LogLLM: Log-based Anomaly Detection Using Large Language Models

Wei Guan[1], Jian Cao[1*], Shiyou Qian[1], Jianqi Gao[1], Chun Ouyang[2]
[1]*Department of Computer Science and Engineering, SJTU, Shanghai, China*
[2]*The School of Information Systems, QUT, Brisbane, Australia*
{guan-wei, cao-jian, qshiyou, 193139}@sjtu.edu.cn, c.ouyang@qut.edu.au

***Abstract*—Software systems often record important runtime information in logs to help with troubleshooting. Log-based anomaly detection has become a key research area that aims to identify system issues through log data, ultimately enhancing the reliability of software systems. Traditional deep learning methods often struggle to capture the semantic information embedded in log data, which is typically organized in natural language. In this paper, we propose LogLLM, a log-based anomaly detection framework that leverages large language models (LLMs). LogLLM employs BERT for extracting semantic vectors from log messages, while utilizing Llama, a transformer decoder-based model, for classifying log sequences. Additionally, we introduce a projector to align the vector representation spaces of BERT and Llama, ensuring a cohesive understanding of log semantics. Unlike conventional methods that require log parsers to extract templates, LogLLM preprocesses log messages with regular expressions, streamlining the entire process. Our framework is trained through a novel three-stage procedure designed to enhance performance and adaptability. Experimental results across four public datasets demonstrate that LogLLM outperforms state-of-the-art methods. Even when handling unstable logs, it effectively captures the semantic meaning of log messages and detects anomalies accurately.**

***Index Terms*—System log, anomaly detection, large language model, deep learning, log analysis**

I. Introduction

Ensuring high availability and reliability is crucial for large-scale software-intensive systems [1], [2]. As these systems become more complex and expansive, the occurrence of anomalies becomes unavoidable [3], [4]. Even a minor issue can lead to performance degradation, data integrity problems, and substantial losses in both customers and revenue. Therefore, anomaly detection is vital for maintaining the health and stability of complex software-intensive systems [5].

Software-intensive systems typically produce console logs that record system states and critical runtime events [6]. Engineers can utilize this log data to evaluate system health, identify anomalies, and trace the root causes of issues. However, due to the potentially vast volume of logs, manually analyzing them for anomalies can be both labor-intensive and prone to mistakes [7]. Consequently, log-based anomaly detection has emerged as a key area in automated log analysis, focusing on the automatic identification of system anomalies through log data.

Numerous deep learning-based methods [8]–[22] for log-based anomaly detection have been proposed. These methods typically employ sequential deep learning models such as LSTM [23] and transformers [24]. These methods can be further divided into reconstruction-based methods [8]–[15] and binary classification-based methods [16]–[22]. Reconstruction-based methods involve designing and training a deep neural network to reconstruct input log sequences, with anomalies detected based on reconstruction errors. The underlying principle is that anomalous samples cannot be accurately reconstructed. Binary classification-based methods, on the other hand, involve designing a binary classifier to classify samples as either normal or anomalous. These methods often require labeled anomalies for training purposes. It is recognized that system logs are documented in natural language and contain a significant amount of semantic information. Nevertheless, traditional deep learning-based methods struggle to effectively capture this information.

In recent years, significant advancements have been achieved in LLMs, such as GPT-4 [25], Llama 3 [26], and ChatGLM [27]. These models are characterized by their vast parameter sizes and are pretrained on substantially larger datasets, ranging from several gigabytes to terabytes in size. This extensive pretraining equips them with remarkable language comprehension abilities, enabling superior performance in tasks such as summarization, paraphrasing, and instruction following even in zero-shot scenarios [28]. Existing methods that utilize LLMs for log-based anomaly detection can be categorized into prompt engineering-based [7], [29]–[31] and fine-tuning-based [3], [32]–[40] approaches. Prompt engineering-based methods leverage the zero/few-shot capabilities of LLMs to detect anomalies based solely on the models' internal knowledge. However, these methods often struggle to customize solutions for specific datasets, leading to suboptimal detection performance. Fine-tuning-based methods integrate LLMs into deep neural networks and tailor them to user-specific datasets. Nevertheless, these methods encounter challenges such as limited semantic understanding, suboptimal LLM utilization (relying solely on LLMs for semantic information extraction), and insufficient consideration of input data format, which can lead to memory overflow.

To tackle the aforementioned challenges, we propose LogLLM, a novel log-based anomaly detection framework that harnesses LLMs. Unlike traditional methods that rely on

*Corresponding author.

LogLLM：基于日志的大语言模型异常检测

Wei Guan1, 曹建 [1*], 钱石友 1, 高建奇 1, 欧阳春 ang[2]
[1]*Department of Computer Science and Engineering, SJTU, Shanghai, China*
The 昆士兰科技大学信息系统学院，布里斯班，澳大利亚 *a*
{关伟，曹健，钱石友，193139}@sjtu.edu.cn, c.ouyang@qut.edu.au

