtime and prediction accuracy of neural networks, making their optimization crucial.

BOA stands as one of the most well-known distribution estimation methods, amalgamating Bayesian networks with evolutionary algorithms to address nearly decomposable problems [55]. BOA is a global optimization algorithm based on Bayes' theorem, suitable for situations where the objective function is difficult to compute or has a high computational cost [56]. Its core idea is to guide the search process by building a probabilistic objective function model to find the parameter configuration that optimizes the objective function [57]. While this approach is highly effective for determining hyperparameters for deep learning models, it has not been widely explored in the field of DALD.

In BOA, we first assume that the objective function (i.e., points determined by different hyperparameters) follows a prior distribution [58]. Then, at each iteration, based on the current probabilistic model of the objective function, we select a point that is most likely to improve performance for evaluation. After evaluation, we incorporate the new observed result into the model and update the probabilistic model [59]. This process continues until a preset number of iterations is reached or other stopping criteria are met. The advantage of BOA lies in its ability to intelligently select the next evaluation point based on historical observations, thereby finding parameter configurations close to the optimal solution in fewer iterations [60,61].

Conventional Gaussian process-based BOA is limited to handling continuous or numerical variables in the parameter space [62]. However, in the realm of deep learning, numerous parameters are discrete or exhibit conditional relationships with each other [63]. Within the framework of Sequential Model-based Algorithm Configuration (SMAC) [64], Random Forest regression serves as a versatile surrogate model and an effective alternative to Gaussian Processes, particularly suitable for handling discrete and categorical variables. By using a model-based search strategy, SMAC approximates the objective function through ensemble predictions generated by the random forest, enabling efficient exploration and exploitation of the configuration space. Random forests consist of multiple regression trees, which are similar to decision trees and perform well with categorical input data. The overall prediction is made by computing the mean and variance of the outputs from each tree. These ensemble predictions serve as the output of the surrogate model, providing estimates of performance metrics for various hyperparameter configurations.

SMAC uses an acquisition function to balance exploration and exploitation, selecting configurations that are either expected to perform well or are uncertain and worth exploring. This approach necessitates an iterative process, enabling SMAC to enhance the predictive capability of the surrogate model by continuously acquiring new evaluation data when the initial data is limited. Each iteration helps identify configurations with better performance and explores uncertain yet potentially beneficial configurations, thereby improving overall optimization effectiveness through a balance between exploration and exploitation. The iterative process of SMAC starts with an initial set of configurations and their performance data. The surrogate model is trained on this data to capture the relationship between configurations and performance. SMAC then searches the configuration space guided by the surrogate model, proposing new configurations to evaluate. These evaluations update the surrogate model, refining its accuracy and improving the selection of subsequent configurations. This loop continues until a stopping criterion is met, such as a maximum number of evaluations or a satisfactory performance level. SMAC's ability to handle high-dimensional, mixed discrete and continuous configuration spaces makes it particularly effective for complex optimization tasks.

3. Methodology

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

3.1. Overview

SemiSMAC, as a semi-supervised DALD framework, requires only a few labeled log examples to start with. The whole framework consists of five phases: (1) Semi-supervised sample initialization: We select log

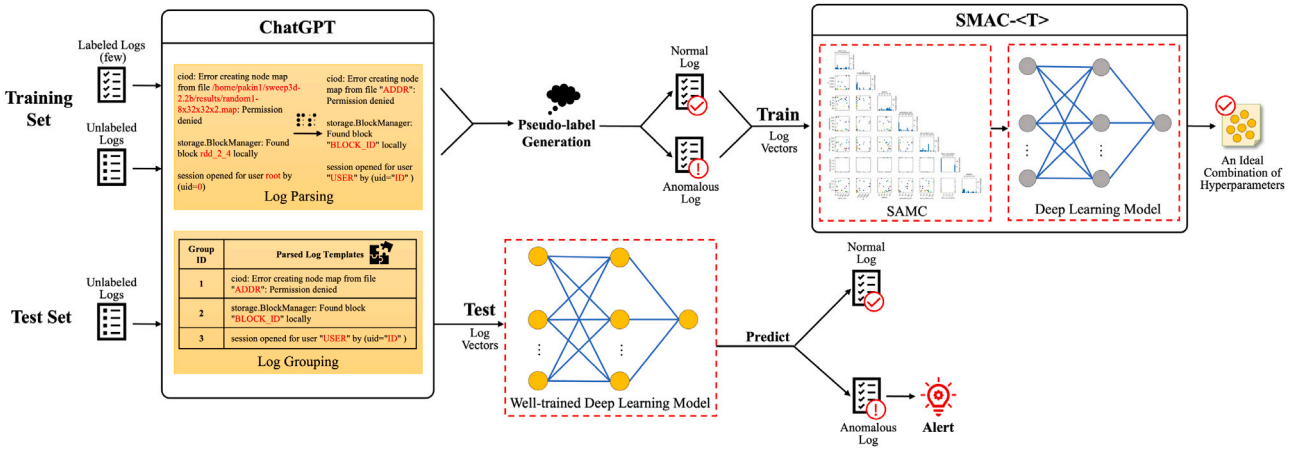


Fig. 3. Overview of the SemiSMAC-<T>framework.

sequences with different semantics from the original dataset through DBSCAN clustering, manually parse and label them, and use them as grouping prompts for ChatGPT. (2) Log processing based on ChatGPT: We leverage ChatGPT for automated log parsing and grouping. (3) Pseudo-label generation: Based on the grouping results from ChatGPT and the labels of the labeled logs in each group (normal or abnormal), we generate pseudo-labels for other unlabeled logs within the same group. (4) Log representation: We apply an embedding layer to transform discrete log entries into semantic vectors, enriching the log data representation. (5) Anomaly detection model building: We build SemiSMAC-LSTM to perform anomaly detection tasks. SMAC is a powerful framework for optimizing hyperparameters of neural networks. It replaces the surrogate model of Bayesian Optimization Algorithms (BOA) with a random forest, enabling the determination of discrete parameters within deep learning models while requiring less time. Fig. 3 illustrates a comprehensive overview of our proposed SemiSMAC-<T>framework. With SemiSMAC, we can automatically tune the hyperparameters to achieve optimal performance, thus overcoming the limitations of manual tuning and static configurations.

3.2. Semi-supervised sample initialization

Logs record software anomaly states and are generated in large quantities. For instance, Alibaba's cloud computing system produces approximately 30–50 GB (around 120–200 million lines) of logs per hour [65]. However, after parsing into log templates, many logs are semantically identical. During the preparation phase, we selected a tiny portion of labeled logs, both normal and abnormal, even though some previous studies [13,16,19] only labeled normal logs. We believe that many normal and abnormal logs share similar characteristics. By selecting representative logs from both categories, we can avoid abnormal logs being grouped with normal logs by ChatGPT due to similar features, enhancing the grouping accuracy of ChatGPT.

In collaboration with engineers, we utilized the DBSCAN clustering method to cluster the original logs and then manually selected some logs. DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [66] is a density-based clustering method that identifies points with high density as cluster cores and assigns other points to their respective clusters based on density connectivity. Its advantages include not requiring a pre-specified number of clusters and effectively identifying noise points [16]. The specific selection steps are as follows: (1) We select 3–5 log sequences from each cluster generated by DBSCAN clustering, and these log sequences are manually confirmed to have the same semantic features. (2) Manually parse the selected log sequences to obtain log templates, label the log templates as normal or abnormal, and then assign a group ID to different templates. Through these two steps, we obtained a diverse and representative labeled sample on a small scale.

3.3. Log processing based on chatgpt

In this section, we follow previous studies [26,39] to leverage ChatGPT for log parsing and also innovatively incorporate log grouping that is used for the pseudo-label assignment process into the above process, thereby alleviating the tolerance to error parsing and improving the pseudo-label assignment accuracy. Accordingly, we devised a comprehensive prompt template for ChatGPT to accomplish the above objective, outlined in Fig. 4, including (1) Task Description, (2) Replacement Rules, (3) Grouping Rules, (4) Situated Learning, and (5) Target Log.

Previous research [40] has shown that in-context learning further enhances ChatGPT's performance in log parsing. Both our Grouping Rules and Situated Learning approaches require providing ChatGPT with a small set of representative log samples for optimal results. To achieve this, we used cosine similarity to identify the log entries that are most similar. Specifically, each log entry is represented as a vector $\mathbf{x}$ in a high-dimensional space. The cosine similarity between two log vectors is defined as:

$$\text{Cosine Similarity}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|} \tag{1}$$

Where $\mathbf{x} \cdot \mathbf{y}$ denotes the dot product of vectors $\mathbf{x}$ and $\mathbf{y}$, and $\|\mathbf{x}\|$ and $\|\mathbf{y}\|$ represent their magnitudes (Euclidean norms). Both the new log entry $\mathbf{x}$ and each candidate sample $\mathbf{y}$ are normalized before computing the similarity. Once similarity scores are obtained for all candidate logs, they are ranked in descending order. Based on this ranking, the Grouping Rules method selects the top five most similar samples, while the Situated Learning approach selects the top 3. This selection ensures that ChatGPT receives log patterns that are most relevant, enhancing the accuracy of log grouping and pseudo-label assignment. Unlike Section 3.2, which focuses on constructing a diverse and representative labeled sample set, this stage aims to retrieve the most relevant examples from that set to support ChatGPT in log grouping and pseudo-labeling through in-context learning. Furthermore, each section of the prompt template was tailored according to the specific characteristics of the log sequences, and ChatGPT was employed to refine these prompts, ensuring adaptability to various log file formats.

