sciendo

JAISCR, 2026, Vol. 16, No. 2, pp. 125 – 144
10.2478/jaiscr-2026-0007

FRLOG: LOG ANOMALY DETECTION BASED ON THREE-STAGE TRAINING WITH REFT FINE-TUNING FOR LARGE LANGUAGE MODEL

Keyuan Qiu[1], Zhejie Xu[1], Tao Luo[1], Meifang Yan[2], Ruru Liu[3], Pengjin Liu[1], Feng Chen[1,*]

[1]*College of Information Science and Technology, Shihezi University, No. 221, Beisi Road, Shihezi City, 832003, Xinjiang Uygur Autonomous Region, China*

[2]*College of Computer Science and Technology, Xinjiang Normal University, 102 Xinyi Road, Urumqi City, 830054, Xinjiang Uygur Autonomous Region, China*

[3]*College of Artificial Intelligence, Jilin University, No. 2699, Qianjin Street, Changchun City, 130012, Jilin Province, China*

[*]*E-mail: cf_inf@shzu.edu.cn*

Submitted: 11th June 2025; Accepted: 27th December 2025

Abstract

The goal of log anomaly detection is to accurately detect system anomalies from logs. Traditional methods often suffer from insufficient generalization and delayed anomaly detection when dealing with semantically diverse and loosely structured log data. As the complexity of the system increases, the size of the logs is getting larger and larger, and it has become impractical to analyze them manually. To this end, this paper proposes FRLog, a log anomaly detection framework based on large language model, which realizes contextualized semantic embeddings of log sequences by fusing BERT and LLaMA models, thereby enabling more accurate log anomaly detection. Meanwhile, the parameter fine-tuning strategy ReFT is introduced, and the semantic bootstrapping, representation alignment and global tuning process are optimized by a three-phase collaborative training mechanism. Experimental results on three typical log datasets, BGL, HDFS and Thunderbird, show that FRLog outperforms the existing mainstream methods in terms of F1, Precision and Recall, especially in complex scenarios, demonstrating stronger anomaly discrimination and sample efficiency, which verifies its superiority and robustness in the log anomaly detection task.

Keywords: anomaly detection, system security, model optimisation, log analysis

1 Introduction

System logs refer to records generated during the operation of modern network devices, operating systems, and service programs that document system states and event information. As the scale and complexity of large-scale systems continue to grow, both external failures and internal errors may lead to system crashes. Therefore, it is essential to detect anomalies in the system promptly and accurately to ensure reliability and stability, thereby minimizing unnecessary losses [1].

Unlike other types of data [2, 3, 4, 5], log data records include time information, operational details, execution logic, and event information, resulting in semi-structured characteristics. Among many deep learning-based methods, researchers often encode raw log messages directly and perform anomaly detection via text classification. For instance, LogBERT [6] and NeuralLog [7] extract the semantics of raw log messages and represent them as semantic vectors, using classification models based on BERT [8] or Transformer [9] for anomaly detection.

Although existing methods have achieved significant performance in log anomaly detection, their practical application still faces several challenges. First, most existing methods rely heavily on large amounts of high-quality training data. When data is insufficient or lacks representativeness, the model may fail to accurately learn the patterns of normal log sequences. Second, as systems evolve and update, existing models may not adapt to new log formats and behavioral patterns, requiring periodic retraining. Moreover, log parsers typically rely on rules or prior structures to separate templates and variables. Some parsers normalize content during template extraction, which may remove or simplify fields, thereby breaking the semantic structure of original logs and making it difficult for models to learn effective contextual information.

To address these challenges, this paper proposes a high-efficiency log anomaly detection framework named FRLog, which integrates LLaMA with ReFT (Representation Fine-Tuning). This approach utilizes BERT to extract semantic representations of logs and applies ReFT to inject low-rank interventions into the intermediate hidden states of LLaMA, guiding task-specific semantic adaptation. A three-stage training strategy is adopted to mitigate target drift and semantic bias during model training. Without relying on log parsers, FRLog can effectively capture semantic anomalies in complex logs, significantly reduce fine-tuning overhead for large language models, and improve model adaptability and sample efficiency.

2 Related works

Current research on contextual anomaly detection in system logs can be broadly categorized into four types: traditional machine learning methods, recurrent neural network (RNN)-based methods, Transformer-based methods, and Large Language Model (LLM)-based methods.

Traditional machine learning methods: Early work by He et al. [10] compared various supervised and unsupervised anomaly detection methods based on traditional machine learning and statistical techniques. Unsupervised approaches include principal component analysis [11] and log clustering [12], while supervised ones include support vector machines [13], logistic regression [14], and decision trees [15]. These methods typically rely on extracting feature vectors for each log entry, which are then used for anomaly detection. Common algorithms include KNN [12, 16], Bayesian networks [17, 18], Hidden Markov Models [19, 20], and Isolation Forests [21, 22, 23]. However, they often require manual feature engineering, which may omit latent information critical to detecting anomalies.

RNN-based methods: With the rise of RNNs, researchers have applied them and their variants to log anomaly detection. The LogRobust model leverages Word2Vec for log template embedding, uses TF-IDF for word weighting, and applies a Bi-LSTM to determine anomalies. However, LogRobust relies on log parsers and faces difficulties in sequence construction. DeepLog [24] detects anomalies based on log key and parameter value sequences using LSTM. Log2Vec [25] represents logs as a heterogeneous graph, learns embeddings, and applies DeepLog-style detection. Yet, it performs poorly on domain-specific terms and named entities [26]. Wang et al. [27] proposed a CNN-LSTM model for capturing dimensional information in text, while Luo et al. [28] combined latent Dirichlet allocation (LDA) [29] with a GRU-CNN for classification.

Transformer-based methods: Most RNN models predict the next log based on prior messages to model normal sequences. However, changes in log dependencies often reduce accuracy, and such objectives fail to capture global patterns. Transformer-based models have thus been explored. Xiao et al. [30] proposed Loader, which integrates variable fusion into Transformer to better combine template and variable information. BERT has demonstrated superior feature extraction in NLP and achieved state-of-the-art results

across 11 tasks [28], prompting its adoption in log anomaly detection. Guo et al. [31] proposed LogBERT, while Yu et al. [32] introduced MFLAD, combining BERT with Bi-LSTM. Qiu et al. [33] used a BERT-Electra pseudo-anomaly generation framework with a discriminator for detection.

LLM-based methods: With the advancement of LLMs such as GPT and LLaMA, their use in log anomaly detection is gaining interest. Qi et al. [34] introduced LogGPT, which predicts next log entries using reinforcement learning. However, as an autoregressive model, it lacks clear classification boundaries, suffers from token confusion and overflow in long sequences, and offers limited semantic interpretability. Similar models include those by He et al. [35], Han et al. [36], and Hadadi et al. [37]. Guan et al. [38] proposed a three-stage training approach with QLoRA for efficient tuning, but [39] pointed out that it struggles with complex structures and semantic variance, and often overfits. Moreover, LoRA is rank-sensitive and less effective in low-resource scenarios. Recently, Nasser et al. [40] demonstrated the effectiveness of stacking multiple LLMs in phishing URL detection, highlighting the potential of ensemble strategies in enhancing robustness and accuracy. Likewise, Romaszewski et al. [41] explored number-oriented LLMs derived from Random Forest models, showing that task-specific adaptation of LLMs can significantly improve interpretability and performance on structured data. These studies suggest that the design of prompt paradigms, model stacking, and hybridization with classical models are promising directions for further exploration in log anomaly detection.

In response to these challenges, this paper proposes a novel framework, FRLog, for log anomaly detection. Based on a three-stage training approach, FRLog introduces ReFT (Representation Fine-Tuning) as a parameter-efficient strategy. By injecting low-rank interventions into hidden representations, ReFT enables precise modeling of anomaly semantics. Compared to conventional fine-tuning, ReFT achieves better generalization and sample efficiency, particularly under complex log structures and data scarcity.

3 Proposed methodology

3.1 Data pre-processing

Since the variable parameters contained in the log messages carry dynamic runtime information, which is often not related to anomalies but makes model training more difficult, we need to identify and replace these parameters. Given the structured nature of the logs and the distinctive format of the variable parameters, we use regular expression(RE) to replace parameters such as accounts, paths, IP addresses, etc. with uniform <*> tags for the log format. This approach requires no training and improves performance significantly over traditional log parsers.

3.2 Log grouping

Simple outlier detection usually does not require grouping of logs because single anomalous events are considered context-independent. However, production environments often need to reveal anomalous patterns in multiple log events, such as changes in event sequences or correlations in time logs. Therefore, the model needs to logically group log events and analyze each event group individually or in correlation with each other.