摘要 —— **软件系统通常在日志中记录重要的运行时信息，以帮助故障排除。基于日志的异常检测已成为一个关键的研究领域，旨在通过日志数据识别系统问题，最终提高软件系统的可靠性。传统的深度学习方法往往难以捕捉嵌入在日志数据中的语义信息，这些数据通常以自然语言组织。在本文中，我们提出了 LogLLM，一个基于日志的异常检测框架，该框架利用大型语言模型（LLMs）。LogLLM 使用 BERT 从日志消息中提取语义向量，同时利用基于 transformer 解码器的 Llama 对日志序列进行分类。此外，我们引入了一个投影器，以对齐 BERT 和 Llama 的向量表示空间，确保对日志语义的连贯理解。与需要日志解析器提取模板的传统方法不同，LogLLM 使用正则表达式预处理日志消息，简化了整个过程。我们的框架通过一个新颖的三阶段程序进行训练，旨在提高性能和适应性。在四个公开数据集上的实验结果表明，LogLLM 优于最先进的方法。即使处理不稳定的日志，它也能有效地捕捉日志消息的语义意义并准确检测异常。**

关键词：系统日志，异常检测，大语言模型，深度学习，日志分析

I. 引言

确保高可用性和可靠性对于大规模软件密集型系统至关重要。随着这些系统变得更加复杂和庞大，异常的发生变得不可避免。即使是微不足道的问题也可能导致性能下降、数据完整性问题，以及客户和收入的大量损失。因此，异常检测对于维护复杂软件密集型系统的健康和稳定性至关重要。

软件密集型系统通常会产生控制台日志，记录系统状态和关键运行时事件。工程师可以利用这些日志数据来评估系统健康、识别异常和追踪问题的根本原因。然而，由于日志的潜在大量，手动分析它们以查找异常既费时又容易出错。因此，基于日志的异常检测已成为自动日志分析的关键领域，专注于通过日志数据自动识别系统异常。

近年来，针对基于日志的异常检测，已经提出了许多基于深度学习的算法。这些方法通常采用序列深度学习模型，如 LSTM 和 Transformer。这些方法可以进一步分为基于重建的方法和基于二分类的方法。基于重建的方法涉及设计和训练一个深度神经网络来重建输入日志序列，异常检测基于重建误差。其基本原理是异常样本无法被准确重建。另一方面，基于二分类的方法涉及设计一个二分类器来将样本分类为正常或异常。这些方法通常需要标记的异常样本进行训练。众所周知，系统日志是用自然语言编写的，并且包含大量的语义信息。然而，传统的基于深度学习的算法在有效捕获这些信息方面存在困难。

近年来，LLM（大型语言模型）取得了显著的进展，如 GPT、Llama 和 ChatGLM。这些模型以其庞大的参数规模为特征，并在大量数据集上进行了预训练，数据集大小从几个 GB 到 TB 不等。这种广泛的预训练使它们具备了卓越的语言理解能力，即使在零样本或少量样本的情况下，也能在摘要、释义和指令遵循等任务中表现出色。现有的利用 LLM 进行基于日志的异常检测的方法可以分为基于提示工程的方法和基于微调的方法。基于提示工程的方法利用 LLM 的零样本 / 少量样本能力，仅基于模型内部知识来检测异常。然而，这些方法往往难以针对特定数据集定制解决方案，导致检测性能不佳。基于微调的方法将 LLM 集成到深度神经网络中，并针对用户特定的数据集进行定制。然而，这些方法面临着诸如语义理解有限、LLM 利用不足（仅依赖 LLM 进行语义信息提取）以及未充分考虑输入数据格式等问题，可能导致内存溢出。

为了应对上述挑战，我们提出了 LogLLM，这是一种基于日志的异常检测框架，它利用了大型语言模型。与传统方法不同，传统方法依赖于

通讯作者。

log parsers for template extraction, LogLLM preprocesses log messages using regular expressions, thereby streamlining the entire process. LogLLM, a fine-tuning-based method, utilizes BERT, a transformer encoder-based model, to extract semantic vectors from log messages. Additionally, it employs Llama, a transformer decoder-based model, to classify log sequences. To ensure coherence in log semantics, we introduce a projector that aligns the vector representation spaces of BERT and Llama. Our framework is trained using a novel three-stage procedure designed to enhance both performance and adaptability.

As illustrated in Section V-G, LLMs frequently face out-of-memory challenges due to their extensive parameter sizes [41]. Directly inputting the entire log sequence (by concatenating log messages into a long string) into Llama can lead to out-of-memory issues and potentially confuse the LLM, making it difficult to focus on key points for distinguishing anomalies. By adopting BERT to summarize each log message, LogLLM effectively mitigates these problems. We conduct experiments across four public datasets, and the results demonstrate that LogLLM outperforms state-of-the-art methods. Even when handling unstable logs, where new log templates frequently emerge due to software evolution, it effectively captures the semantic meaning of log messages and detects anomalies accurately. The ablation study confirms the effectiveness of the three-stage training procedure.

The main contributions of our work are as follows:

- We introduce LogLLM, a novel log-based anomaly detection framework leveraging LLMs. This study marks the first attempt to simultaneously employ transformer encoder-based and decoder-based LLMs, specifically BERT and Llama, for log-based anomaly detection.
- We propose a novel three-stage procedure to optimize the training and coordination of different components within the deep model, enhancing both performance and adaptability.
- We conduct extensive experiments on four publicly available real-world datasets, demonstrating that LogLLM achieves exceptional performance.

II. Related Work

In this section, we explore related work in the field of log-based anomaly detection, with a particular focus on deep learning-based methods. We give special attention to approaches that utilize pretrained LLMs.

A. Traditional Deep Learning for Log-based Anomaly Detection

Many traditional deep learning-based methods for log-based anomaly detection have been proposed. These works can be grouped into two types based on the training paradigm: reconstruction-based methods and binary classification-based methods.