Task Description: This section outlines the log parsing task assigned to ChatGPT, which includes an introduction, objectives, and expected results. Initially, we elucidated the log parsing task to ChatGPT, delineating the expected results. To accomplish this, we employ two prevalent types of ChatGPT instruction: indirect (i.e., Chain-of-Thought prompts) and direct instructions. Indirect instructions included phrases such as 'replace dynamic variables' or 'parse the content of the logs' to convey our objectives. Direct instructions were more precise, such as 'output the result directly, without explanation'.

Prompt Template (PT)

<Task Description> \n <Replacement Rules> \n <Grouping Rules> \n <Situated Learning> \n <Target Log> \n

You need to parse the content of the logs and replace the dynamic variables with templates, and then you need to determine which group the log belongs to after parsing. Please output the results directly without explanation.
You can refer to these replacement rules:
- Time Replacement: Replace all timestamps and specific times with "TIME".
- IP Address Replacement: Replace IP addresses with "IP".
(more rules…)
You can refer to these existing grouping rules:
- G1 ------ ciod: Error creating node map from file "ADDR": Permission denied
- G2 ------ L3 EDRAM error(s) (dcr "NUM") detected and corrected
(more rules…)
Here are some examples:
- Original Log: "[DEMO_LOG]"
- Replaced Log: "[TEMPLATE]"
Please parse the log template from this log message: "[LOG]"

Fig. 4. The log parsing prompt template of ChatGPT.

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 or transforming certain text to align with log parsing requirements. For instance, 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. We customize the replacement rules for log sequences based on the log formats of different datasets (e.g., the HDFS dataset contains unique block/task IDs). Refining prompts for ChatGPT enables it to effectively parse log formats from different datasets.

Grouping Rules: This section elaborates on enabling ChatGPT to learn grouping information in order to determine log template grouping after parsing. We manually grouped a tiny subset of selected logs and input this grouping information into ChatGPT, thereby providing it with grouping references.

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 SemiSMAC'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.

3.4. Pseudo-label generation

After ChatGPT completes log parsing and grouping, we obtain well-processed log templates and their corresponding groups. At this stage, the log templates represent the form after handling dynamic variables, so log templates within the same group are semantically identical. ChatGPT categorizes log sequences with semantically identical log templates into the same group, indicating that they share a common label. Because we previously selected and grouped a small portion of labeled logs, we can determine which group these parsed log templates might belong to based on whether they contain labeled logs. In other words, if a group contains known log-sequences, then new log-sequences in that group are more likely to share the same label (i.e. normal or abnormal). Otherwise, these log-sequences belong to new groups. Since these groups do not contain labeled logs, pseudo-labels cannot be generated and will be discarded. We assign pseudo-labels to groups containing known log sequences, where the pseudo-labels are determined based on the type of labeled log sequences within that group. Finally, we merge the labeled log samples (initially labeled logs) with the generated pseudo-labeled data (log templates grouped by ChatGPT) to create a new training dataset.

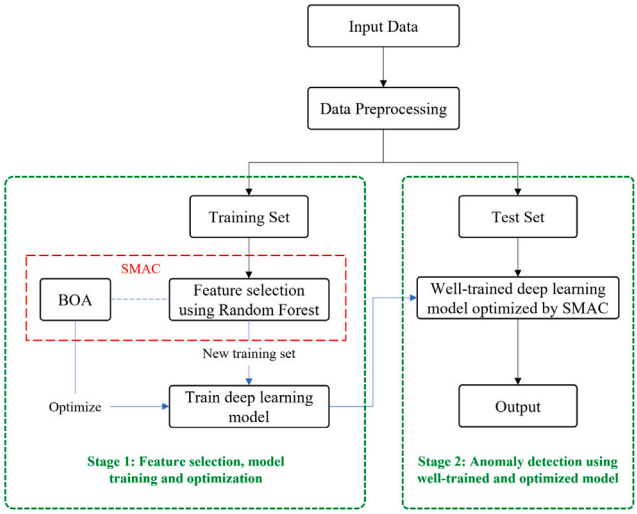


Fig. 5. Flowchart of the SemiSMAC model.

3.5. Log representation

Log sequences must be converted into numerical vectors before they can serve as input to deep learning models. In our study, we apply an embedding layer to transform discrete log entries into high-dimensional semantic vectors. This technique, inspired by word embeddings in natural language processing (NLP), allows the capture of intricate contextual relationships within software logs [67]. Each unique identifier in the logs, such as error codes or event types, is associated with a vector that is fine-tuned during the neural network's training process. These semantic vectors enrich the log data representation, facilitating more sophisticated pattern recognition and improving the model's ability to perform downstream tasks like anomaly detection and classification with higher accuracy.

3.6. Anomaly detection model building

The pseudo-labeled dataset constructed will serve as the training set for the anomaly detection framework SemiSMAC. We have integrated the SMAC method into the deep learning model to build an effective and efficient anomaly detection model. In this section, we introduce the optimization process using the SMAC algorithm. As shown in Fig. 5, the application of this software anomaly detection model comprises two stages: (1) Feature selection, model training and optimization. (2) Anomaly detection using well-trained and optimized model. The primary steps are as follows.

3.6.1. Stage 1: Feature selection, model training and optimization

Step 1: Construct a training set using initial variables as the input for the deep learning model, then feed this training set into the model. The input of the training set is:

$$Input = [\lambda_{i,j}] = \begin{bmatrix} \lambda_{1,1} & \lambda_{2,1} & \cdots & \lambda_{n,1} \\ \lambda_{1,2} & \lambda_{2,2} & \cdots & \lambda_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{1,m} & \lambda_{2,m} & \cdots & \lambda_{n,m} \end{bmatrix}$$

The output of the training set is:

$$Output = [y_j] = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

where n represents the total number of training samples, and $\lambda_{i,j}$ is the ith variable of the jth input sample. $y_j \in \{0, 1\}$ represents the label corresponding to the jth input sample, with a value of 1 denoting an abnormal label and 0 representing a normal label.

Algorithm 1 Sequential Model-based Algorithm Configuration (SMAC)

1: **Initialize:**
2: Randomly sample n_0 points to obtain n_0 results: $\{(x_1, f(x_1)), (x_2, f(x_2)), \ldots, (x_{n_0}, f(x_{n_0}))\}$;
3: Determine the maximum number of iterations N.
4: **Model Training:**
5: Train a random forest model using the initial n_0 points.
6: **while** $n \leq N$ **do**
7: Generate candidate points through large-scale random sampling. Use the trained random forest model to calculate the mean and standard deviation of the candidates, and compute the acquisition function $a_n(x)$ for each candidate.
8: Select the point with the highest $a_n(x)$ as x_{n+1}.
9: Evaluate the new point x_{n+1} to get $f(x_{n+1})$. *Note: This step is computationally intensive as it requires model training.*
10: Update the model with $n = n + 1$ points and retrain the random forest.
11: **end while**
12: Return the point with the maximum $f(x)$ among the current evaluated points.

Step 2: SMAC constructs the surrogate model of Bayesian Optimization Algorithm (BOA) using a Random Forest, as shown in Algorithm 1. Suppose there are initial sample points $\{(x_1, f(x_1)), (x_2, f(x_2)), \ldots, (x_n, f(x_n))\}$ *(Line 1–2)*. Here, x_n represents not sample data points from the dataset but rather hyperparameters of deep learning models, and $f(x_n)$ denotes the anomaly detection performance on the input ($[\lambda_{i,j}]$) of the training set under that model's hyperparameter configuration, typically measured by the accuracy or F1-score. Based on these points, we establish a random forest model to fit $f(x_n)$ *(Line 4–5)*. This process is similar to constructing a multidimensional Gaussian distribution formed by n points in Gaussian regression. In SMAC, the primary process of training a random forest as the surrogate model involves: first, sampling with replacement from the initial dataset to build multiple bootstrap sampling subsets; then, training decision trees on each subset by recursively splitting nodes through random feature selection. This process is repeated T times, generating T decision trees. Ultimately, by integrating all decision trees for prediction and uncertainty estimation, an efficient random forest surrogate model approximating the objective function is formed.

Step 3: By training a well-performing random forest and setting the iteration number N in advance *(Line 3)*, SMAC initiates extensive random sampling to determine the optimal candidate points *(Line 6–11)*. Leveraging the capability of random forests to handle discrete variables, we only need to add conditional restrictions in the parameter space to ensure that certain impossible scenarios are not sampled, or to set ranges or intervals for sampling points, allowing SMAC to conduct sampling. For a new point x, each tree in the random forest provides a prediction, and the average of all tree predictions gives the mean, while the standard deviation of predictions gives the standard deviation. Importantly, the advantage of random forests lies in their prediction complexity for a point x, where each tree's prediction complexity is only $O(\log N)$, significantly reducing prediction time compared to the complex matrix multiplication required in Gaussian regression processes [68].