Conventional log grouping methods include fixed windows, sliding windows, and session windows. As shown in figures 1, 2, and 3, fixed windows are based on splitting a continuous sequence of logs into fixed-size windows that do not overlap each other according to the log timestamps and window sizes, and are commonly used for timed monitoring tasks.

Sliding windows are also grouped based on log timestamp data, which can be determined based on the length of time to determine the size of the window or the number of entries in the log to determine the size of the window and step division based on the size of the window and the window before and after the window will overlap.

The session window is grouped based on the session identifier of the log. The system can have multiple tasks executing concurrently, causing the logs to be executed in a staggered order.

In this paper, the model uses sliding grouping and session grouping for different log datasets to avoid context loss due to time intervals or logical

Figure 1. Fixed windows grouping

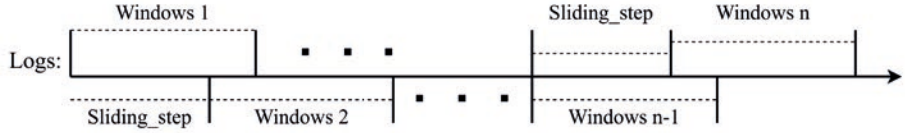


Figure 2. Sliding windows grouping

breaks in order to better reflect the current state of that node.

3.3 FRLog model

As shown in figure 4, the FRLog model consists of three components: the BERT, the Projector, and the Llama.The BERT is used to extract the semantic representations of the log messages, the Llama employs a Meta-Llama-3-8B pre-trained model for the classification of the log sequences, and the Projector is used to align the two representation spaces. Each component undergoes a specific optimization process in training.

Each preprocessed log message is tokenized using the BERT tokenizer, and the [CLS] token is used to represent the entire semantics. A linear layer followed by a `tanh` activation function is applied to produce the final semantic embedding. For a sequence of N log messages, the output is a semantic vector sequence $C = (c_1, c_2, ..., c_N) \in \mathbb{R}^{N \times d_{\text{BERT}}}$.

The Projector is a linear layer that maps the BERT output C to the Llama-compatible embedding space $E = (e_1, e_2, ..., e_N) \in \mathbb{R}^{N \times d_{\text{Llama}}}$, achieving representation alignment.

We employ Prompt Tuning to fine-tune Llama. The input is composed of three parts:

- Part 1: Prompt text (e.g., `"The following is a sequence of syslog messages:"`);
- Part 2: Token embeddings output from the projector;
- Part 3: Query text (e.g., `"Is this part of the log sequence abnormal? n"`).

These three parts are tokenized and embedded to form the embedding sequences E_1, E, E_3, which are then concatenated as $[E_1 \parallel E_2 \parallel E_3]$ and input into Llama. The overall process is shown in Algorithm 1.

Algorithm 1: LLaMA Input Construction Process

Input: Log sequence `Logs`, Prompt text `Prompt`, Query text `Query`
Output: LLaMA input embedding vector E

`prompt_text` ← "Below is a sequence of system log messages:"
Encode `prompt_text` using LLaMA tokenizer to obtain vector E_1
Feed `Logs` into the BERT model to extract the [CLS] representation for each log message
Map the extracted representations using Projector to obtain semantic vector E_2
`query_text` ← "Is this sequence normal or anomalous?"
Encode `query_text` using LLaMA tokenizer to obtain vector E_3
Concatenate E_1, E_2, E_3 to form the final input vector $E = [E_1 \| E_2 \| E_3]$
Feed the concatenated vector E into the LLaMA model for anomaly classification

On the other hand,in order to further reduce the computational and memory overhead of large language model fine-tuning while enhancing model stability and generalization in log anomaly detection tasks, we introduce the ReFT fine-tuning strategy as a lightweight alternative to LoRA. This approach maintains the primary architecture of the

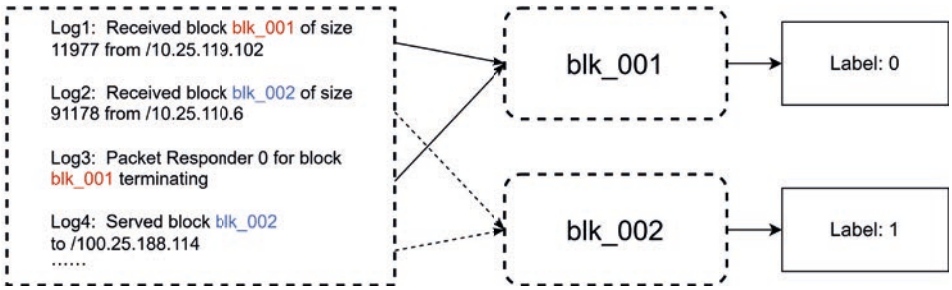


Figure 3. Session windows grouping

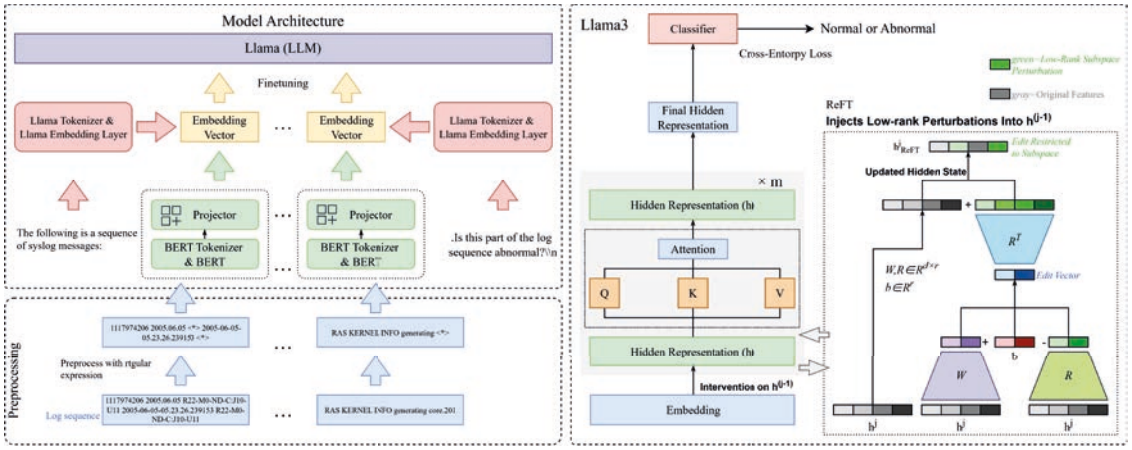


Figure 4. FRLog model structure diagram

model while enabling efficient adaptation to downstream tasks.

The ReFT module is inserted before each self-attention mechanism in every transformer layer, acting on the hidden representation output from the previous layer, denoted as $h^{(j-1)}$. The module applies low-rank perturbations to the original representation to guide the attention module toward task-relevant semantic regions. Ultimately, these features are passed through stacked attention and feed-forward layers to produce the final classification output.

Specifically, the ReFT operation is defined as:

$$h^{(j)}_{\text{ReFT}} = h^{(j-1)} + R^{\top}\left(Wh^{(j-1)} + b - Rh^{(j-1)}\right), \quad (1)$$

where $R \in \mathbb{R}^{r\times d}$ is the low-rank projection matrix, $W \in \mathbb{R}^{d\times r}$, and $b \in \mathbb{R}^{r}$ are trainable parameters. Here, d denotes the hidden dimension and $r \ll d$ represents the dimension of the low-rank subspace (set to $r = 8$ in our experiments). For initialization, R is orthogonally initialized to ensure subspace stability, W is initialized using the Xavier scheme, and the bias term b is set to zero to avoid introducing large perturbations at the beginning of training.

Regarding the intervention granularity, ReFT modules are inserted independently at the layer level before each self-attention mechanism of the Transformer, so that every layer has its own ReFT parameters rather than being applied separately to each attention head. This design ensures both flexibility in representation and efficiency in parameter usage.

This operation restricts perturbations to a task-related low-rank subspace, thereby avoiding disruption of the general semantics already learned by the pretrained model. In practice, ReFT is applied to the [CLS] token representation in BERT and to the final token representation in LLaMA. Since ReFT does not modify the original weights and does not rely on quantization, it can be directly applied to full-precision models for fine-tuning.

Compared with other parameter-efficient fine-tuning (PEFT) methods, ReFT differs in both its intervention position and its expressive formulation. LoRA introduces low-rank updates directly

into the weight matrices, while QLoRA further reduces memory usage by quantizing pre-trained weights into 4-bit precision with low-rank adapters to compensate for quantization errors. In contrast, ReFT operates in the hidden representation space: it injects low-rank perturbations before each self-attention block without modifying the original weights or relying on quantization. This design allows ReFT to preserve full-precision semantics while retaining sufficient expressive capacity through the subspace dimension r.The detailed comparison is shown in Table 1.