Reconstruction-based methods [8]–[15] involve designing and training a deep neural network to reconstruct input log sequences. Anomalies are detected based on reconstruction errors. Normal log sequences can be reconstructed with minimal errors, while anomalous log sequences cannot be effectively reconstructed, resulting in significantly higher reconstruction errors. These methods consistently train the deep model on normal data that is free of anomalies, which means they are semi-supervised.

DeepLog [8] adopts LSTM to predict the next log template ID based on past log sequences. Similarly, LogAnomaly [9] predicts the next log template ID based on both sequential and quantitative patterns. Autoencoders (AEs) [10]–[13] and generative adversarial networks (GANs) [14], [15] are widely used in reconstruction-based methods. For example, LogAttn [10] adopts an AE that incorporates a temporal convolutional network (TCN) to capture temporal semantic correlations and a deep neural network (DNN) to capture statistical correlations. Duan et al. [14] use a GAN, where an encoder-decoder framework based on LSTM serves as the generator. Convolutional neural networks (CNNs) are used as the discriminator. The reconstruction error is calculated based on the difference between the input and the output from the generator.

Binary classification-based methods [16]–[22] often employ deep neural networks that output either one or two values. Typically, a single value represents the probability that a sample belongs to the anomalous class, and anomalies are detected by applying a threshold to convert this probability into a binary classification. When two values are output, they represent the probabilities of the sample belonging to the normal and anomalous classes, respectively.

Most methods [16]–[20] typically train deep models in a supervised manner. For example, Zhang et al. [16] propose LayerLog, which integrates word, log, and logseq layers to extract semantic features from log sequences. CNNs are utilized in [17], [18] to develop a binary classifier. LogRobust [19] integrates a pre-trained Word2Vec model, specifically FastText [42], and combines it with TF-IDF weights to learn representation vectors of log templates. These vectors are then fed into an attention-based Bi-LSTM model for anomaly detection. LogGD [20] transforms log sequences into graphs and utilizes a graph transformer neural network that combines graph structure and node semantics for log-based anomaly detection.

Some work [21], [22] involves training binary classifiers in a semi-supervised manner. For example, Trine [21] uses a transformer encoder [24] to encode normal log sequences into vector representations and a generator to produce random fake vector representations. The discriminator, which is composed of a transformer and a multi-layer perceptron (MLP), is trained to distinguish whether the given vector representations are normal log sequences and it is subsequently used to detect anomalies. PLELog [22] tackles the challenge of insufficient labeling by employing probabilistic label estimation and develops an attention-based GRU neural network for anomaly detection.

It is acknowledged that system logs are recorded in natural

LogLLM 通过正则表达式预处理日志消息，从而简化了整个流程。LogLLM 是一种基于微调的方法，利用基于 Transformer 编码器的模型 BERT 从日志消息中提取语义向量。此外，它还采用基于 Transformer 解码器的模型 Llama 对日志序列进行分类。为了确保日志语义的一致性，我们引入了一个投影器，以对齐 BERT 和 Llama 的向量表示空间。我们的框架采用了一种新颖的三阶段训练程序，旨在提高性能和适应性。

如第 V-G 节所述，LLM 由于其庞大的参数规模，经常面临内存不足的问题 [41]。直接将整个日志序列（通过将日志消息连接成一个长字符串）输入到 Llama 中可能导致内存不足问题，并可能使 LLM 困惑，使其难以关注区分异常的关键点。通过采用 BERT 对每个日志消息进行总结，LogLLM 有效地缓解了这些问题。我们在四个公开数据集上进行了实验，结果表明 LogLLM 优于现有方法。即使处理不稳定的日志，由于软件演变，新日志模板频繁出现，它也能有效地捕捉日志消息的语义意义并准确检测异常。消融研究证实了三阶段训练程序的有效性。

我们工作的主要贡献如下：

我们引入了 LogLLM，这是一个利用 LLM 的基于日志的异常检测框架。这项研究是首次尝试同时使用基于 Transformer 编码器和解码器的 LLM，即 BERT 和 Llama，进行基于日志的异常检测。

我们提出了一种新颖的三阶段程序，以优化深度模型中不同组件的训练和协调，从而提高性能和适应性。

我们在四个公开可用的真实世界数据集上进行了广泛的实验，证明了 LogLLM 取得了卓越的性能。

II. 相关工作

在本节中，我们探讨了基于日志的异常检测领域的相关工作，特别关注基于深度学习的方法。我们特别关注利用预训练大型语言模型（LLM）的方法。

A. 基于传统深度学习的日志异常检测

针对基于日志的异常检测，已经提出了许多基于传统深度学习的方法。根据训练范式，这些工作可以分为两类：基于重建的方法和基于二分类的方法。

基于重建的方法 –[8][15] 涉及设计和训练一个深度神经网络来重建输入日志

基于重建的方法 {{v2}}–{{v4}} 包括设计和训练一个深度神经网络来重建输入日志序列。异常检测基于重建误差。正常的日志序列可以以最小的误差重建，而异常的日志序列则无法有效重建，导致显著更高的重建误差。这些方法始终在无异常的正常数据上训练深度模型，这意味着它们是半监督的。