During the iterative process, SMAC randomly samples multiple points, forming a new multidimensional normal distribution for the newly added point and the previous n points. For each candidate point, the random forest provides individual predictions from its constituent trees; the mean and standard deviation of these predictions are then computed to estimate the surrogate function's output and uncertainty. We employ Expected Improvement (EI) [69] as the acquisition function $a_n(x)$, selecting x_{n+1} based on the maximum value of $a_n(x)$, and computing $f(x_{n+1})$ for this point. Subsequently, adding this point to the initial set of n points creates a new sample point collection for the next iteration loop.

Step 4: After extensive iterative sampling, the final model will determine an optimal set (i.e. the maximum $f(x)$) of model hyperparameter configurations *(Line 12)*.

3.6.2. Anomaly detection using well-trained and optimized model

Feed the feature variable data from the test set into the well-trained deep learning model and determine the presence of anomaly logs based on the model's output.

After constructing an anomaly detection model using SemiSMAC, it can be deployed for real-time system monitoring. When a log sequence is received, SemiSMAC first processes the logs, obtains semantic vectors through log representation, and then uses the well-trained deep learning model to predict whether it is normal or abnormal. If an anomaly is detected, an alert will be promptly issued and sent to engineers for early diagnosis and resolution.

In general, the SMAC method has significant advantages over grid search [70] in hyperparameter optimization. By constructing a probabilistic model for intelligent search, SMAC can find better-performing hyperparameter combinations with fewer evaluations, significantly enhancing efficiency and reducing computational costs. Especially in high-dimensional hyperparameter spaces, SMAC can better balance exploration and exploitation, avoiding the inefficiencies caused by evaluating all possible combinations in grid search.

4. Experimental setup

4.1. Datasets

We employed four public datasets to evaluate the performance of our anomaly detection model: the HDFS [71], BGL [72], Thunderbird [72], and Spirit [72]. These datasets are labeled with log types and exhibit substantial differences in log formats, enabling us to assess the model's generalization performance. The specific distribution of logs is outlined in Table 1. For all downstream anomaly detection experiments, we split the entire dataset into 80% for training and 20% for testing. From the training portion, 20% is further set aside as a validation set. Consequently, 64% of the data is used for training, 16% for validation, and 20% for final testing. All splits are performed in chronological order to prevent data leakage across time.

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

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

Thunderbird dataset is an open log dataset collected from the Thunderbird supercomputer at Sandia National Labs (SNL). The dataset contains manually labeled normal and abnormal log messages. For computational feasibility, we use a subset of 10,000,000 continuous

Table 1
Number of log messages in each dataset.

Dataset	Category	Total logs	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%)
Thunderbird	Supercomputer	10,000,000	4,934 (0.49%)
Spirit	Supercomputer	5,000,000	764,500 (15.29%)

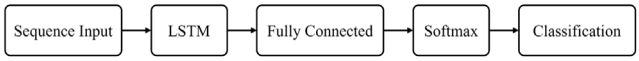


Fig. 6. A simple LSTM network architecture.

log messages from the original dataset of over 200 million messages. Among these, 4,934 (0.49%) are labeled as abnormal.

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

4.2. Model parameters

We utilize ChatGPT (GPT-4-turbo) via the OpenAI API to support log parsing and grouping in our framework. To reduce randomness and ensure reproducibility, we fix the prompt format and set the decoding parameters as follows: temperature = 0.1, top_p = 1.0, and n = 1. These settings encourage deterministic output while retaining sufficient expressiveness for log pattern interpretation. We use cosine similarity to retrieve the top-k most relevant labeled log examples, which are then inserted into the prompt as in-context examples to guide ChatGPT's response. All prompt templates are held constant throughout all experiments. To account for potential non-determinism in ChatGPT's responses (due to the use of a non-zero temperature), we repeat each experiment five times and report the average results to ensure robustness and reproducibility.

Log data typically exhibits temporal sequences characterized by events with intricate time intervals and patterns. LSTM (Long Short-Term Memory) is particularly suitable for anomaly detection tasks in log data due to its capability to effectively capture and retain long-term dependencies within sequential data. LSTM networks utilize LSTM units as fundamental components in their hidden layers and have proven effective in modeling complex sequences with long-term dependencies [73]. LSTM, equipped with its gated mechanisms including the input gate i_t, forget gate f_t, and output gate o_t, plays a crucial role in managing information flow over time. The forget gate controls the degree to which previous internal state information is retained, while the output gate regulates the flow of information from the current internal state to the external state. By learning and retaining essential features across extended sequences, LSTM can effectively discern anomalous patterns, even when these anomalies are non-contiguous in time or occur at different temporal scales. Consequently, LSTM offers more precise and reliable anomaly detection capabilities compared to traditional approaches relying solely on statistical features or simpler models.

Therefore, we select LSTM as the default anomaly detection model for SemiSMAC, and all subsequent experiments are based on SemiSMAC-LSTM. As illustrated in Fig. 6, a basic LSTM network architecture is employed, comprising an input layer, LSTM layer, fully connected layer, softmax layer, and output layer.

The input layer consists of 32 nodes, representing the input for the anomaly log text, which comprises 32 characters. The LSTM layer, also known as the hidden layer, contains multiple LSTM units. Following this is a fully connected layer with 2 output nodes, corresponding to the two classes for prediction. To predict class labels, the network concludes with a softmax layer and classification output layer. The output of the fully connected layer is classified by the softmax function into either 0 or 1. In the softmax layer, classification probabilities are calculated, and the classification output layer categorizes the test data into two groups: events and non-events. In this experiment, the activation function used is the Parametric Rectified Linear Unit (PReLU). PReLU is employed to nonlinearly activate the output features of the convolutional layer. Compared to the classical ReLU function, PReLU enhances model fitting with minimal additional computational cost and a reduced risk of overfitting. The mathematical formula for PReLU is:

$$f_{PReLU(y)} = \begin{cases} y & y > 0 \\ \alpha y & y \leq 0 \end{cases} \tag{2}$$

where y represents the input to a neuron, and α is a small positive parameter learned during training. This allows for a small, positive gradient when the unit is inactive, rather than zero, as in the case of the standard ReLU activation function. PReLU generalizes ReLU by introducing adaptability to the negative part of the function. The challenge of anomaly detection in software logs is typically framed as a binary classification problem. Therefore, A binary cross-entropy loss function is chosen as the cost function, defined as:

$$loss = -[y \log(y^*) + (1 - y) \log(1 - y^*)] \tag{3}$$

Where y represents the actual label, and y^* represents the output of the fully connected layer. The binary cross-entropy loss function quantifies the proximity between the actual output and the expected output.

The weights and biases of the LSTM network are initialized randomly, and the Adaptive Moment Estimation (Adam) algorithm is employed to optimize the internal parameters of the LSTM network during the training process.

Similar to the LSTM, the SemiSMAC-LSTM model incorporates certain parameters present in the LSTM model. The difference lies in the use of BOA within the SMAC method to determine hyperparameters such as *Batch Size*, *Learning Rate*, *Epochs*, *Embedding Dimension*, *Hidden Layer Size*, *Number of Layers*, and *Dropout Rate*. As shown in Table 2, establishing specific parameter ranges for optimization is crucial to achieve effective optimization results and ensure the feasibility of the optimized hyperparameters.

Table 2
The ranges of hyperparameters to be optimized.

Hyperparameters	Value/Range
Batch Size	32, 64, 128, 256
Learning Rate	[0.0001, 0.01]
Epochs	20, 40, 60, 80, 100
Embedding Dimension	32, 64, 128
Hidden Layer Size	32, 64, 128
Number of Layers	1, 2
Dropout Rate	[0, 0.5]