In addition, to ensure model convergence and generalization performance, we adopt a three-stage training strategy:

Stage 1: LLaMA layer fine-tuning

Only the last few layers of LLaMA are unfrozen, while the remaining parts remain frozen. The loss function is binary cross-entropy, and the optimizer is AdamW with a learning rate of $1e{-}5$. A 5% warmup and cosine annealing scheduler are applied. The Projector is trained together with LLaMA at this stage to ensure the stability of dimensional mapping.

Stage 2: Training BERT + Projector + ReFT plugin embeddings

BERT and the Projector are unfrozen, and the parameters of the ReFT module are introduced for training. The loss function remains cross-entropy, but different modules are assigned differentiated learning rates: BERT adopts a smaller learning rate $5e{-}6$, while the Projector and ReFT adopt a larger learning rate $5e{-}5$. The optimizer continues to be AdamW, and weights are initialized from the best checkpoint of Stage 1.

Stage 3: Jointly fine-tune LLaMA and the ReFT module within the embedding component.

All parameters are unfrozen for end-to-end joint optimization. The loss function remains cross-entropy, but a weighted cross-entropy is introduced in cases of class imbalance. The learning rate is further reduced to $1e{-}6$ to ensure stable convergence. Training continues from the best checkpoint of Stage 2 to guarantee continuity across stages.

In addition, for the log anomaly detection task, normal logs constitute the vast majority, while abnormal logs are inherently rare. This severe class imbalance may lead the model to develop a training bias—tending to classify all inputs as normal—thus degrading the performance of anomaly identification.

According to the target ratio control formula, we aim to satisfy:

$$\frac{N_d}{N_d + N_n} = \beta \tag{2}$$

That is,

$$N_d = \beta \cdot (N_d + N_n) \tag{3}$$

where α denotes the proportion of anomaly samples in the original dataset; β is the target ratio of anomaly samples after resampling. Let S_{num} be the total number of samples in the original training set, and let $N_n = (1-\alpha) \cdot S_{\text{num}}$ be the number of normal samples. Then N_d is the desired number of anomaly samples after resampling.

Expanding and rearranging the equation on the right-hand side, we obtain:

$$\begin{aligned} N_d &= \beta N_d + \beta N_n \\ (1-\beta) N_d &= \beta N_n \\ N_d &= \frac{\beta}{1-\beta} \cdot N_n \end{aligned} \tag{4}$$

Since $N_n = (1-\alpha) \cdot S_{\text{num}}$, substituting in yields:

$$N_d = \frac{\beta(1-\alpha)}{1-\beta} \cdot S_{\text{num}} \tag{5}$$

Therefore, we derive that the number of anomaly samples should be increased through duplication to the target amount. This formula ensures that after resampling, the proportion of anomaly samples in the training set aligns with the target ratio β, thereby effectively mitigating the model bias problem caused by class imbalance.

On this basis, the training objective of the model adopts the standard Binary Cross-Entropy (BCE) loss function.

Let the ground-truth label be $y \in \{0,1\}$ and the predicted probability be $\hat{y} \in (0,1)$. The loss function is defined as:

Table 1. Comparison of different PEFT methods.

Method	Insertion Position	Quantization	Expressive Source	Limitation
LoRA	Weight matrices	No	Low-rank updates	Sensitive to rank r
QLoRA	Quantized weights	Yes (4-bit)	LoRA + quant. comp.	Quantization noise
ReFT	Hidden representations	No	Low-rank subspace (r)	Requires r tuning

$$\mathcal{L}_{\text{BCE}} = -(y \log \hat{y} + (1-y)\log(1-\hat{y})) \quad (6)$$

When the model determines that a log sequence is abnormal, it outputs the class label "Abnormal." Otherwise, it outputs "Normal."

4 Experiments

4.1 Environmental Configuration

The model parameters in this paper are set as shown in Table 2 below.

4.2 Experimental datasets

This experiment utilizes three widely used public datasets in the field of log anomaly detection: BGL, HDFS, and Thunderbird. The training and testing sets are split in an 8:2 ratio.

BGL [42, 43] is an open log dataset collected from the BlueGene/L supercomputer system at Lawrence Livermore National Labs (LLNL) in California, which consists of 131,072 processors and 32,768 GB of memory. The logs include alert and non-alert messages labeled by alert categories, reflecting characteristics of large-scale parallel computing environments. In addition, the dataset contains a massive number of log entries with a relatively broad anomaly distribution, which is beneficial for evaluating model performance in high-dimensional log environments. As a result, BGL has been widely used in studies on log parsing, anomaly detection, and fault prediction.

HDFS [44, 45] is a log dataset aggregated from the HDFS system in a laboratory at the Chinese University of Hong Kong. The system comprises one name node and 32 data nodes, and the dataset exceeds 16 GB in size. As a distributed file system log, it reflects the characteristics of logs generated in distributed storage environments and mainly contains contextual anomaly types.

Thunderbird [42, 43] is an open log dataset collected from the Thunderbird supercomputer system at Sandia National Labs (SNL) in Albuquerque, which has 9,024 processors and 27,072 GB of memory. The logs contain alert and non-alert messages labeled by alert categories, and the entries cover a wide range of information including hardware, networking, and user operations. The dataset includes a large number of unstructured or semi-structured logs, making it suitable for evaluating the model's generalization ability in unstructured log environments.

The composition of anomalies in the three datasets is summarized in Table 3, where each dataset includes 4 to 5 major anomaly categories and more than 10 subtypes of anomalies.

4.3 Evaluation Indicators

To comprehensively evaluate the performance improvement of the FRLog model in the detection task, this study introduces multiple evaluation metrics. Considering that the detection method used in this study is a classification task, four commonly used metrics are adopted: Precision, Recall, F1 Score, and AUC (Area Under the Curve).

$$\text{precision} = \frac{TP}{TP+FP} \quad (7)$$

$$\text{recall} = \frac{TP}{TP+FN} \quad (8)$$

$$\text{AUC} = \int_0^1 \frac{TP}{TP+FN}\, d\left(\frac{FP}{FP+TN}\right) \quad (9)$$

$$F1 = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \quad (10)$$

TP (True Positive) refers to the number of samples that are actually positive and are also predicted as positive by the model. TN (True Negative) refers to the number of samples that are actually negative

Table 2. Experimental Environment Configuration

Experimental environment	Environmental configuration
System	Linux
GPU	H20-NVLink
CPU	20 vCPU Intel(R) Xeon(R) Platinum 8457C
PyTorch	2.1.0
Python	3.10
CUDA	12.1

Table 3. Anomaly composition of each dataset

Dataset	Type Quantity	Failure Type
BGL	5	Human error, environmental error, network failure, software failure, hardware failure
HDFS	5	Process errors, network failures, data errors, system failures, system vulnerabilities
Thunderbird	4	Hard disk failure, memory failure, CPU failure, power failure

and are also predicted as negative by the model. FP (False Positive) refers to the number of samples that are actually negative but are incorrectly predicted as positive by the model. FN (False Negative) refers to the number of samples that are actually positive but are incorrectly predicted as negative by the model.

4.4 Comparison of experimental results analysis

4.4.1 Analysis of experimental results comparing different models

To evaluate the performance of the FRLog model on log anomaly detection tasks, several competitive baseline models are selected for comparison.

LogBERT [6] is a log anomaly detection model based on a pre-trained BERT language model.

FastLogAD [45] leverages a pre-trained ELECTRA model to generate synthetic anomalies by replacing key tokens in log templates.

Log2Graphs [46] represents log events as graph structures and applies graph neural networks for representation learning.

LogGPT [36] utilizes the autoregressive language modeling capability of GPT-2 to perform log sequence modeling and detect anomalies based on token-level likelihoods.

LogLLM [38] integrates both BERT and LLaMA language models to align features and perform stage-wise fine-tuning for anomaly detection.

Table 4 presents the performance of various models on the datasets, with all reported results obtained from their respective original publications. From the experimental results, it can be observed that the FRLog model achieves significant performance improvements on the BGL and HDFS datasets, and maintains strong performance on the Thunderbird dataset. Compared with LogLLM and LogGPT, ReFT does not simply perform shallow fine-tuning on LLMs. Instead, it guides the internal representation space by compressing redundant features and aligning key semantics, enabling the model to more effectively distinguish between normal and anomalous log entries.