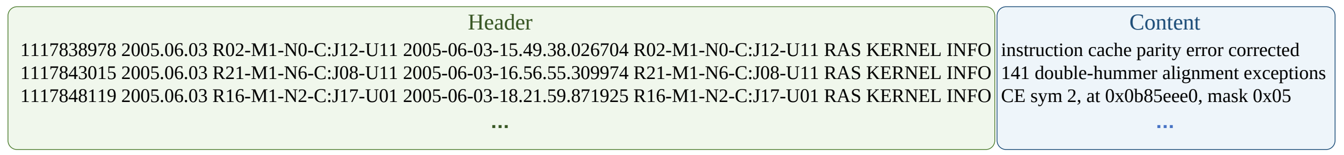
DeepLog [8] 采用 LSTM 预测基于过去日志序列的下一个日志模板 ID。类似地，LogAnomaly [9] 基于序列和定量模式预测下一个日志模板 ID。自编码器（AEs）[10]–[13] 和生成对抗网络（GANs） [14], [15] 在基于重建的方法中得到广泛应用。例如，LogAttn [10] 采用一个结合时间卷积网络（TCN）以捕获时间语义相关性和深度神经网络（DNN）以捕获统计相关性的自编码器。Duan 等人 [14] 使用一个 GAN，其中基于 LSTM 的编码器 - 解码器框架作为生成器。卷积神经网络（CNNs）用作判别器。重建误差基于输入和生成器输出的差异计算。

基于二分类的方法 [16]–[22] 通常使用输出一个或两个值的深度神经网络。通常，一个值表示样本属于异常类的概率，通过应用阈值将此概率转换为二进制分类来检测异常。当输出两个值时，它们分别表示样本属于正常类和异常类的概率。

大多数方法 [16]–[20] 通常以监督方式训练深度模型。例如，张等人 [16] 提出了 LayerLog，该模型整合了词、日志和日志序列层以从日志序列中提取语义特征。CNNs 在 [17], [18] 中被用于开发二分类器。LogRobust[19] 整合了一个预训练的 Word2Vec 模型，特别是 FastText [42], ，并将其与 TF-IDF 权重结合来学习日志模板的表示向量。然后，这些向量被输入到一个基于注意力的 Bi-LSTM 模型中进行异常检测。LogGD [20] 将日志序列转换为图，并利用一个结合图结构和节点语义的图变换神经网络进行基于日志的异常检测。

一些工作 [21], [22] 涉及以半监督方式训练二元分类器。例如，Trine [21] 使用 Transformer 编码器 [24] 将正常日志序列编码成向量表示，并使用生成器生成随机的虚假向量表示。判别器由 Transformer 和多层感知器（MLP）组成，用于区分给定的向量表示是否为正常日志序列，随后用于检测异常。PLELog [22] 通过采用概率标签估计和开发基于注意力的 GRU 神经网络来解决标签不足的挑战，用于异常检测。

公认的是，系统日志以自然的方式记录。



```
Header                                                                                                    Content
1117838978 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.49.38.026704 R02-M1-N0-C:J12-U11 RAS KERNEL INFO  instruction cache parity error corrected
1117843015 2005.06.03 R21-M1-N6-C:J08-U11 2005-06-03-16.56.55.309974 R21-M1-N6-C:J08-U11 RAS KERNEL INFO  141 double-hummer alignment exceptions
1117848119 2005.06.03 R16-M1-N2-C:J17-U01 2005-06-03-18.21.59.871925 R16-M1-N2-C:J17-U01 RAS KERNEL INFO  CE sym 2, at 0x0b85eee0, mask 0x05
...                                                                                                       ...
```

Fig. 1: An example of a system log.

language and contain a substantial amount of semantic information. However, traditional deep learning-based methods face challenges in capturing this information.

B. LLMs for Log-based Anomaly Detection

Existing LLMs can be categorized into transformer encoder-based models, such as BERT [43], RoBERTa [44], and Span-BERT [45], and transformer decoder-based models, including GPT-4 [25], Llama 3 [26], and ChatGLM [27]. Two prevalent strategies for utilizing LLMs are prompt engineering and fine-tuning.

Prompt engineering-based methods [7], [29]–[31] detect anomalies solely by relying on the internal knowledge of LLMs. These methods typically employ transformer decoder-based models. For instance, Qi et al. [7] employ ChatGPT for zero-shot and few-shot log-based anomaly detection, utilizing prompt templates that integrate the log sequence directly. However, this approach becomes impractical when using a large window size for grouping log messages. Egersdoerfer et al. [30] address this issue by maintaining a summary-based memory, which summarizes the previous log messages, eliminating the need to input the entire log sequence for anomaly detection. RAGLog [31] uses a retrieval augmented generative (RAG) framework [46] to analyze log entries by querying its store of samples of normal log entries. They design prompt templates for LLMs to determine whether a queried log entry is normal or abnormal. Prompt engineering-based methods often struggle to customize solutions for specific datasets, which can lead to suboptimal detection performance in particular datasets.

Fine-tuning-based methods [3], [32]–[40] incorporate LLMs into deep neural networks and customize them to the user's own dataset. Some methods [32]–[35], although adopting transformer encoder-based LLMs for anomaly detection, do not capture the semantic information within log sequences. For example, LogBERT [32] and LAnoBERT [33] utilize BERT to reconstruct the input sequence of log template IDs (IDs of log string templates) and detect anomalies based on reconstruction errors, disregarding the semantic information. Other methods [3], [36]–[39] use transformer encoder-based LLMs solely for extracting semantic information from log messages and then employ either smaller models [3], [36]–[38] or distance-based comparison [39] for classification. For instance, NeuralLog [3] leverages BERT to extract semantic vectors from raw log messages, which are subsequently used to detect anomalies via a transformer-based classification model. Similarly, RAPID [39] utilizes transformer encoder-based models to extract semantic vectors and performs anomaly detection by comparing each query log sequence with its nearest document log sequence. Hadadi et al. [40] directly input template sequences parsed from log sequences, into GPT models and fine-tune it to accurately predict sequence labels. However, this approach faces two key challenges. First, the boundaries between templates within the sequences are unclear, making it difficult for the model to learn the sequential dependencies. Second, each template may be tokenized into multiple tokens by the LLM's tokenizer, and a single sequence can contain numerous log templates. As a result, an excessive number of tokens may be generated, often exceeding the token (memory) limits of LLMs [41], thereby restricting the length of sequences that can be processed. These two challenges are further demonstrated in Section V-G.