4.3. Research questions

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

- **RQ1: What are the optimal hyperparameters of the model under different datasets?**
  Neural networks encompass multiple hyperparameters. In traditional LSTM models, hyperparameter settings are often based on empirical knowledge, making it challenging to identify the most suitable parameters through such heuristic methods. This study selects the HDFS, BGL, Thunderbird, and Spirit datasets and determines the optimal values for seven hyperparameters of the LSTM model using the SMAC method: *Batch Size*, *Learning Rate*, *Epochs*, *Embedding Dimension*, *Hidden Layer Size*, *Number of Layers*, and *Dropout Rate*. We set the number of iterations to 20. The SMAC method first utilizes BOA to select a combination of hyperparameters, then evaluates the importance of initial variables through a random forest, and records the classification accuracy of the training dataset. The optimal hyperparameter combination is determined through iterative refinement. This method can uncover the optimal parameter combinations for anomaly detection models across different datasets to achieve the best anomaly detection performance.
- **RQ2: What is the performance of SemiSMAC-LSTM against existing DALD approaches?**
  In this study, we conducted a comparison between SemiSMAC-LSTM and six benchmark models across four datasets: HDFS, BGL, Thunderbird, and Spirit. These benchmark models encompass unsupervised learning-based models (DeepLog), supervised learning-based models (HitAnomaly, NeuralLog, LogRobust), and semi-supervised learning-based models (PLELog, LogBERT). Detailed descriptions of these benchmark models can be found in Section 4.4. The experimental results of DeepLog are taken from the LogBERT paper, while results for other baseline models are obtained from their respective original publications or official implementations.
  For fairness, we adopted the best-known hyperparameter configurations as reported in those sources. Notably, models such as HitAnomaly, NeuralLog, LogRobust, and LogBERT primarily rely on manual hyperparameter tuning guided by validation performance, often using empirical settings and limited sensitivity analysis to adjust parameters like window size, learning rate, hidden dimensions, and loss weights. In contrast, PLELog, and DeepLog employ grid search to explore combinations of key hyperparameters based on validation accuracy. To further reduce potential bias, we conducted limited grid search for several baselines (e.g., DeepLog and PLELog) across different datasets. However, these alternative configurations did not yield better performance than those reported in the original works. Therefore, for consistency and reproducibility, we retained the original settings.
- **RQ3: How does ChatGPT perform in log processing?**
  To verify the accuracy of ChatGPT's log processing, we selected all datasets provided on the LogHub platform and compared the results with state-of-the-art rule-based parsers. Log parsers such as Drain [17], Logram [13], AEL [25], Spell [27], SPINE [74], UniParse [75], and LogPPT [26] employ various techniques to transform unstructured log data into meaningful information: Drain [17] uses a hierarchical tree for scalability, Logram [13] applies probabilistic models for adaptability, AEL [25] leverages NLP for semantic labeling, Spell [27] groups similar messages through clustering, SPINE [74] normalizes logs with syntactic pattern recognition, UniParse [75] introduces a unified framework combining sequence labeling and tree-based methods, and LogPPT [26] leverages adaptive random sampling and template-free prompt tuning with a virtual label token "PARAM" to enable efficient and accurate few-shot log parsing using a RoBERTa-based language model. These diverse methods enhance log analysis by addressing different system monitoring challenges. For log processing, we employ the GPT-4-turbo version and access it via the official API[1] provided by OpenAI.
- **RQ4: How does SemiSMAC-LSTM perform on low-resource datasets?**
  In practical industrial scenarios, such as factory production lines and navigation robot environments, limited log information is often generated, posing a significant challenge for efficient anomaly detection with minimal data. To systematically evaluate the performance of SemiSMAC-LSTM on low-resource datasets, we selected two representative datasets: HDFS and BGL. These datasets originate from different systems — HDFS from a distributed file system and BGL from a supercomputer — resulting in greater variability between the logs. We then selected various scales of the HDFS and BGL datasets, ranging from 0.1% to 15%. Specifically, we experimented with 0.1%, 1%, 5%, 10%, and 15% of the data to verify our approach.
  To ensure representative and balanced evaluation, we selected HDFS and BGL — each representing a distinct system category — based on their log diversity, anomaly distribution, and suitability for low-resource analysis. Among the three supercomputer datasets, BGL was chosen due to its higher log format diversity and more balanced anomaly ratio (7.34%) compared to Thunderbird (0.49%) and Spirit (15.29%), making it better suited for evaluating model performance in low-resource and generalization scenarios.
- **RQ5: How well does the SMAC method generalize across different models, and how efficient is it in tuning time?**
  Traditional deep learning models such as Long Short-Term Memory (LSTM) [73], Gated Recurrent Units (GRU) [76], and Convolutional Neural Networks (CNN) [77] have shown strong potential in log anomaly detection. However, their performance often hinges on carefully tuned hyperparameters, which may vary significantly across different datasets. To evaluate the generalizability and efficiency of SMAC in improving deep learning-based anomaly detection, we integrate SMAC into several standard architectures and compare their performance against counterparts tuned using conventional grid search.
  Specifically, we evaluate twelve models on the HDFS and BGL datasets: SemiSMAC-LSTM, LSTM, SemiSMAC-GRU, GRU, SemiSMAC-CNN, CNN, SMAC-DeepLog, DeepLog, SMAC-LogRobust, LogRobust, SMAC-PLELog, and PLELog. These include both classic neural models and three representative approaches from different learning paradigms—DeepLog (unsupervised), LogRobust (supervised), and PLELog (semi-supervised). For all models, we tune the following hyperparameters: batch size, learning rate, number of epochs, embedding dimension, hidden size, number of layers, and dropout rate. Each model is evaluated under two tuning strategies: grid search and SMAC-based Bayesian optimization. We report both the anomaly detection performance (F1-score) and the time required for hyperparameter search under the same computational conditions. This comprehensive evaluation enables us to assess the effectiveness and efficiency of SMAC in optimizing a wide range of models, highlighting its potential as a general-purpose tuning framework for log anomaly detection.

4.4. Benchmark models

In this section, we present benchmark models for comparison with SemiSMAC-LSTM 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. Notably, we include DeepLog as a representative unsupervised baseline due to its widespread adoption and strong influence in the log anomaly detection community. This choice enables a meaningful and consistent

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

comparison across different learning paradigms — unsupervised, supervised, and semi-supervised — particularly in RQ4 and RQ5, where we evaluate model robustness in low-resource scenarios and investigate the generalizability and efficiency of SMAC-based hyperparameter optimization across diverse detection frameworks.

DeepLog [13] 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 [20] 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 [44] is a supervised classifier model built on the Transformer architecture and incorporates a tokenizer. This strategy empowers the model to learn unique features from raw log data directly without the intermediate step of log parsing, which frequently results in information loss. By integrating a mixture of normal data from the target system and abnormal data from another system, NeuralLog enhances its anomaly detection capabilities, rendering it versatile and efficient for real-world scenarios.

LogRobust [12] is a supervised model built on an attention-based Bi-LSTM architecture. It employs a specialized log parser to preprocess logs, generating TF-IDF and word semantic vectors to extract log features. Both normal and abnormal logs contribute to the training process. The incorporation of attention mechanisms bolsters the model's capacity to pinpoint significant anomalies in log data, thereby enhancing detection accuracy and reliability across various operational contexts.

PLELog [15,16] introduces a semi-supervised log-based anomaly detection technique that employs probabilistic label estimation to efficiently handle unlabeled log data. This innovative approach integrates semantic information from log events into fixed-length vectors, employs HDBSCAN for clustering these vectors, and utilizes an attention-based GRU network to refine the detection process. Empirical evaluations conducted on datasets such as HDFS and BGL showcase PLELog's robust anomaly detection capabilities, thereby diminishing the necessity for extensive manual labeling.

LogBERT [21] 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

4.5.1. Evaluation metrics for log parsing

Following prior studies [23,25,33,75,78], we use two metrics to evaluate the effectiveness of LLMs in log parsing: Group Accuracy (GA) and Parsing Accuracy (PA).

Group Accuracy (GA) [25] does not directly evaluate the correctness of the parsed logs. Instead, GA assesses the accuracy of the automatically grouped logs (e.g., G_{parsed} shown in Fig. 2). GA is calculated as the ratio between the number of correctly grouped logs and the total number of logs. Specifically, GA first groups logs based on the parsed logs (i.e., generated by a log parser), and then compares the resulting groups with those from G_{truth}. For instance, the log parsing result in Fig. 2 has a GA of 5/8. Once the raw logs are parsed, they are grouped into different groups. We can see that Logs 1 and 2 are grouped together, Logs 7 and 8 are grouped together, and Logs 3, 4, 5, and 6 form a separate group by themselves. The G_{parsed} for groups 1 to 4 match the grouping results obtained by G_{truth}. However, Logs 6 to 8 form two groups (G_{parsed} 4 and 5) when using the parsed log template, whereas there is only one group (G_{truth} 5) if using the log template from the ground truth. As a result, the GA for this example is 5/8.

Prior studies [23,33,75] have highlighted the limitations of GA. For instance, even if the logs are perfectly grouped with a GA of 100%, the parsed log template may not fully match the ground truth template due to unresolved or incorrectly resolved dynamic variables in the parsed templates. As a result, GA cannot indicate whether the logs are parsed correctly. Furthermore, if Logs 4 and 5 are not parsed correctly but are still grouped in a cluster, GA will still be 100%.

Parsing Accuracy (PA) [75] requires the log template to match the ground truth template exactly. For example, the log parsing result in Fig. 2 has a PA of 1/2 (lower than the GA of 5/8) because there are unresolved or incorrectly resolved dynamic variables in Logs 4, 5, 7, and 8. Hence, PA is a stricter metric than GA and aligns more closely with practical requirements [33,75,78]. Prior studies also found that parsing the dynamic variables can help downstream log analysis tasks [79–81], further demonstrating the importance of PA over GA.

4.5.2. Evaluation metrics for log anomaly detection

Log anomaly detection is treated as a binary 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.

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

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.

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

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.

$$F1 - Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{6}$$

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

Table 3
An Ideal Combination of Hyperparameters Optimized by SMAC across Four Datasets.

Hyperparameters	Values under the HDFS dataset	Values under the BGL dataset	Values under the Thunderbird dataset	Values under the Spirit dataset
Batch Size	32	256	32	64
Learning Rate	0.00919	0.00192	0.00465	0.00665
Epochs	37	91	43	21
Embedding Dimension	43	92	34	37
Hidden Layer Size	126	52	53	85
Number of Layers	1	1	1	2
Dropout Rate	0.29178	0.22292	0.48495	0.28385

(False Negative) is the count of anomalous log sequences that the model failed to detect.