Compared with traditional methods, FRLog exhibits clear advantages. LogBERT, which uses a BERT encoder, possesses a certain ability for contextual modeling. However, due to its structure being limited to local windows, it struggles to capture long-range dependencies, resulting in limited performance on datasets like BGL and HDFS that contain long sequences. FastLogAD enhances recall by generating pseudo-anomalies, which leads to high recall scores. However, this also causes a general drop in precision, resulting in more false alarms and larger fluctuations in the overall F1 score. Log2Graphs constructs event graphs for structural modeling, which is well-suited for datasets with strong topological dependencies, such as Thunderbird. Nevertheless, its graph construction relies on handcrafted rules and priors, lacking semantic generalization capability.

Table 4. Performance comparison of different models on BGL, HDFS, and Thunderbird datasets.

Model	BGL			HDFS			Thunderbird		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
LogBERT	0.8940	0.9232	0.9083	0.8702	0.7810	0.8232	0.9675	0.9652	0.9664
FastLogAD	0.8828	**0.9866**	0.9314	0.8480	0.9999	0.9177	0.9545	0.9991	0.9763
Log2Graphs	0.9610	0.9310	0.9464	0.8740	0.9120	0.8933	0.9900	**1.0000**	0.9950
LogGPT	0.9410	0.9788	0.9598	0.9140	0.9276	0.9211	0.9734	**1.0000**	0.9865
LogLLM	0.8610	0.9790	0.9160	0.9940	**1.0000**	0.9970	0.9960	0.9960	**0.9960**
Ours	**0.9429**	0.9808	**0.9615**	**0.9962**	**1.0000**	**0.9981**	**0.9987**	0.9865	0.9925

Note: Boldface indicates the best performance in each column.

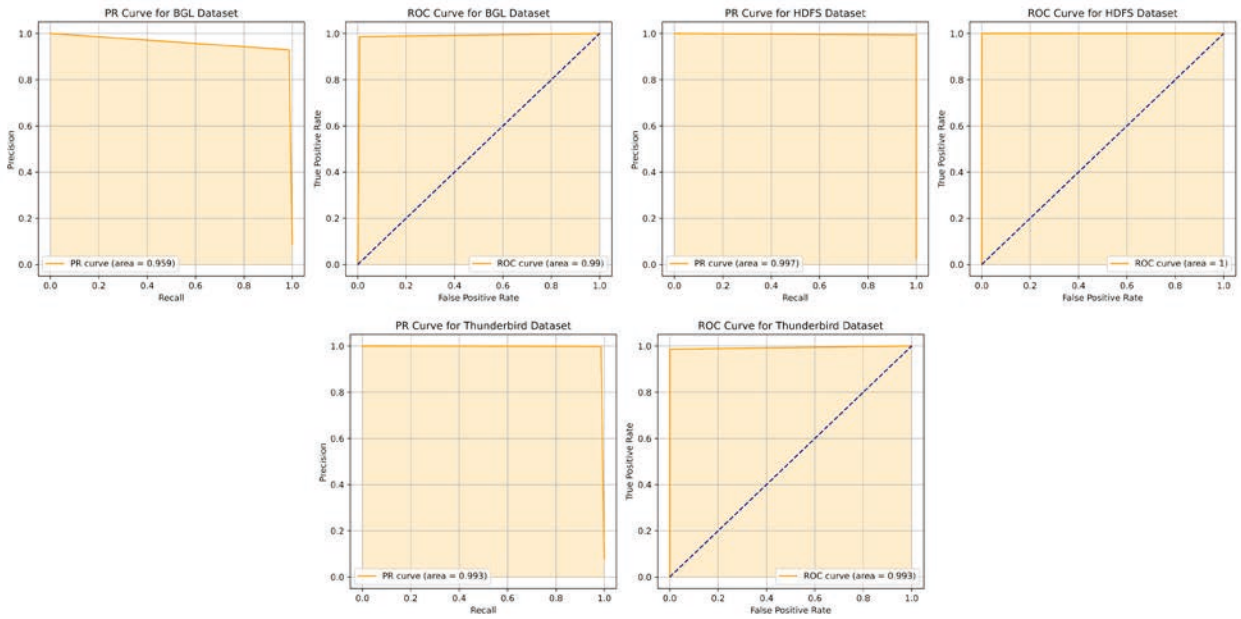


Figure 5. PR curves and ROC curves of FRLog model on three datasets

As shown in the figure 5, the model demonstrates excellent performance across all three datasets, achieving near-perfect results on HDFS, which highlights its strong discriminative capability. Moreover, it maintains high precision on the structurally complex Thunderbird dataset, verifying its robustness and generalization ability. These results indicate that the model is well-suited for diverse real-world log scenarios.

The figure 6 illustrates the confusion matrix results of the FRLog model on the BGL, HDFS, and Thunderbird datasets. Overall, the results demonstrate that FRLog achieves both high accuracy and robustness in anomaly detection tasks, indicating strong discriminative capability for distinguishing between normal and anomalous log entries.

To comprehensively evaluate the effectiveness and robustness of the proposed method, we conducted experiments on three public log datasets—BGL, HDFS, and Thunderbird. Considering that the results of a single run may be affected by random initialization and data partitioning, we performed 10 independent runs with different random seeds and reported the mean and standard deviation of the results. Furthermore, to verify the statistical significance of the performance differences, we conducted paired t-tests between FRLog and the best-performing baseline model on each dataset, and reported the average difference Δ, the 95% confidence interval (95% CI), the t-value, and the p-value. In terms of the choice of baselines, LogGPT was selected as the comparator on the BGL dataset, LogLLM was chosen on the HDFS dataset, and LogLLM was also adopted as the comparator on the Thunderbird dataset.

Table 5 presents the paired t-test results on the three benchmark datasets. Overall, the FRLog model exhibits consistent advantages on BGL and HDFS, while performing slightly worse than LogLLM on Thunderbird. Specifically, on BGL, FRLog achieves an average F1 of 0.9615, showing a significant improvement over LogGPT. On HDFS, the performance gap between FRLog and LogLLM is only at the scale of one-thousandth, yet statistical testing still confirms the significance, indicating that the model can maintain a stable advantage on large-scale log data. In contrast, on Thunderbird, FRLog achieves a slightly lower average F1 compared to LogLLM. Although the difference is statistically significant, both models remain in the high-performance range around 0.99, suggesting that their practical effectiveness is nearly equivalent. Considering that Thunderbird logs are highly templated and semantically repetitive, we conjecture that such characteristics weaken the role of cross-event dependency modeling, thereby limiting the advantages of FRLog.

In summary, FRLog demonstrates clear value in semantically complex and long-sequence scenarios, while its performance on structurally simple datasets remains competitive with state-of-the-art methods.

4.4.2 Analysis of the impact of different log preprocessing strategies

To evaluate the impact of different preprocessing strategies on log anomaly detection performance, we conducted a comparative study involving four commonly used preprocessing methods: directly using raw logs (Raw), extracting template content via log parsers (Template), and the RE-based replacement strategy proposed in this work. All methods were trained and evaluated under the same model architecture to ensure fair comparison, with only the preprocessing strategy being varied. Experiments were conducted on three benchmark datasets—HDFS, BGL, and Thunderbird—using Precision, Recall, and F1-score as evaluation metrics. The experimental results are summarized in Table 6.

The experimental results indicate that all three preprocessing methods achieve high F1 scores on the HDFS dataset, which exhibits relatively regular log structure. Among them, the Template-based approach, relying on log parsers, shows a slight advantage. However, on more structurally complex datasets such as BGL and Thunderbird, the RE method achieves the best detection performance. Although the Raw preprocessing method retains the complete original log content, it also introduces a large number of dynamic variables. These highly variable elements, while not directly related to anomalies, significantly interfere with the model's ability to learn meaningful abnormal patterns.