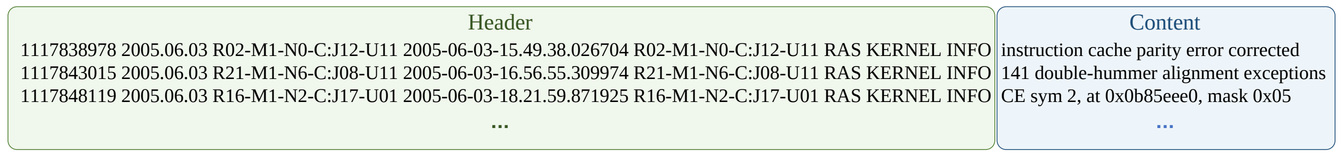
LogLLM is a fine-tuning-based method that utilizes BERT for extracting semantic vectors from log messages and Llama, a transformer decoder-based model, for log sequence classification. This method aligns the vector representation spaces of BERT and Llama using a projector. The use of BERT ensures clear boundaries between log messages, as each message is represented by a distinct embedding vector, thereby enhancing classification performance. Moreover, when memory and parameter size of Llama are held constant, this approach can handle longer sequences compared to directly tokenizing the entire log sequence using Llama's tokenizer.

III. Preliminaries

To establish the groundwork for subsequent sections, we introduce the **system log**, which records the system's events and internal states during runtime. A system log contains a list of log messages in chronological order.

Fig. 1 presents a snippet of a raw system log generated by the BGL (the BlueGene/L supercomputer system), with each log message ordered according to the recorded time. These raw log messages are semi-structured texts consisting of a **header** and **content**. The header, determined by the logging framework, includes information such as timestamp, verbosity level (e.g., WARN/INFO), and component [47]. The log content comprises a constant part (keywords that reveal the log template) and a variable part (parameters that carry dynamic runtime information). In this paper, we focus solely on the content of each log message.

The log messages can be grouped into **log sequences** (i.e., series of log messages that record specific execution flows) based on session or fixed/sliding windows [48]. **Session window** partitioning groups log messages according to their session IDs, thereby generating sequences that include the log



```
Header                                                                                                    Content
1117838978 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.49.38.026704 R02-M1-N0-C:J12-U11 RAS KERNEL INFO  instruction cache parity error corrected
1117843015 2005.06.03 R21-M1-N6-C:J08-U11 2005-06-03-16.56.55.309974 R21-M1-N6-C:J08-U11 RAS KERNEL INFO  141 double-hummer alignment exceptions
1117848119 2005.06.03 R16-M1-N2-C:J17-U01 2005-06-03-18.21.59.871925 R16-M1-N2-C:J17-U01 RAS KERNEL INFO  CE sym 2, at 0x0b85eee0, mask 0x05
...                                                                                                       ...
```

图 1：系统日志的一个示例。

语言并包含大量语义信息。然而，传统的基于深度学习的传统方法在捕捉这些信息方面面临挑战。

B. 基于日志的异常检测的 *LLM*

现有的 LLM 可以分为基于 transformer 编码器的模型，如 BERT [43]，RoBERTa [44]，和 Span-BERT [45],，以及基于 transformer 解码器的模型，包括 GPT-4 [25]，Llama 3 [26]，和 ChatGLM [27]。利用 LLM 的两种常见策略是提示工程和微调。

基于提示工程的方法仅通过依赖 LLM 的内部知识来检测异常。这些方法通常采用基于 transformer 解码器的模型。例如，Qi 等人 [7] 使用 ChatGPT 进行零样本和少量样本的基于日志的异常检测，利用将日志序列直接整合到提示模板中的提示模板。然而，当使用大窗口大小对日志消息进行分组时，这种方法变得不切实际。Egersdoerfer 等人 [30] 通过维护基于摘要的内存来解决此问题，该内存总结了之前的日志消息，从而消除了在异常检测中输入整个日志序列的需要。RAGLog [31] 使用检索增强生成（ RAG ）框架 [46] 通过查询其存储的正常日志条目样本来分析日志条目。他们为 LLM 设计提示模板以确定查询的日志条目是正常还是异常。基于提示工程的方法通常难以针对特定数据集定制解决方案，这可能导致特定数据集中的检测性能不佳。