In this paper, we use the F1-Score as the primary metric for evaluating the models. 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.

5. Results and analysis

5.1. Determination of model hyperparameters (RQ1)

The initial step in the SemiSMAC-LSTM model involves determining the optimal hyperparameter values for the LSTM model using the SMAC method across four datasets: HDFS, BGL, Thunderbird, and Spirit. A total of 20 iterations were set for this process. The combination of hyperparameters is initially chosen by BOA, and the random forest is used to assess the significance of initial variables. The classification accuracy of the training dataset is recorded, and the optimal set of hyperparameters is determined through iterative steps. Ultimately, the ideal combination of hyperparameters is achieved at the 16th iteration for the HDFS dataset, the 12th iteration for the BGL dataset, the 15th iteration for the Thunderbird dataset, and the 19th iteration for the Spirit dataset, as shown in Table 3. It is worth noting that BOA operates swiftly as it establishes an alternative hyperparameter search space model and conducts its search within this dimension, rather than the original actual hyperparameter space. In conclusion, the most ideal combination of hyperparameters is preserved, representing the LSTM model parameters derived through the SMAC method.

5.2. Effectiveness of semismac-LSTM (RQ2)

In this section, we compare SemiSMAC-LSTM with six benchmark models across four datasets — HDFS, BGL, Thunderbird, and Spirit — as shown in Table 4. To summarize, SemiSMAC-LSTM consistently outperforms state-of-the-art ALD models across diverse datasets, including three supervised learning-based models. This highlights its remarkable generalization ability to heterogeneous log formats and superior anomaly detection performance.

As can be seen from Table 4, all semi-supervised and fully supervised models outperform the unsupervised model DeepLog, emphasizing the critical importance of leveraging historical samples in anomaly log detection tasks. However, discrepancies in log formats across the four datasets make it challenging for log parsers to generalize effectively, resulting in inconsistent performance for most ALD models. For instance, HitAnomaly and LogRobust 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. A similar pattern is observed for the Thunderbird and Spirit datasets, where LogRobust, PLELog, and LogBERT exhibit significant performance differences, highlighting the challenge of generalization.

In contrast, SemiSMAC-LSTM demonstrates stable information extraction capabilities across logs with different formats. Notably, as a semi-supervised approach, SemiSMAC-LSTM outperforms all three supervised learning models — LogRobust, NeuralLog, and HitAnomaly — by 1.5%, 0.5%, and 0.5% on the HDFS dataset, respectively; by 15.3%, 0.3%, and 6.3% on the BGL dataset; by 30%, 3%, and 10% on the Thunderbird dataset; and by 4%, 7.5%, and 14.9% on the Spirit dataset. These results highlight the competitive advantage of SemiSMAC-LSTM in handling heterogeneous log formats and achieving superior anomaly detection performance.

Finding: SemiSMAC-LSTM consistently outperforms current state-of-the-art ALD approaches across diverse datasets, demonstrating stable and outstanding performance in all datasets, which highlights its impressive generalization ability across heterogeneous log formats and superior anomaly detection capabilities.

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

In this section, we evaluate ChatGPT's performance in log parsing and grouping. Experiments were conducted on a benchmark dataset provided by the LogHub[2] [5] platform. This dataset contains logs from 16 open-source systems and is widely used to evaluate and compare the accuracy of log parsers [17,23,27]. The studied systems span various domains, including distributed systems, operating systems, mobile systems, supercomputers, server applications, and standalone software. Each system includes 2,000 log messages along with their respective log templates and parameters, serving as the ground truth for evaluating log parsers. It is worth noting that the dataset in Section 4.1 is sourced from the HDFS and BGL datasets provided by the LogHub platform. We assess the performance of each log parser through Grouping Accuracy (GA) and Parsing Accuracy (PA). Table 5 shows both the grouping accuracy (GA) and parsing accuracy (PA) of the state-of-the-art rule-based methods and our designed ChatGPT Parser.

Overall, due to the design of tailored prompt components for ChatGPT—namely the *replacement rules*, *grouping rules*, and *situated learning*—which are customized for each dataset, our approach achieves significantly superior performance. Specifically, for the GA metric, our method achieved the best results on 11 out of 16 datasets, with an average score 9.8% higher than the widely used rule-based parser, Drain. For the PA metric, the evaluation standard is more stringent: log sequences are considered failed if even a single dynamic variable is not successfully identified. This results in existing parsers averaging a high score of 0.861 in GA but only 0.335 in PA. With our prompt-specific approach, ChatGPT excels at distinguishing dynamic variables in log messages, successfully extracting and processing them, ultimately achieving an average PA score of 0.947, a 54.3% improvement over other parsers. Fig. 7 illustrates some log templates generated by our method and state-of-the-art methods after log parsing.

From Fig. 7, we can see that rule-based log parsers struggle to identify many specific dynamic log variables (such as domain URLs,

[2] https://github.com/logpai/loghub

Table 4
The performance of different models on four datasets.

Model	HDFS			BGL			Thunderbird			Spirit		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Unsupervised learning-based												
DeepLog	0.884	0.695	0.773	0.897	0.828	0.861	0.484	0.895	0.628	0.438	**1**	0.609
Supervised learning-based												
HitAnomaly	**1**	0.970	0.980	0.950	0.900	0.920	0.950	0.810	0.880	0.920	0.750	0.840
NeuralLog	0.960	**1**	0.980	0.980	0.980	0.980	0.920	**0.990**	0.950	0.919	0.908	0.914
LogRobust	0.980	0.960	0.970	0.910	0.770	0.830	0.610	0.780	0.680	0.985	0.915	0.949
Semi-supervised learning-based												
PLELog	0.950	0.963	0.957	0.965	**0.999**	0.982	0.864	0.941	0.901	0.931	0.767	0.841
LogBERT	0.870	0.781	0.823	0.894	0.923	0.908	0.962	0.965	0.963	0.862	0.807	0.834
SemiSMAC-LSTM	0.981	0.989	**0.985**	**0.980**	0.989	**0.983**	**0.984**	0.977	**0.980**	**0.987**	0.991	**0.989**

Table 5
A comparison of the grouping accuracy (GA) and parsing accuracy (PA) for the state-of-the-art methods and our designed ChatGPT parser (the last column).

	Drain		Logram		AEL		Spell		SPINE		UniParser		LogPPT		*Our Parser*	
	GA	PA	GA	PA	GA	PA	GA	PA	GA	PA	GA	PA	GA	PA	GA	PA
Distributed systems																
HDFS	0.998	0.355	0.930	0.005	0.998	0.625	**1**	0.301	0.866	0.499	**1**	0.948	0.721	0.943	**1** ± 0	**0.974** ± 0.03
Hadoop	0.948	0.269	0.451	0.113	0.869	0.262	0.778	0.113	0.950	0.279	0.691	0.889	0.483	0.666	**0.962** ± 0.02	**0.984** ± 0.03
Spark	0.920	0.360	0.282	0.259	0.905	0.360	0.905	0.321	0.925	0.337	0.854	0.795	0.476	0.952	**0.998** ± 0.02	**0.991** ± 0.01
Zookeeper	0.967	0.497	0.724	0.474	0.921	0.496	0.964	0.452	0.989	0.502	0.988	0.988	0.967	0.845	**1** ± 0	**0.993** ± 0.01
OpenStack	0.733	0.019	0.326	0	0.758	0.019	0.764	0	0.384	0.011	**1**	0.516	0.534	0.406	**1** ± 0	**0.926** ± 0.03
Super computers																
BGL	**0.963**	0.342	0.587	0.125	0.957	0.344	0.787	0.197	0.923	0.376	0.918	0.949	0.245	0.938	0.961 ± 0.01	**0.978** ± 0.02
HPC	0.887	0.636	0.911	0.643	0.903	0.678	0.654	0.530	0.945	0.667	0.777	0.941	0.782	**0.997**	**0.975** ± 0.02	0.980 ± 0.02
Thunderbird	**0.955**	0.047	0.554	0.004	0.941	0.036	0.844	0.027	0.665	0.051	0.579	0.654	0.564	0.401	0.950 ± 0.03	**0.912** ± 0.02
Operating systems																
Windows	**0.997**	0.462	0.694	0.141	0.690	0.153	0.989	0.004	0.684	0.151	0.731	0.869	0.991	0.915	0.993 ± 0.01	**0.979** ± 0.03
Linux	0.690	0.184	0.361	0.124	0.405	0.174	0.152	0.088	0.545	0.108	0.285	0.164	0.205	0.168	**0.898** ± 0.01	**0.935** ± 0.01
Mac	**0.787**	0.218	0.568	0.169	0.764	0.169	0.757	0.033	0.761	0.204	0.737	0.688	0.544	0.390	0.785 ± 0.02	**0.705** ± 0.02
Mobile systems																
Android	0.831	0.548	0.742	0.278	0.773	0.393	0.863	0.150	0.938	0.181	0.505	0.607	0.709	0.794	**0.885** ± 0.02	**0.913** ± 0.01
HealthApp	0.780	0.231	0.267	0.112	0.568	0.163	0.639	0.152	0.983	0.446	0.461	0.817	**0.989**	**1**	0.905 ± 0.01	0.885 ± 0.03
Server applications																
Apache	**1**	0.694	0.313	0.007	**1**	0.694	**1**	0.694	**1**	0.276	0.948	0.942	0.786	0.948	**1** ± 0	**1** ± 0
OpenSSH	0.789	0.508	0.611	0.298	0.537	0.246	0.556	0.191	0.676	0.253	0.275	0.289	0.277	0.654	**1** ± 0	**1** ± 0
Standalone software																
Proxifier	0.527	0	0.504	0	0.495	0.495	0.527	0.478	0.049	0.016	0.509	**1**	0.989	**1**	**1**± 0	**1** ± 0
Average	0.861	0.335	0.551	0.172	0.780	0.331	0.761	0.233	0.767	0.272	0.704	0.731	0.580	0.751	**0.957**	**0.947**

Note: The highest values for GA and PA for each system are highlighted in **bold**. The results for Drain and Logram are based on the evaluation conducted by Khan et al. [33] on the corrected log dataset, while the results for the other rule-based parsers are from the research by Le et al. [79] and Jiang et al [40].