In contrast, the RE method normalizes the log content via RE replacements, effectively removing

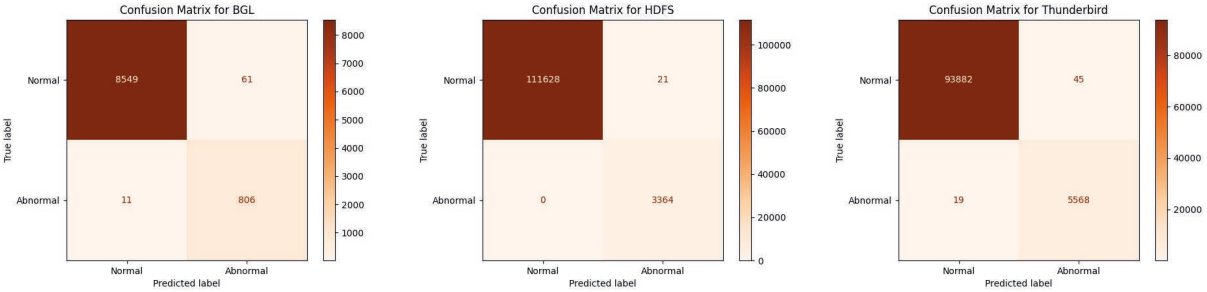


Figure 6. Confusion matrix for FRLog model on three datasets

Table 5. Paired t-test results (F1) between Ours and the baseline models over 10 independent runs on three datasets

Dataset	Ours (Mean $\pm$ Std)	Baseline (Mean $\pm$ Std)	Δ (Ours$-$Baseline)	95% CI for Δ	t	p-value
BGL	0.9615 ± 0.00011	LogGPT: 0.9598 ± 0.00014	+0.00170	[0.00158, 0.00181]	33.84	8.5×10^{-11}
HDFS	0.9981 ± 0.00009	LogLLM: 0.9970 ± 0.00010	+0.00113	[0.00105, 0.00122]	30.46	2.2×10^{-10}
Thunderbird	0.9954 ± 0.00012	LogLLM: 0.9960 ± 0.00010	-0.00062	$[-0.00105, -0.00019]$	-3.23	1.1×10^{-2}

Table 6. Performance comparison of different preprocessing strategies on BGL, HDFS, and Thunderbird datasets.

Preprocessing	**BGL**			**HDFS**			**Thunderbird**		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Raw	**0.9672**	0.7553	0.8479	0.9932	0.9990	0.9948	0.8041	0.8640	0.8331
Template	0.9429	0.9711	0.9573	**0.9970**	**1.0000**	**0.9985**	0.9009	0.9395	0.9194
RE (Ours)	0.9429	**0.9808**	**0.9615**	0.9962	**1.0000**	0.9981	**0.9987**	**0.9865**	**0.9925**

Note: Boldface indicates the best performance in each column.

redundant noise while preserving essential semantic structure. This significantly enhances the model's generalization capability in recognizing anomaly patterns. The Template method, although capable of extracting structural templates, depends heavily on the design of the log parser. As a result, it may incorrectly remove constant tokens or retain variable ones, leading to information loss or residual noise. While Template typically outperforms Raw, it still falls short of RE in terms of stability and adaptability.

4.4.3 Accuracy analysis of long-term sequential log series

This section further provides an in-depth analysis of the performance of the FRLog model on long-span log sequences. To this end, we select the top five log groups with the largest number of log entries from the BGL, HDFS, and Thunderbird datasets as experimental objects. The number of log entries in these groups ranges from tens of thousands to hundreds of thousands, with time spans varying from several days to several tens of days. By conducting experiments on these long-span logs, the objective is to evaluate the modeling capability and accuracy of different fine-tuning strategies under conditions of extended time spans and massive log entries, thereby validating their effectiveness in improving anomaly detection performance.

The specific results are shown in the figure 7. In the comparative experiments on the three datasets, ReFT demonstrates the most stable and superior overall performance. On the BGL dataset, the F1 Score of ReFT consistently surpasses that of LoRA and QLoRA, and its advantage further expands as the training data scale increases; LoRA ranks second, while QLoRA exhibits a significant performance drop, with a large gap. This is mainly because BGL log sequences are relatively long with stronger contextual dependencies, where the quantization process of QLoRA weakens feature representation capability, whereas ReFT can better capture cross-event dependencies through fine-grained representation modulation.

On the HDFS dataset, all three methods maintain a high level of performance, with ReFT being slightly superior to LoRA, and QLoRA showing a minor disadvantage at the thousandths level. This is attributed to the highly regularized patterns and relatively low noise of HDFS logs, enabling all methods to extract stable features, and thus the differences are not significant.

On the Thunderbird dataset, ReFT and LoRA achieve similar performance, both remaining stable around 0.990, while QLoRA lags behind significantly, with overall scores below 0.968. This is due to the strong anomaly pattern perturbations in Thunderbird logs, where the low-precision quantization of QLoRA tends to lose discriminative information in the presence of subtle variations, while ReFT, with its parameter-efficient design, enhances cross-layer feature alignment and representation robustness.

Overall, the experimental results indicate that ReFT consistently exhibits greater stability and performance compared to LoRA across different datasets. The main reason is that although LoRA reduces training overhead, it is less effective at capturing fine-grained dependencies when dealing with long-span or noisy logs. In contrast, ReFT directly applies low-rank perturbations to the intermediate hidden space, allowing semantic features across layers to be adjusted more flexibly and precisely. This enables better modeling of cross-event dependencies in long sequences and stronger robustness in datasets with pronounced anomaly perturbations, such as Thunderbird. Even in highly regularized datasets like HDFS, ReFT still maintains a slight advantage through more effective feature alignment. Therefore, the ReFT fine-tuning strategy provides a more stable and generalizable solution for log anomaly detection in diverse environments.

4.4.4 Efficiency and memory usage analysis

After comparing the detection performance of different fine-tuning methods, it is also necessary to further analyze their efficiency and memory usage. For the task of log anomaly detection, the model must not only ensure detection accuracy but also balance training and inference speed as well as hardware resource consumption, which directly affects its deployability in large-scale and distributed environments. Therefore, this section conducts a systematic evaluation of the efficiency and resource usage of ReFT, LoRA, and QLoRA in terms of memory consumption, training time, and inference

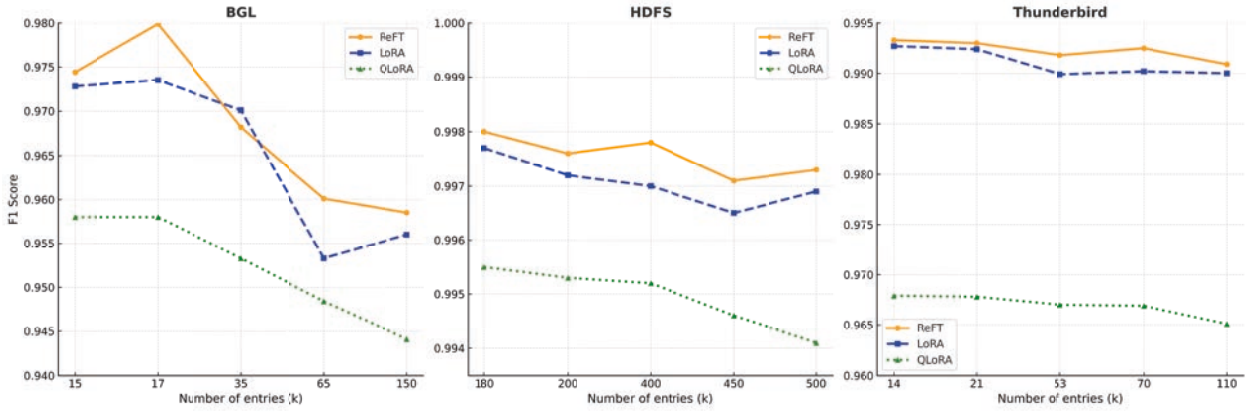


Figure 7. Performance of Different Fine-tuning Strategies on Long-span Log Sequences

Table 7. Results of different fine-tuning strategies on the BGL dataset

Fine-tuning Strategy	Memory Usage (GB)	Training Time (h)	Inference Latency per 1k Samples (s)
QLoRA	34	10	6.5
LoRA	60	21	6.0
ReFT	46	18	6.1

overhead, so as to verify the applicability of different fine-tuning strategies in practical scenarios.

As shown in Table 7, LoRA consumes the highest memory on the BGL dataset, followed by ReFT, while QLoRA requires the least. In terms of training time, LoRA takes 21 hours, ReFT is slightly faster, and QLoRA requires only 10 hours. Regarding inference latency, the differences among the three methods are small, all ranging between 6.0–6.5 seconds per 1k samples, with QLoRA being marginally slower. Overall, the BGL dataset contains long and structurally complex log entries, which leads to relatively high memory consumption and training time. However, the differences at the inference stage are smaller, mainly due to the quantization mechanism of QLoRA, which introduces relatively lightweight delays.

As shown in Table 8, the memory usage and training time of all three methods on the HDFS dataset are significantly higher than those on BGL and Thunderbird. Specifically, LoRA consumes 71 GB of memory and requires 35 hours of training; ReFT consumes 59 GB with 28 hours of training; and QLoRA consumes 44 GB with 15 hours of training. Notably, the inference latency shows the opposite trend: ReFT requires only 5.8 seconds per 1k samples, LoRA about 6.0 seconds, while QLoRA is slightly slower.