基于微调的方法将 LLM 纳入深度神经网络并将它们定制到用户的自己的数据集。一些方法 [32]–[35]，虽然采用基于 transformer 编码器的 LLM 进行异常检测，但没有捕捉到日志序列中的语义信息。例如，LogBERT [32] 和 LAnoBERT [33] 使用 BERT 重建日志模板 ID （日志字符串模板的 ID ）的输入序列，并根据重建错误检测异常，忽略了语义信息。其他方法 [3], [36]–[39] 使用基于 transformer 编码器的 LLM 仅从日志消息中提取语义信息，然后使用较小的模型 [3], [36]–[38] 或基于距离的比较 [39] 进行分类。例如，NeuralLog [3] 利用 BERT 从原始日志消息中提取语义向量，这些向量随后用于通过基于 transformer 的分类模型检测异常。同样，RAPID [39] 利用基于 transformer 编码器的模型提取语义向量并执行异常检测。通过将每个查询日志序列与其最近的文档日志序列进行比较来进行检测。Hadadi 等人 [40] 直接将解析自日志序列的模板序列输入到 GPT 模型中，并对其进行微调以准确预测序列标签。然而，这种方法面临两个关键挑战。首先，序列中模板之间的边界不明确，这使得模型难以学习序列依赖关系。其次，每个模板可能被 LLM 的标记器标记成多个标记，并且单个序列可以包含多个日志模板。因此，可能会生成过多的标记，通常超过 LLM 的标记（内存）限制，从而限制了可以处理的序列长度。这两个挑战在第五节 -G 中进一步演示。

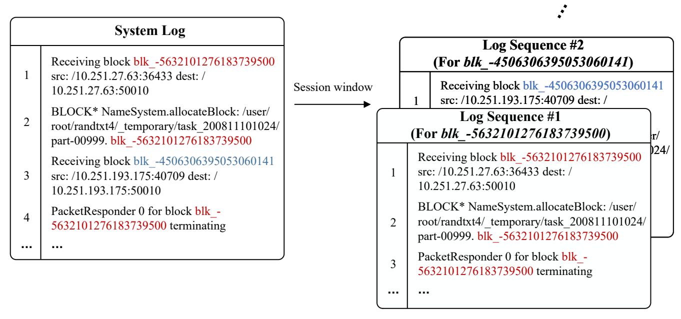
LogLLM 是一种基于微调的方法，它使用 BERT 从日志消息中提取语义向量，并使用 Llama （一个基于 transformer 解码器的模型）进行日志序列分类。该方法使用投影仪对 BERT 和 Llama 的向量表示空间进行对齐。BERT 的使用确保了日志消息之间的清晰边界，因为每个消息都由一个独特的嵌入向量表示，从而提高了分类性能。此外，当保持 Llama 的内存和参数大小不变时，这种方法可以处理比直接使用 Llama 的标记器对整个日志序列进行标记更长的序列。

III. 预备知识

为了为后续章节奠定基础，我们介绍了**系统日志**，该日志记录了系统在运行时的事件和内部状态。系统日志包含按时间顺序排列的日志消息列表。

图 1 展示了由 BGL （ BlueGene/L 超级计算机系统）生成的原始系统日志片段，每个日志消息都按记录的时间顺序排列。这些原始日志消息是半结构化文本，由一个**标题**和**内容**组成。标题由日志框架确定，包括时间戳、详细程度（例如，WARN/INFO ）和组件 [47] 等信息。日志内容包含一个固定部分（揭示日志模板的关键词）和一个可变部分（携带动态运行时信息的参数）。在本文中，我们仅关注每个日志消息的内容。

日志消息可以根据会话或固定 / 滑动窗口（**即记录特定执行流程的一系列日志消息）进行分组**。[48] 会话窗口分区根据日志消息的会话 ID 将日志消息分组，从而生成包含日志的消息序列。

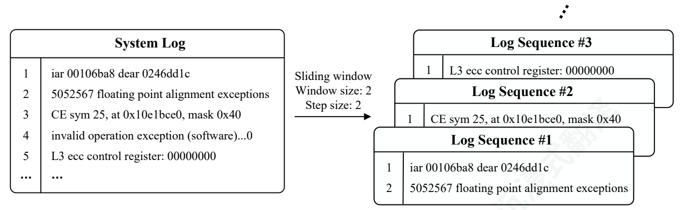


```
System Log
1   Receiving block blk_-5632101276183739500
    src: /10.251.27.63:36433 dest: /
    10.251.27.63:50010
2   BLOCK* NameSystem.allocateBlock: /user/
    root/randtxt4/_temporary/task_200811101024/
    part-00999. blk_-5632101276183739500
3   Receiving block blk_-4506306395053060141
    src: /10.251.193.175:40709 dest: /
    10.251.193.175:50010
4   PacketResponder 0 for block blk_-
    5632101276183739500 terminating
... ...
```

```
Log Sequence #2
(For blk_-4506306395053060141)
1   Receiving block blk_-4506306395053060141
    src: /10.251.193.175:40709 dest: /

Log Sequence #1
(For blk_-5632101276183739500)
1   Receiving block blk_-5632101276183739500
    src: /10.251.27.63:36433 dest: /
    10.251.27.63:50010
2   BLOCK* NameSystem.allocateBlock: /user/
    root/randtxt4/_temporary/task_200811101024/
    part-00999. blk_-5632101276183739500
3   PacketResponder 0 for block blk_-
    5632101276183739500 terminating
... ...
```

(a) Session window



```
System Log
1   iar 00106ba8 dear 0246dd1c
2   5052567 floating point alignment exceptions
3   CE sym 25, at 0x10e1bce0, mask 0x40
4   invalid operation exception (software)...0
5   L3 ecc control register: 00000000
... ...
```

```
Log Sequence #3
1   L3 ecc control register: 00000000

Log Sequence #2
1   CE sym 25, at 0x10e1bce0, mask 0x40

Log Sequence #1
1   iar 00106ba8 dear 0246dd1c
2   5052567 floating point alignment exceptions
```

(b) Sliding window

Fig. 2: Illustrative examples of log message partitioning.

messages within each session. For example, Fig. 2a illustrates the HDFS [49] logs undergoing the session window grouping process, where the *block_id* serves as the session ID. In contrast, **fixed/sliding window** partitioning groups log messages based on a fixed size (window size), which can be defined by either the time span or the number of log messages. This method creates sequences that capture snapshots of system log messages over time. For example, Fig. 2b illustrates the BGL [50] logs undergoing the sliding window grouping process, with a window size of 2 messages and a step size of 2 messages.