APIs, addresses, block/task IDs, usernames, etc.) due to significant variations in their occurrences in log data. This leads to lower PA scores. In contrast, ChatGPT effectively identifies and processes these variables after learning from our few-shot prompts. For instance, ChatGPT correctly identified address variables in the second and fifth log messages, as well as block IDs, usernames, and UIDs in the fourth and sixth log messages, which were not recognized by the rule-based parser. However, during the processing stage, ChatGPT can also be affected by noise. For example, in the first log message, it may occasionally replace numbers with "ID" or "CORE_ID" instead of accurately with "NUM". Even with more detailed prompts, this issue cannot be entirely avoided.

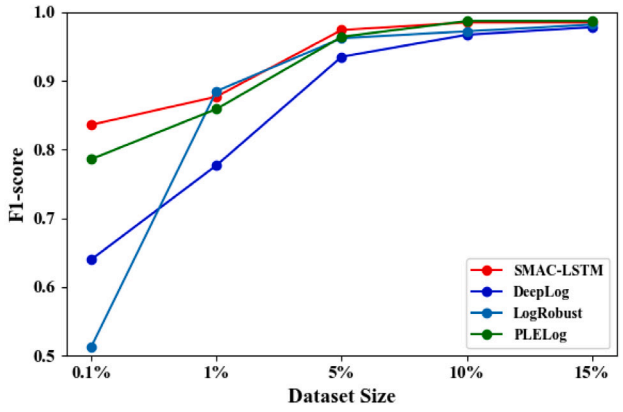
Finding: By customizing replacement rules, grouping rules, and situated learning, our ChatGPT-based parser achieves higher average GA and PA scores across 16 datasets compared to other state-of-the-art parsers, highlighting its effectiveness.

5.4. Exploration in low-resource scenarios (RQ4)

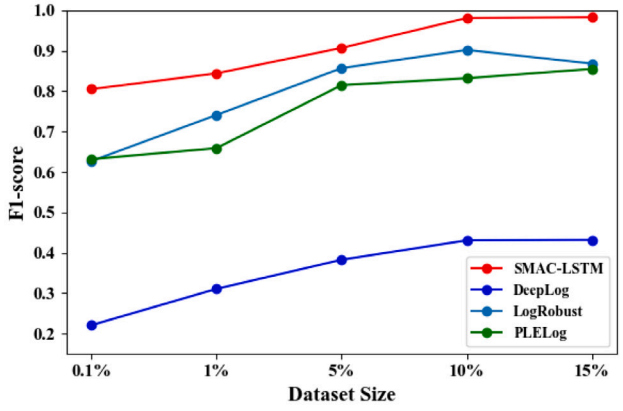
In real-world environments, there are currently numerous small-scale datasets where traditional anomaly detection models may not perform well due to limited data availability. To evaluate the effectiveness of SemiSMAC-LSTM in such scenarios, we constructed low-resource settings by selecting various proportions of the HDFS and BGL datasets, specifically 0.1%, 1%, 5%, 10%, and 15%, using chronological sampling to preserve the temporal order of events. For comparison, we selected representative models from different learning paradigms: DeepLog (unsupervised), LogRobust (supervised), and PLELog (semi-supervised). All experiments were conducted independently under consistent conditions. Fig. 8 illustrates the F1-scores of each model on the two datasets. For fair comparison, the baseline models were tuned using grid search, with hyperparameters selected based on their original implementations or adjusted within a reasonable range when necessary. In contrast, our model leveraged SMAC for automated hyperparameter optimization tailored to each dataset.

Original Log Sequence	Our ChatGPT Parser	State-of-the-art Parsers
generating core.385	generating core."CORE_ID" generating core."NUM"	generating core.<*>
ciod: Error creating node map from file /home/pakin1/sweep3d-2.2b/results/random1-8x32x32x2.map: Permission denied	ciod: Error creating node map from file "ADDR": Permission denied	ciod: Error creating node map from file /home/pakin<*>/sweep<*>d-<*>.<*>b/results/random<*>x<*>. map: Permission denied
Received block blk_-9671458564739018O4 of size 67108864 from /10.250.6.191	Received block blk_"NUM" of size "NUM" from "IP"	Received block blk_<*> of size <*> from /<*>
storage.BlockManager: Found block rdd_2_4 locally	storage.BlockManager: Found block "BLOCK_ID" locally	storage.BlockManager: Found block rdd_2_4 locally
ciod: Error loading /g/g24/buber/Yunsic/BlueGene/partad.develf/taddriver.32.exe: program image too big, 361544528 > 266076160	ciod: Error loading "ADDR": program image too big, "NUM" > "NUM"	ciod: Error loading /g/g<*>/buber/Yunsic/BlueGene/partad.develf/taddriver.<*>.exe: program image too big, <*> > <*>
session opened for user root by (uid=0)	session opened for user "USER" by (uid="ID")	session opened for user root by (uid=0)

Fig. 7. Examples of log templates after log parsing.



(a) HDFS



(b) BGL

Fig. 8. The performance of models on low-resource datasets.

As can be seen from Fig. 8, 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% and 3.5%–20.5% in terms of F1-score. As for the fully supervised approach, namely LogRobust, its also performance decreases by 21.6% at most. For SemiSMAC-LSTM, the anomaly detection performance slightly decreases in low-resource datasets, but the reduction is extremely limited. Even when the data scale is reduced to 0.1%, the F1-score on the HDFS and BGL datasets only decreases by 14.9% and 17.8%, respectively. Furthermore, as the data scale reaches 10%, its F1-score stabilizes, remaining equivalent to that at 100% scale, indicating the highly robust nature of SemiSMAC-LSTM in anomaly detection.

In comparison, the other three models exhibit unstable anomaly detection performance in low-resource datasets, especially at a data scale of 0.1%. For instance, as shown in Fig. 8(a), with the HDFS dataset at a 0.1% data scale, SemiSMAC-LSTM's F1-score improves by 19.6%, 5.0%, and 17.8% compared to DeepLog, PLELog, and LogRobust, respectively. As the dataset scale increases, all models experience significant improvements in F1-scores when the scale reaches 1%, with respective increases of 15.8% (DeepLog), 10.55% (PLELog), 7.7% (LogRobust), and 4.1% (SemiSMAC-LSTM). Once the dataset scale reaches 10%, all models achieve stable F1-scores.

As shown in Fig. 8(b), with the BGL dataset at a 0.1% data scale, SemiSMAC-LSTM achieves an F1-score of 0.805, surpassing PLELog and LogRobust by 17.3% and 17.8%, respectively. Even LogRobust, trained under full supervision, struggles to learn log sequence features in extremely limited data conditions without the most suitable hyperparameter combinations for the attention-based Bi-LSTM architecture. As the dataset size increases, the F1-scores of all four models improve, but LogRobust shows some fluctuations. At a dataset size of 15%, LogRobust's F1-score is 3.4%, which is lower than that at the data size of 10%. Additionally, at a dataset size of 15%, SemiSMAC-LSTM achieves an F1-score of 0.983, marking a 26.5% average increase over the other three models.

It is worth noting that due to the variation in dataset scales, the hyperparameters determined by SemiSMAC-LSTM also change accordingly. Table 6 presents the optimal hyperparameter configurations optimized by SMAC at the 0.1% scale of the HDFS and BGL datasets. A comparison between Tables 3 and 6 reveals that, for different scales and types of datasets, SemiSMAC-LSTM can consistently determine the optimal hyperparameter combination through iterative processes. Furthermore, SemiSMAC-LSTM's performance in anomaly detection with the determined optimal hyperparameters outperforms other models, demonstrating its robustness in handling imbalanced and low-resource datasets. This also implies that SemiSMAC-LSTM's capability in identifying anomalous logs is reliable.