The root cause of this difference lies in the characteristics of the HDFS dataset. Since the log entries are shorter and the structures are highly repetitive, the sequential modeling burden during inference is relatively small, making inference faster. However, due to the massive volume of data, the training stage requires processing a significantly larger number of samples, leading to substantially higher memory and time overhead compared to other datasets.

As shown in Table 9, the memory usage on the Thunderbird dataset is similar to that on BGL, with LoRA consuming 66 GB, ReFT 57 GB, and QLoRA 37 GB. However, the overall training time is longer than that of BGL. The inference latency remains close to BGL, with all three methods falling between 6.0–6.6 seconds per 1k samples. The main reason is that Thunderbird log sequences are relatively long and the event structures are complex, which increases training costs compared to BGL. Nevertheless, due to the structural characteristics of the dataset, the inference stage maintains latency comparable to BGL.

Overall, QLoRA achieves the lowest memory and training time consumption, but suffers from the slowest inference and the most significant accuracy degradation. LoRA strikes a balance among the three aspects but incurs the highest overhead.

Table 8. Results of different fine-tuning strategies on the HDFS dataset

Fine-tuning Strategy	Memory Usage (GB)	Training Time (h)	Inference Latency per 1k Samples (s)
QLoRA	44	15	6.6
LoRA	71	35	6.0
ReFT	59	28	5.8

Table 9. Results of different fine-tuning strategies on the Thunderbird dataset

Fine-tuning Strategy	Memory Usage (GB)	Training Time (h)	Inference Latency per 1k Samples (s)
QLoRA	37	13	6.6
LoRA	66	25	6.0
ReFT	57	21	6.1

ReFT, on the other hand, provides a more balanced trade-off: while ensuring higher accuracy than QLoRA, it requires less memory and training time than LoRA, and maintains competitive inference efficiency, making it a more stable choice in practice.

4.5 Analysis of ablation experiment results

4.5.1 Analysis of the results of ablation experiments with fine-tuning strategies

To further evaluate the performance of FRLog under different PEFT strategies, this study conducts a comparative analysis on three representative log anomaly detection datasets: BGL, HDFS, and Thunderbird. Three mainstream PEFT methods are examined: LoRA, QLoRA, and ReFT, with the goal of systematically assessing their trade-offs between resource consumption and detection accuracy.

The experimental settings are as follows:

- **FRLog w/o LoRA**: LoRA modules are inserted into the attention and feed-forward layers of the LLaMA model, and only the inserted parameters are updated during training.
- **FRLog w/o QLoRA**: Builds upon LoRA by incorporating 4-bit quantization for weight compression.
- **FRLog w/ ReFT**: Employs a low-rank hidden state intervention mechanism to guide the task-specific adaptation of internal representations in the LLM.

All methods follow the same three-stage training procedure to ensure fairness. The training data ratio is kept fixed across all experiments, and F1 score is used as the primary evaluation metric on the three datasets.

As shown in the figure 8, on the BGL dataset—which is relatively well-structured and exhibits stable semantic templates—all three methods achieve comparable performance. Although ReFT is not the absolute best, it still demonstrates a certain performance advantage. On the HDFS dataset, which is highly structured with a relatively homogeneous anomaly distribution, LoRA performs the best. However, the differences among the three methods are minor, indicating that all fine-tuning strategies are effective in modeling datasets with strong regularities.

In contrast, on the Thunderbird dataset, which is structurally complex and characterized by diverse anomaly semantics, ReFT significantly outperforms both LoRA and QLoRA. The performance of LoRA and QLoRA declines considerably on this dataset, especially for QLoRA, which suffers from a noticeable degradation in semantic modeling ability due to extreme quantization. These results suggest that for log anomaly detection tasks involving high diversity and dynamically shifting data distributions, ReFT is better suited to task-specific modulation of internal representations, thereby achieving more robust performance.

4.5.2 Training data scaling ablation experiments

In order to simulate the deployment scenarios with scarce training samples in real logging systems, this paper designs training data ratio ablation experiments, controlling the proportion of training

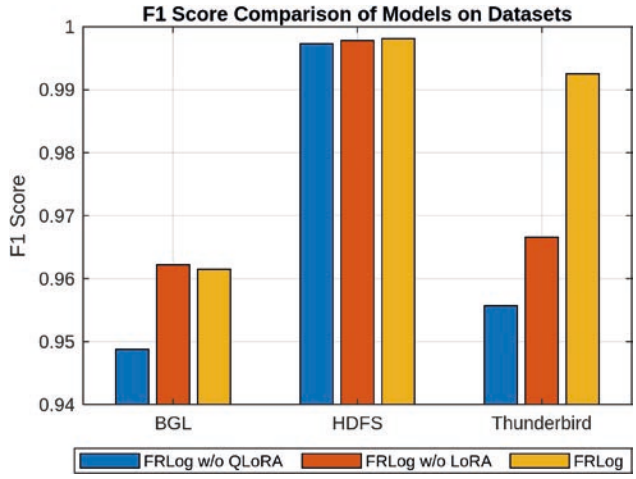


Figure 8. Comparison of F1 values of FRLog models under different fine-tuning strategies

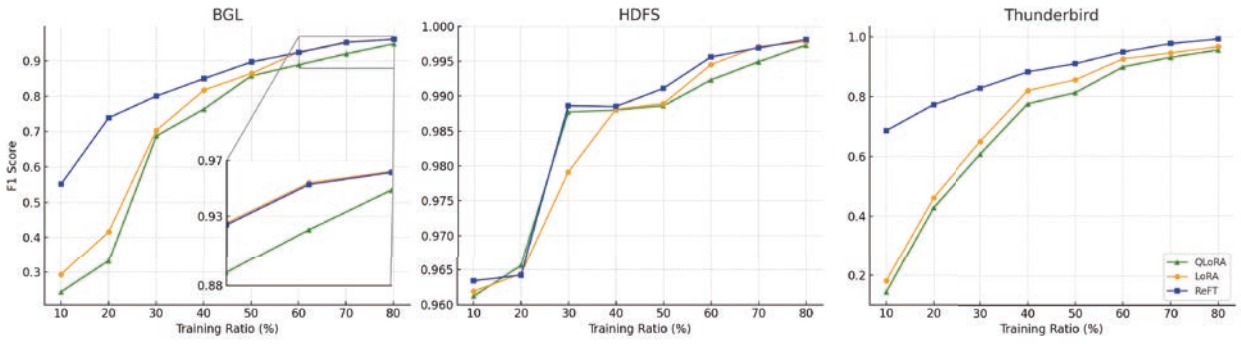


Figure 9. Comparison of F1 scores of LoRA, QLoRA and ReFT with different training data shares

data from 10% to 80%, and evaluates the performance changes of three PEFT strategies under the FRLog framework while keeping the test set unchanged.

From the figure 9, it can be clearly observed that there is a significant difference in the sample efficiency performance of different fine-tuning methods on the three datasets, especially when the proportion of training data is low, ReFT exhibits higher F1 scores and faster convergence on all datasets. This is due to the fact that the distribution of anomalies in the BGL dataset is relatively complex, thus requiring the model to have certain semantic modeling ability, and ReFT shows faster convergence and higher efficiency; HDFS has a fixed structure, concentrated anomalies, and a simple task, so the difference between the fine-tuning methods is not obvious; and Thunderbird has a loose structure and varied anomalies, so ReFT shows stronger modeling ability and sample robustness.

4.5.3 An experimental analysis of the ablation of a three-stage training strategy

To validate the practical contribution of the three-stage training strategy adopted by the FRLog model to its performance, we design a comprehensive ablation study targeting the structural impact of each stage. Specifically, the full training procedure includes:

- **Stage 1**: Fine-tuning the LLaMA model to learn the task prompt template for answering whether a log sequence is abnormal.
- **Stage 2**: Training the log encoder (BERT + Projector + ReFT) to produce embeddings aligned with the LLaMA decoder representations.
- **Stage 3**: Jointly fine-tuning the entire model to achieve end-to-end optimization.

To evaluate the contribution of each stage, we construct the following ablation variants by removing or skipping specific stages:

- **w/o Stage 1**: Skip the task-specific pretraining of LLaMA and directly perform joint training with the encoder.
- **w/o Stage 2**: Skip the independent training of the encoder and randomly initialize its parameters for joint fine-tuning with the decoder.
- **w/o Stage 3**: No joint fine-tuning is performed; only the outputs of Stage 1 and Stage 2 are retained.
- **w/o Stage 1&2**: Both Stage 1 and Stage 2 are skipped; the model is trained in a single stage from scratch.
- **FRLog**: All three stages are included and used as the performance reference.