The objective of log-based anomaly detection is to identify anomalous log sequences, facilitating the recognition of potential issues within the system's operational behavior.

IV. Methodology

In this section, we present our innovative anomaly detection framework, LogLLM. As illustrated in Fig. 3, the log sequence undergoes preprocessing using regular expressions before being fed into a deep neural network that integrates BERT [43], a projector, and Llama [26] for log sequence classification. In the following sections, we will provide detailed insights into log sequence preprocessing, the architecture of the deep model, and the model training procedure.

A. Preprocessing

Considering that the log message content includes variable parameters carrying dynamic runtime information, which is always irrelevant to the anomalies and complicates deep model training, as demonstrated in Section V-F, a technique is needed to identify these parameters and replace them with a constant token. Log parsers, such as Drain [51] and Spell [52], are widely adopted in log-based anomaly detection methods and appear to be a useful technique. However, as noted by Le et al. [3], existing log parsers do not always perform correctly on all log datasets and struggle to handle out-of-vocabulary (OOV) words in new log messages, resulting in a loss of semantic information. When logs are unstable, these parsers become increasingly ineffective over time, making it difficult to support subsequent anomaly detection.

Thanks to the structured log generation process, the textual format of parameters representing specific objects can be easily identified using regular expressions [53]. Consequently, we replace each variable parameter, such as account, directory path, and IP address, with '<*>'. Despite its simplicity, this technique offers significant performance advantages. Compared with log parsers, this preprocessing technique is more effective and does not require training.

B. Model Architecture

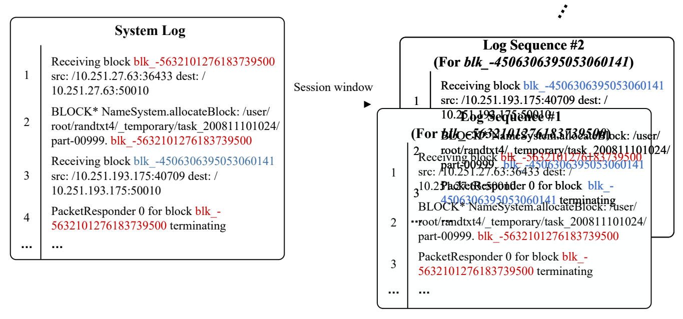
As shown in Fig. 3, our deep model consists of three main components: BERT, a projector, and Llama. Both BERT and Llama are pretrained LLMs. BERT is utilized to extract vector representations of log messages, while Llama is employed to classify the log sequences. The projector serves as a bridge, aligning the vector representation spaces of BERT and Llama. It is important to note that our model incorporates only one instance of BERT and one projector.

1) BERT: BERT generates a semantic vector by processing the semantic vector of the classification token ([CLS]) through a linear layer followed by a *tanh* activation function. Each log message, once preprocessed, is encoded into a semantic vector using the BERT tokenizer and BERT model. For a preprocessed log sequence, the output of BERT is a sequence of semantic vectors $C = (c_1, c_2, \ldots, c_N) \in \mathbb{R}^{N \times d_{BERT}}$, where N represents the length of the log sequence (i.e., the number of log messages) and d_{BERT} is the dimension of each semantic vector (i.e., hidden size).

2) Projector: The projector is a linear layer that maps the semantic vectors $C \in \mathbb{R}^{N \times d_{BERT}}$ to the token embedding vectors accepted by Llama, represented as $E = (e_1, e_2, \ldots, e_N) \in \mathbb{R}^{N \times d_{Llama}}$, where d_{Llama} is the hidden size of Llama. The projector is designed to align the vector representation spaces of BERT and Llama.

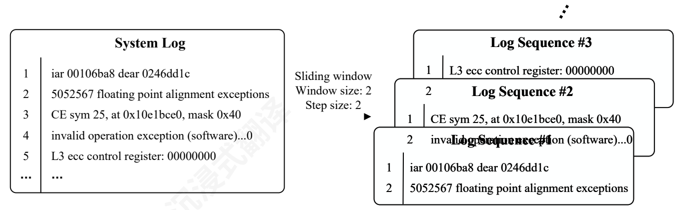
3) Llama: To conduct prompt tuning on Llama, the transformer decoder-based LLM, we generate corresponding textual queries based on embedded log sequences. Specifically, each query consists of three components.