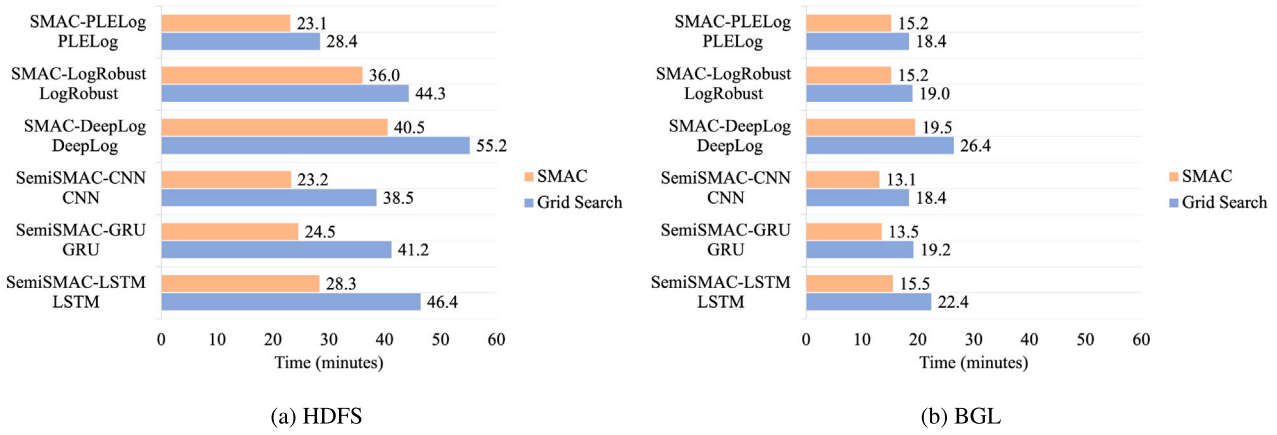


Fig. 9. Comparison of hyperparameter search time between SMAC and grid search across two datasets.
Note: Due to the inherent variability in log anomaly detection tasks, we consider an F1-score difference greater than 0.005 between the two hyperparameter tuning strategies to be indicative of a significant performance gap. In our experiments, this threshold was met in multiple model-dataset combinations. Notably, for LogRobust, the F1-scores achieved on the HDFS and BGL datasets were 0.982 and 0.859, respectively—both clearly higher than the original results reported using empirical hyperparameter settings (0.970 on HDFS and 0.830 on BGL).

Table 6
The optimal hyperparameter configuration for SMAC at the 0.1% scale of the HDFS and BGL datasets.

Hyperparameters	Values under the HDFS dataset	Values under the BGL dataset
Batch Size	32	128
Learning Rate	0.00686	0.00993
Epochs	63	24
Embedding Dimension	39	52
Hidden Layer Size	78	126
Number of Layers	1	1
Dropout Rate	0.45466	0.31586

Finding: SemiSMAC-LSTM consistently outperforms models from different learning paradigms and demonstrates robustness across diverse data scales, showing superior performance even on datasets as small as 0.1%. Moreover, its consistent and superior performance on both datasets highlights its generalization capabilities.

5.5. Exploring the generalization and tuning efficiency of SMAC (RQ5)

We evaluate twelve models on the HDFS and BGL datasets: SemiSMAC-LSTM, LSTM, SemiSMAC-GRU, GRU, SemiSMAC-CNN, CNN, SMAC-DeepLog, DeepLog, SMAC-LogRobust, LogRobust, SMAC-PLELog, and PLELog. Each model is assessed under two hyperparameter tuning strategies: traditional grid search and SMAC-based Bayesian optimization. Fig. 9 shows the corresponding time cost for hyperparameter search under both strategies on both datasets.

Figs. 9(a) and 9(b) illustrate the time cost of hyperparameter optimization using two strategies — SMAC and grid search — across all models on the HDFS and BGL datasets. Based on our statistical analysis, all models achieved comparable F1-scores under both tuning strategies. Specifically, LogRobust benefited notably from SMAC, achieving an average improvement of 2.1% in F1-score across the two datasets compared to its performance under empirical hyperparameter settings reported in the original paper.

Moreover, SMAC consistently demonstrates a significant efficiency advantage regardless of the dataset. On average, it reduces the total tuning time by 30.6% on HDFS and 25.4% on BGL compared to grid search. These results highlight the computational efficiency and practical value of SMAC, making it a more scalable and effective solution for hyperparameter optimization in real-world log anomaly detection scenarios.

Notably, unlike typical machine learning tasks where models are trained once and deployed in relatively stable environments, log anomaly detection is inherently a continuous and evolving process. Real-world software systems undergo frequent updates, patches, and the introduction of new modules and functionalities, all of which contribute to changes in log formats and structures. These changes are particularly common in newly deployed or resource-constrained environments, where manual reconfiguration of log analysis pipelines is often impractical. As a result, hyperparameters that were once optimal may degrade in effectiveness over time, necessitating periodic retuning to maintain detection accuracy. This dynamic landscape highlights the need for an automated, efficient, and generalizable tuning mechanism — such as SMAC — to ensure sustained anomaly detection performance with minimal human intervention across diverse and evolving log ecosystems.

Finding: SMAC demonstrates strong generalization capability across diverse deep learning architectures and achieves comparable performance to grid search while substantially reducing tuning time, offering clear advantages in efficiency.

6. Threats to validity

This section clarifies the threats to external, internal, construction, and conclusion validity, respectively.

6.1. External validity

Since log formats can vary widely across systems, organizations, and regions — sometimes even using different natural languages — the results obtained on the four datasets used in our experiments may not fully represent performance in all real-world environments. To address this, we selected datasets with diverse formats and domains. Nevertheless, future work should incorporate logs from a broader range of systems and industry applications to further enhance generalizability.

Furthermore, privacy and security concerns may arise when using ChatGPT in production environments, especially when dealing with sensitive system logs. In our study, we exclusively use the official OpenAI API. According to OpenAI's data usage policy,[3] data submitted via the API is not used for training purposes and is not retained after processing, unless users explicitly opt in. Furthermore, all datasets used

[3] https://openai.com/enterprise-privacy

in our experiments (e.g., HDFS, BGL) are publicly available and do not contain any personal or proprietary information. For real-world applications, additional safeguards — such as log anonymization, secure API usage, or deployment of locally hosted open-source LLMs — should be considered. Our framework is model-agnostic and can be easily adapted to privacy-preserving environments as needed.

6.2. Internal validity

One such factor is the noise introduced by ChatGPT during log parsing. As observed in Section 5.3, ChatGPT occasionally fails to correctly replace certain dynamic variables, leading to noisy templates. This noise may propagate through the pipeline and negatively impact pseudo-labeling and downstream anomaly detection. While we designed structured prompt templates to mitigate this issue, further refinement and prompt engineering are necessary to minimize noise in future iterations.

Another potential internal threat lies in the manual selection of labeled samples used to guide ChatGPT's in-context learning. Although we collaborate with experienced engineers and apply clustering techniques to ensure diversity and representativeness, the initial labeled samples may still introduce bias. Such bias could affect the grouping accuracy and the quality of generated pseudo-labels. In future work, we plan to investigate automated or active learning-based sample selection strategies to reduce potential human-induced bias.

6.3. Construct validity

In our study, the use of ChatGPT for log parsing and grouping relies heavily on predefined extraction and replacement rules. These rules are designed based on log format characteristics but may not capture every nuance in the data. Inaccurate variable replacement can lead to misgrouping, affecting the quality of pseudo-label generation. Although we customized rules for each dataset, a more robust and adaptive method for dynamic variable identification may further improve construct validity.

6.4. Conclusion validity

While our experiments showed strong and consistent performance across four datasets, the conclusions may be affected by factors such as model initialization, random seed settings, or parameter sensitivity. To reduce such risks, we conducted each experiment five times and reported the average results. Nevertheless, additional statistical testing and broader replication studies would further strengthen the reliability of our conclusions.

Another potential threat to conclusion validity lies in the inconsistency of hyperparameter optimization strategies across compared models. While our approach leverages SMAC for dataset-specific tuning, most baseline methods rely on manual configurations or limited grid search based on their original publications. Although we made efforts to adopt the best-known settings and conducted additional tuning where feasible, performance differences may partially reflect disparities in tuning quality. In future work, we plan to incorporate a fully standardized tuning protocol to further enhance the fairness and interpretability of performance comparisons.

7. Conclusion

In this study, we proposed SemiSMAC, a semi-supervised framework that integrates ChatGPT-assisted log parsing and grouping with SMAC-based hyperparameter optimization. The adaptive search capability of SMAC makes it an effective tool for enhancing deep learning models. Experimental results on multiple public datasets show that SemiSMAC effectively optimizes hyperparameters and improves anomaly detection performance, including in low-resource settings. By mitigating errors in log parsing and automating model configuration, the framework enhances both the practicality and robustness of log anomaly detection. Importantly, we emphasize that unlike traditional ML tasks, log anomaly detection must be continuously adapted to evolving systems and newly emerging log formats. Therefore, even modest reductions in tuning time — such as the 25%–30% achieved by SMAC — accumulate into significant practical gains in long-term system maintenance. Future work will explore extending SemiSMAC to support real-time log streams and evaluating its adaptability to diverse system environments and evolving log formats. Furthermore, while we adopt random forests as the surrogate model in SMAC, evaluating alternative surrogate models — such as SVMs or gradient boosting — remains a promising direction for future investigation.