As shown in Table 10, on the BGL dataset, removing the task-guided pretraining stage (Stage 1) leads to a significant degradation in the model's overall discriminative ability. This is mainly because the BGL dataset contains a large number of semantic templates, and Stage 1 plays a critical role in learning those templates with explicit task guidance. In addition, retaining only the encoder training stage (Stage 2) may further weaken the model's performance. When the input data lacks clear task-oriented guidance, the encoder is more prone to deviating from the essential semantics of the anomaly detection task.

The HDFS dataset originates from a distributed system and is highly structured, with fixed log formats and concentrated anomaly types that typically correspond to a few distinct system errors. It exhibits strong pattern regularity. Most anomaly detection tasks on this dataset can be accomplished by identifying the presence or absence of specific log template sequences. As a result, the model's performance relies less on complex training strategies.

Although the Thunderbird dataset is similar to BGL in that both are semi-structured or unstructured log datasets with content more akin to natural language, Thunderbird differs in several critical aspects. It has a more unstable structure, and its anomalies span multiple subsystems with highly diverse semantics. Consequently, the model must make contextualized judgments. The absence of any stage in the training strategy leads to noticeable performance degradation. In particular, the lack of task-guided pretraining severely undermines the model's ability to detect anomalies. This further highlights the importance of all three

Table 10. Experimental results of removing different training stages on three datasets.

Model	BGL/F1	HDFS/F1	Thunderbird/F1
w/o Stage 1	0.6982	0.9979	0.4237
w/o Stage 2	0.7792	0.9980	0.8580
w/o Stage 3	0.8254	0.9979	0.8508
w/o Stage 1&2	0.7550	0.9980	0.7737
FRLog	**0.9615**	**0.9981**	**0.9925**

Note: Boldface indicates the best performance in each column.

stages—guidance, alignment, and joint optimization—especially in complex, semantically diverse scenarios where each component of the three-stage training strategy is indispensable.

4.5.4 Analysis of the prompt ablation experiment

In this section, we design a set of systematic ablation experiments to evaluate the effect of different types of prompts on the performance of the FRLog model. Considering that prompt design may influence the representation distribution and attention patterns of large language models in the task of log anomaly detection, we conduct a comparison by varying only the prompt paradigms while keeping the datasets, model architecture, and training procedures unchanged.

Specifically, we select three representative types of prompts:

1. **Fixed description prompt (Fixed)**: provides the model with task context but does not include an explicit question, e.g., "Below is a sequence of system log messages."
2. **Question–answer prompt (Q&A)**: adds an explicit task instruction on top of the description, e.g., "Below is a sequence of system log messages. Is this sequence normal or anomalous?"
3. **Minimal prompt (Minimal)**: retains only a highly condensed task instruction, e.g., "Is this normal?"

By comparing the experimental results of these three types of prompts, we analyze the extent to which prompt design influences the detection performance of the model. The results are illustrated in Figure 10.

The results in the figure demonstrate that different prompt types have a significant impact on the detection performance of FRLog. The Q&A prompt explicitly provides task instructions, requiring the model to determine whether the log sequence is normal or anomalous. This minimizes semantic ambiguity and achieves the best detection performance. Although the Minimal prompt is more concise in form, retaining only the task instruction of whether the sequence is normal, its core information remains complete; therefore, its performance is only slightly lower than that of Q&A. In contrast, the Fixed prompt merely states that the input is a sequence of log messages without including any task-specific information. Without explicit task guidance, the model struggles to focus on the anomaly detection objective, resulting in significantly worse performance. Overall, the results indicate that whether the prompt contains clear task guidance is a key factor affecting the detection performance of the model.

5 Conclusion

This paper proposes FRLog, a log anomaly detection method based on a three-stage training strategy and the ReFT approach for large language models. FRLog integrates the semantic modeling capabilities of BERT and LLaMA, and employs ReFT to perform low-rank interventions on the intermediate hidden representation space of the language model, enabling precise modeling of anomalous semantics. Additionally, a three-stage collaborative training mechanism is introduced to effectively mitigate semantic drift and target misalignment during training, thereby improving the model's generalization ability and sample efficiency. Experimental results on three representative log datasets—BGL, HDFS, and Thunderbird—demonstrate that FRLog outperforms existing state-of-the-art methods in terms of Precision, Recall, and F1 score. In particular, it exhibits superior robustness and discriminative abil-

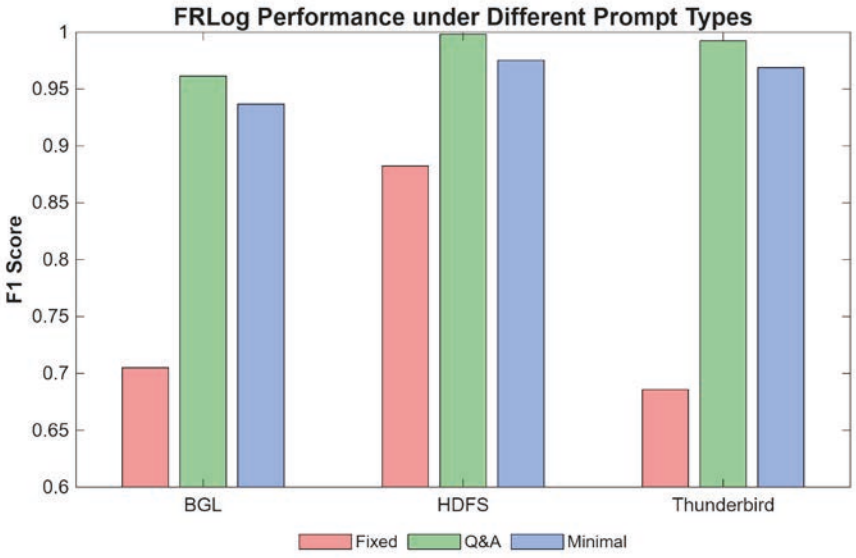


Figure 10. Comparison of FRLog Detection Performance under Different Prompt Types

ity in structurally complex and semantically diverse log environments.

Despite these promising results, FRLog also has certain limitations. Its performance still depends on the availability of sufficiently representative training data, and severe scarcity or distribution shifts in anomaly samples may degrade detection accuracy. Although ReFT reduces computational and memory overhead compared to LoRA, the model continues to incur noticeable training costs on very large-scale datasets. Moreover, while FRLog demonstrates robustness in structurally complex logs, its generalization to entirely novel log formats or unseen system behaviors may require further adaptation. Finally, the model is somewhat sensitive to prompt design, and inappropriate prompts may hinder anomaly discrimination ability. These limitations should be carefully considered when deploying FRLog in real-world log management systems.

Future work will further explore the application potential of FRLog in online log stream detection, cross-system log generalization, and root cause localization. In addition, we will investigate its deployment in large-scale distributed log management systems, focusing on fault tolerance, load balancing, and efficient synchronization. To further enhance scalability and privacy preservation, we plan to combine FRLog with federated learning frameworks, enabling collaborative model training across multiple organizations or data centers without sharing raw logs. This will not only improve robustness against heterogeneous data distributions but also address data security and regulatory constraints in practical scenarios. Moreover, further enhancements will be pursued by incorporating knowledge augmentation and multimodal information fusion to improve the model's comprehension and adaptability across various environments.

References

[1] Jia Tong, Li Ying, Wu Zhonghai. Review of fault diagnosis of distributed software system based on log data. *Journal of Software*, 2020, 31(7): 1997–2018.

[2] Qiu K, Zhang Y, Chen F. MSM: point cloud alignment algorithm through multi-scale feature fusion and cross-attention mechanism. *Intelligent Service Robotics*, 2025: 1–17.

[3] Qiu K, Zhang Y, Zhao J, et al. A Multimodal Sentiment Analysis Approach Based on a Joint Chained Interactive Attention Mechanism. *Electronics*, 2024, 13(10): 1922.

[4] Qiu K, Zhang Y, Ren Z, et al. SpemNet: A Cotton Disease and Pest Identification Method Based on Efficient Multi-Scale Attention and Stacking Patch Embedding. *Insects*, 2024, 15(9): 667.

[5] Qiu K, Yan M, Luo T, et al. FedAware: a distributed IoT intrusion detection method based on

fractal shrinking autoencoder. *Journal of King Saud University - Computer and Information Sciences*, 2025, 37(7): 1–21.

[6] Guo H, Yuan S, Wu X. Logbert: Log anomaly detection via BERT. In: *Proc. of IJCNN*, IEEE, 2021: 1–8.

[7] Le V H, Zhang H. Log-based anomaly detection without log parsing. In: *ASE*, IEEE, 2021: 492–504.