CRediT authorship contribution statement

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

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

Data availability

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

References

[1] 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.
[2] Xingfang Wu, Heng Li, Foutse Khomh, On the effectiveness of log representation for log-based anomaly detection, Empir. Softw. Eng. 28 (6) (2023) 137.
[3] Jian-Guang Lou, Qiang Fu, Shenqi Yang, Ye Xu, Jiang Li, Mining invariants from console logs for system problem detection, in: 2010 USENIX Annual Technical Conference, USENIX ATC 10, 2010.
[4] Yicheng Sun, Jacky Keung, Hi Kuen Yu, Shuo Liu, Yihan Liao, Jingyu Zhang, Beyond log parsers: A scalable AI-driven framework for efficient log anomaly detection in software engineering, in: Proceedings-2025 IEEE 49th International Conference on Computers, Software, and Applications, COMPSAC 2025, 2025.
[5] 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.
[6] 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.
[7] Yicheng Sun, Jacky Wai Keung, Zhen Yang, Shuo Liu, Hi Kuen Yu, SemiRALD: A semi-supervised hybrid language model for robust anomalous log detection, Inf. Softw. Technol. (2025) 107743.
[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] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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, no. 7, 2019, pp. 4739–4745.

[20] 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.

[21] 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.

[22] 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: 2024 IEEE 48th International Conference on Computers, Software, and Applications, COMPSAC, IEEE, 2024.

[23] 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.

[24] Yinglung Liang, Yanyong Zhang, Hui Xiong, Ramendra Sahoo, Failure prediction in ibm bluegene/l event logs, in: Seventh IEEE International Conference on Data Mining, ICDM 2007, IEEE, 2007, pp. 583–588.

[25] 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.

[26] 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.

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

[28] 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.

[29] 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.

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

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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, ICSE, in: 2024 IEEE/ACM 46th International Conference on Software Engineering, vol. 4, IEEE Computer Society, 2024, 883–883.

[36] Konstantinos I. Roumeliotis, Nikolaos D. Tselikas, Chatgpt and open-ai models: A preliminary review, Futur. Internet 15 (6) (2023) 192.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, Marius Kloft, Deep one-class classification, in: International Conference on Machine Learning, PMLR, 2018, pp. 4393–4402.

[42] Xiaoyun Li, Pengfei Chen, Linxiao Jing, Zilong He, Guangba Yu, Swisslog: Robust and unified deep learning based log anomaly detection for diverse faults, in: 2020 IEEE 31st International Symposium on Software Reliability Engineering, ISSRE, IEEE, 2020, pp. 92–103.

[43] Leland McInnes, John Healy, Steve Astels, Hdbscan: Hierarchical density based clustering, J. Open Source Softw. 2 (11) (2017) 205.

[44] 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.

[45] Leland McInnes, John Healy, Accelerated hierarchical density based clustering, in: 2017 IEEE International Conference on Data Mining Workshops, ICDMW, IEEE, 2017, pp. 33–42.

[46] 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.

[47] 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: Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, 2024, pp. 1–13.

[48] Hongcheng Guo, Jian Yang, Jiaheng Liu, Jiaqi Bai, Boyang Wang, Zhoujun Li, Tieqiao Zheng, Bo Zhang, Junran Peng, Qi Tian, Logformer: A pre-train and tuning pipeline for log anomaly detection, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 38, no. 1, 2024, pp. 135–143.

[49] 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.

[50] Junjie Zhao, Diyuan Li, Jian Zhou, Danial J. Armaghani, Aohui Zhou, Performance evaluation of rock fragmentation prediction based on RF-BOA, AdaBoost-BOA, GBoost-BOA, and ERT-BOA hybrid models, Deep. Undergr. Sci. Eng. (2024).

[51] Donald R. Jones, Matthias Schonlau, William J. Welch, Efficient global optimization of expensive black-box functions, J. Global Optim. 13 (1998) 455–492.

[52] Martin Pelikan, Martin Pelikan, Bayesian optimization algorithm, in: Hierarchical Bayesian Optimization Algorithm: Toward a New Gener- Ation of Evolutionary Algorithms, Springer, 2005, pp. 31–48.

[53] Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, Frank Hutter, Fast Bayesian hyperparameter optimization on large datasets, 2017.

[54] James Bergstra, Rémi Bardenet, Yoshua Bengio, Balázs Kégl, Algorithms for hyper-parameter optimization, Adv. Neural Inf. Process. Syst. 24 (2011).

[55] Jasper Snoek, Hugo Larochelle, Ryan P. Adams, Practical Bayesian optimization of machine learning algorithms, Adv. Neural Inf. Process. Syst. 25 (2012).

[56] Martin Pelikan, Bayesian Optimization Algorithm: From Single Level to Hierarchy, University of Illinois at Urbana-Champaign, 2002.

[57] Miloš Kobliha, Josef Schwarz, Jiří Očenášek, Bayesian optimization algorithms for dynamic problems, in: Applications of Evolutionary Computing: EvoWorkshops 2006: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoINTERACTION, EvoMUSART, and EvoSTOC, Budapest, Hungary, April 10-12, 2006. Proceedings, Springer, 2006, pp. 800–804.
[58] Peter I. Frazier, A tutorial on Bayesian optimization, 2018, arXiv preprint arXiv:1807.02811.
[59] Xilu Wang, Yaochu Jin, Sebastian Schmitt, Markus Olhofer, Recent advances in Bayesian optimization, ACM Comput. Surv. 55 (13s) (2023) 1–36.
[60] Roman Garnett, Bayesian optimization, Cambridge University Press, 2023.
[61] Qiang Shang, Derong Tan, Song Gao, Linlin Feng, et al., A hybrid method for traffic incident duration prediction using BOA-optimized random forest combined with neighborhood components analysis, J. Adv. Transp. 2019 (2019).
[62] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, Sequential model-based optimization for general algorithm configuration, in: Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5, Springer, 2011, pp. 507–523.
[63] Marco Laumanns, Jiri Ocenasek, Bayesian optimization algorithms for multi-objective optimization, in: International Conference on Parallel Problem Solving from Nature, Springer, 2002, pp. 298–307.
[64] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, Sequential model-based optimization for general algorithm configuration (extended version), in: Technical Report TR-2010–10, Tech. Rep., University of British Columbia, Computer Science, 2010.
[65] 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.
[66] 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.
[67] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, Tomas Mikolov, Fasttext. zip: Compressing text classification models, 2016, arXiv preprint arXiv:1612.03651.
[68] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, Nando De Freitas, Taking the human out of the loop: A review of Bayesian optimization, Proc. IEEE 104 (1) (2015) 148–175.
[69] Emmanuel Vazquez, Julien Bect, Convergence properties of the expected improvement algorithm with fixed mean and covariance functions, J. Statist. Plann. Inference 140 (11) (2010) 3088–3095.
[70] P.M. Lerman, Fitting segmented regression models by grid search, J. R. Stat. Soc. Ser. C. Appl. Stat. 29 (1) (1980) 77–84.
[71] 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.
[72] 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.
[73] Long Short-Term Memory, Long short-term memory, Neural Comput. 9 (8) (2010) 1735–1780.
[74] Risto Vaarandi, A data clustering algorithm for mining patterns from event logs, in: Proceedings of the 3rd IEEE Workshop on IP Operations & Management, IPOM 2003 (IEEE Cat. No. 03EX764), IEEE, 2003, pp. 119–126.
[75] Yudong Liu, Xu Zhang, Shilin He, Hongyu Zhang, Liqun Li, Yu Kang, Yong Xu, Minghua Ma, Qingwei Lin, Yingnong Dang, et al., Uniparser: A unified log parser for heterogeneous log data, in: Proceedings of the ACM Web Conference 2022, 2022, pp. 1893–1901.
[76] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, Yoshua Bengio, Learning phrase representations using RNN encoder-decoder for statistical machine translation, 2014, arXiv preprint arXiv:1406.1078.
[77] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, Jun Zhou, A survey of convolutional neural networks: analysis, applications, and prospects, IEEE Trans. Neural Netw. Learn. Syst. 33 (12) (2021) 6999–7019.
[78] V.H. Le, H. Zhang, Log parsing with prompt-based few-shot learning, 2023, arXiv. arXiv preprint arXiv:2302.07435.
[79] Zanis Ali Khan, Donghwan Shin, Domenico Bianculli, Lionel Briand, Impact of log parsing on log-based anomaly detection, 2023, arXiv preprint arXiv:2305.15897.
[80] Siyang Lu, BingBing Rao, Xiang Wei, Byungchul Tak, Long Wang, Liqiang Wang, Log-based abnormal task detection and root cause analysis for spark, in: 2017 IEEE International Conference on Web Services, ICWS, IEEE, 2017, pp. 389–396.
[81] Donghwan Shin, Zanis Ali Khan, Domenico Bianculli, Lionel Briand, A theoretical framework for understanding the relationship between log parsing and anomaly detection, in: International Conference on Runtime Verification, Springer, 2021, pp. 277–287.