[8] Devlin J, Chang M W, Lee K, et al. BERT: Pre-training of deep bidirectional transformers for language understanding,2018. http://arxiv.org/abs/1810.04805

[9] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need. In: *Advances in Neural Information Processing Systems*, 2017.

[10] He S, Zhu J, He P, et al. Experience report: system log analysis for anomaly detection. In: *ISSRE*, IEEE, 2016.

[11] Xu W, Huang L, Fox A, et al. Detecting large-scale system problems by mining console logs. In: *SOSP*, ACM, 2009.

[12] Lin Q, Zhang H, Lou J G, et al. Log clustering based problem identification for online service systems. In: *ICSE-C*, ACM, 2016.

[13] Liang Y, Zhang Y, Xiong H, et al. Failure prediction in IBM BlueGene/L event logs. In: *ICDM*, IEEE, 2007.

[14] Bodik P, Goldszmidt M, Fox A, et al. Fingerprinting the datacenter: automated classification of performance crises. In: *EuroSys*, ACM, 2010.

[15] Chen M, Zheng A X, Lloyd J, et al. Failure diagnosis using decision trees. In: *Autonomic Computing*, IEEE, 2004.

[16] Liu Z, Qin T, Guan X, Jiang H, Wang C. An integrated method for anomaly detection from massive system logs. *IEEE Access*, 2018, 6: 30602–30611.

[17] Elmrabit N, Yang S, Yang L, Zhou H. Insider threat risk prediction based on Bayesian network. *Computers & Security*, 2020, 96: 101908.

[18] d'Ambrosio N, Perrone G, Romano S.P. Including insider threats into risk management through Bayesian threat graph networks. *Computers & Security*, 2023, 133: 103410.

[19] Rashid T, Agrafiotis I, Nurse J R C. A new take on detecting insider threats: Exploring the use of hidden Markov models. In: *MIST*, ACM, 2016: 47–56.

[20] Ye X, Han M. An improved feature extraction algorithm for insider threat using hidden Markov model on user behavior detection. *Information & Computer Security*, 2022, 30(1): 19–36.

[21] Liu F T, Ting K M, Zhou Z-H. Isolation-based anomaly detection. *ACM Trans. Knowl. Discov. Data*, 2012, 6(1): 1–39.

[22] Ding Z, Fei M. An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window. *IFAC Proc. Vol.*, 2013, 46(20): 12–17.

[23] Karev D, McCubbin C, Vaulin R. Cyber threat hunting through the use of an isolation forest. In: *CompSysTech*, 2017: 163–170.

[24] Du M, Li F, Zheng G, et al. DeepLog: anomaly detection and diagnosis from system logs through deep learning. In: *CCS*, ACM, 2017.

[25] Liu F, Wen Y, Zhang D, et al. Log2vec: a heterogeneous graph embedding based approach for detecting cyber threats within enterprise. In: *CCS*, ACM, 2019.

[26] Sasaki S, Suzuki J, Inui K. Subword-based compact reconstruction of word embeddings. In: *NAACL-HLT*, ACL, 2019.

[27] Wang J, Yu L C, Lai K R, et al. Dimensional sentiment analysis using a regional CNN-LSTM model. In: *ACL*, 2016.

[28] Luo L X. Network text sentiment analysis method combining LDA text representation and GRU-CNN. *Pers Ubiquit Comput*, 2019, 23: 405.

[29] Yu H, Yang J. A direct LDA algorithm for high-dimensional data—application to face recognition. *Pattern Recognition*, 2001, 34: 2067.

[30] Xiao T, Quan Z, Wang Z J, et al. Loader: A log anomaly detector based on transformer. *IEEE Trans. Serv. Comput.*, 2023, 16(5): 3479–3492.

[31] Guo H, Yuan S, Wu X. Logbert: Log anomaly detection via BERT. In: *IJCNN*, IEEE, 2021: 1–8.

[32] Yu J, Hu Z, Jiang C. Multi-feature-based log event anomaly detection. *Computer Engineering and Science*, 2024, 46(09): 1587–1597.

[33] Qiu K, Zhang Y, Feng Y, et al. LogAnomEX: An Unsupervised Log Anomaly Detection Method Based on Electra-DP and Gated Bilinear Neural Networks. *Journal of Network and Systems Management*, 2025, 33(2): 1–29.

[34] Qi J, Huang S, Luan Z, et al. LogGPT: Exploring ChatGPT for log-based anomaly detection. In: *HPCC/DSS/SmartCity/DependSys*, IEEE, 2023: 273–280.

[35] He M, Jia T, Duan C, et al. LLMeLog: An Approach for Anomaly Detection based on LLM-enriched Log Events. In: *ISSRE*, IEEE, 2024: 132–143.

[36] Han X, Yuan S, Trabelsi M. LogGPT: Log anomaly detection via GPT. In: *BigData*, IEEE, 2023: 1117–1122.

[37] Hadadi F, Xu Q, Bianculli D, et al. Anomaly detection on unstable logs with GPT models. *arXiv preprint* arXiv:2406.07467, 2024.

[38] Guan W, Cao J, Qian S, et al. LogLLM: Log-based Anomaly Detection Using Large Language Models. *arXiv preprint* arXiv:2411.08561, 2024.

[39] Lim Y F, Zhu J, Pang G. Adapting Large Language Models for Parameter-Efficient Log Anomaly Detection. *arXiv preprint* arXiv:2503.08045, 2025.

[40] Nasser H, Trad F, Chehab A. Stacking Large Language Models is All You Need: A Case Study on Phishing URL Detection. *Journal of Artificial Intelligence and Soft Computing Research*, 2025, 15(4): 337–356.

[41] Romaszewski M, Sekuła P, Głomb P, et al. Through the Thicket: A Study of Number-Oriented LLMs Derived from Random Forest Models. *Journal of Artificial Intelligence and Soft Computing Research*, 2025, 15(3): 279–298.

[42] Oliner A J, Stearley J. What Supercomputers Say: A Study of Five System Logs. In: *DSN*, IEEE/IFIP, 2007.

[43] Zhu J, He S, He P, et al. Loghub: A Large Collection of System Log Datasets for AI-driven Log Analytics. In: *ISSRE*, IEEE, 2023.

[44] Xu W, Huang L, Fox A, et al. Detecting Large-Scale System Problems by Mining Console Logs. In: *SOSP*, ACM, 2009.

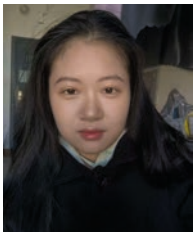
[45] Lin Y, Deng H, Li X. FastLogAD: Log Anomaly Detection with Mask-Guided Pseudo Anomaly Generation and Discrimination. *arXiv preprint* arXiv:2404.08750, 2024.

[46] Li Z, Shi J, Van Leeuwen M. Graph neural networks based log anomaly detection and explanation. In: *ICSE Companion*, IEEE/ACM, 2024: 306–307.

Keyuan Qiu received the B.S. degree in Software Engineering from Southwest Minzu University in 2023 and is currently pursuing the M.S. degree in Electronic Information at Shihezi University. His research interests include network intrusion detection and system anomaly detection.
https://orcid.org/0009-0005-6613-5711

Zhejie Xu is currently pursuing a B.S. degree in Software Engineering at Shihezi University.His research interest is deep learning.
https://orcid.org/0009-0001-7851-8032

Luo Tao received the B.S. degree in Computer Science and Technology from Guangzhou University and the M.S. degree in Electronic Information from Shihezi University. His research interests include intelligent computing and network security.
https://orcid.org/0009-0000-3223-626X



Meifang Yan received the B.S. degree in Software Engineering from Pingdingshan University and is currently pursuing the M.S. degree in Computer Technology at Xinjiang Normal University. Her research interests include bioinformatics and medical informatics.
https://orcid.org/0009-0007-1192-4216

Ruru Liu received the B.S. degree in Software Engineering from Pingdingshan University and the M.S. degree in Electronic Information from Shihezi University. She is currently pursuing the Ph.D. degree in Artificial Intelligence at Jilin University. Her research interests include path planning and deep learning.
https://orcid.org/0009-0002-4822-4399

Pengjin Liu received the B.S. degree in Civil Engineering from North China University of Water Resources and Electric Power and is currently pursuing the M.S. degree in Electronic Information at Shihezi University. His research interests include image processing, computer vision, and bioinformatics.
https://orcid.org/0009-0000-6404-0560

Feng Chen received the B.S. degree in Information Management and Information System from Donghua University of Science and Technology in 2005, and the M.S. degree in Industrial Economics from Shihezi University in 2009. She is currently an associate professor in the College of Information Science and Technology of Shihezi University. Her research interests include computer applications, database technology research, and software design methods.
https://orcid.org/0009-0001-1943-9094