The first component introduces the log sequence, such as "*Below is a sequence of system log messages:*". The second component comprises the token embeddings E output by the projector. The third component queries whether the sequence is anomalous, asking, for instance, ". *Is this sequence normal or anomalous?*". The first and third components are fed into the Llama tokenizer and Llama embedding layer sequentially, producing $E_1 \in \mathbb{R}^{A \times d_{Llama}}$ and $E_3 \in \mathbb{R}^{Q \times d_{Llama}}$, where A and Q are the number of tokens produced by tokenizing the first and third components, respectively. Then, the token embeddings of the three components are concatenated, rep-



```
System Log
1   Receiving block blk_-5632101276183739500
    src: /10.251.27.63:36433 dest: /
    10.251.27.63:50010
2   BLOCK* NameSystem.allocateBlock: /user/
    root/randtxt4/_temporary/task_200811101024/
    part-00999. blk_-5632101276183739500
3   Receiving block blk_-4506306395053060141
    src: /10.251.193.175:40709 dest: /
    10.251.193.175:50010
4   PacketResponder 0 for block blk_-
    5632101276183739500 terminating
... ...
```

(a) Session window



```
System Log
1   iar 00106ba8 dear 0246dd1c
2   5052567 floating point alignment exceptions
3   CE sym 25, at 0x10e1bce0, mask 0x40
4   invalid operation exception (software)...0
5   L3 ecc control register: 00000000
... ...

Log Sequence #3
1   L3 ecc control register: 00000000

Log Sequence #2

Log Sequence #1
1   iar 00106ba8 dear 0246dd1c
2   5052567 floating point alignment exceptions
```

(b) Sliding window

图 2：日志消息分区示例的说明。

每个会话中的消息。例如，图 2a 展示了 HDFS [49] 日志在会话窗口分组过程中的情况，其中 *block_id* 作为会话 ID。相比之下，**xed/sliding window** partitioning 根据固定大小（窗口大小）对日志消息进行分组，该大小可以通过时间跨度或日志消息数量来定义。这种方法创建了捕获系统日志消息随时间快照的序列。例如，图 2b 展示了 BGL[50] 日志在滑动窗口分组过程中的情况，窗口大小为 2 条消息，步长为 2 条消息。

基于日志的异常检测的目的是识别异常日志序列，以便识别系统操作行为中可能存在的问题。

IV. 方法论

在本节中，我们介绍了我们创新的异常检测框架 LogLLM。如图 3 所示，日志序列在输入到深度神经网络之前，使用正则表达式进行预处理，该神经网络集成了 BERT [43],a 投影器和 Llama [26] 用于日志序列分类。在接下来的章节中，我们将详细介绍日志序列预处理、深度模型架构以及模型训练过程。

A. 预处理

考虑到日志消息内容包含携带动态运行时信息的变量参数，这些参数始终与异常无关，并使深度模型训练复杂化，如第 V-F 节所示，需要一种技术来识别这些参数并将它们替换为常量标记。日志解析器，如 Drain [51] 和 Spell [52], ，在基于日志的异常检测方法中得到广泛应用，并似乎是一种有用的技术。然而，正如 Le 等人所指出的。[3], 现有的日志解析器并不总是对所有日志数据集都表现正确，并且难以处理新日志消息中的词汇表外（OOV）单词，导致语义信息丢失。当日志不稳定时，这些解析器随着时间的推移变得越来越无效，使得支持后续异常检测变得困难。

得益于结构化日志生成过程，可以使用正则表达式 [53] 轻松地识别表示特定对象的参数的文本格式。因此，我们将每个变量参数，如账户、目录路径和 IP 地址，替换为 ‘<*>’。尽管这种方法很简单，但它提供了显著的性能优势。与日志解析器相比，这种预处理技术更有效，且不需要训练。

B. 模型架构

如图 3 所示，我们的深度模型由三个主要组件组成：BERT 、投影器和 Llama。BERT 和 Llama 都是预训练的大型语言模型。BERT 用于提取日志消息的向量表示，而 Llama 用于分类日志序列。投影器作为桥梁，将 BERT 和 Llama 的向量表示空间对齐。需要注意的是，我们的模型只包含一个 BERT 实例和一个投影器。

1) BERT：BERT 通过一个线性层后跟一个 *tanh* 激活函数处理分类标记（[CLS]）的语义向量来生成一个语义向量。每个预处理后的日志消息，使用 BERT 标记器和 BERT 模型编码成一个语义向量。对于预处理后的日志序列，BERT 的输出是一个语义向量序列 $C = (c_1, c_2, \ldots, c_N) \in \mathbb{R}^{N \times d_{BERT}}$，其中 N 代表日志序列的长度（即日志消息的数量），d_{BERT} 是每个语义向量的维度（即隐藏大小）。

2) 投影仪：投影仪是一个线性层，它将语义向量 $C \in \mathbb{R}^{N \times d_{BERT}}$ 映射到 Llama 接受的标记嵌入向量，表示为 $E = (e_1, e_2, \ldots, e_N) \in \mathbb{R}^{N \times d_{Llama}}$，其中 d_{Llama} 是 Llama 的隐藏大小。投影仪的设计是为了使 BERT 和 Llama 的向量表示空间对齐。

3) Llama：为了对基于 Transformer 解码器的 LLM Llama 进行提示微调，我们根据嵌入的日志序列生成相应的文本查询。具体来说，每个查询由三个部分组成。

第一个组件引入了日志序列，例如 “ 以下是系统日志消息的序列：”。第二个组件包括投影仪输出的标记嵌入 E 。第三个组件查询序列是否异常，例如询问 “。这个序列是正常还是异常？”。第一个和第三个组件依次输入到 Llama tokenizer 和 Llama embedding layer，产生 $E_1 \in \mathbb{R}^{A \times d_{Llama}}$ 和 $E_3 \in \mathbb{R}^{Q \times d_{Llama}}$，其中 A 和 Q 分别是第一个和第三个组件标记化后产生的标记数量。然后，将三个组件的标记嵌入连接起来，

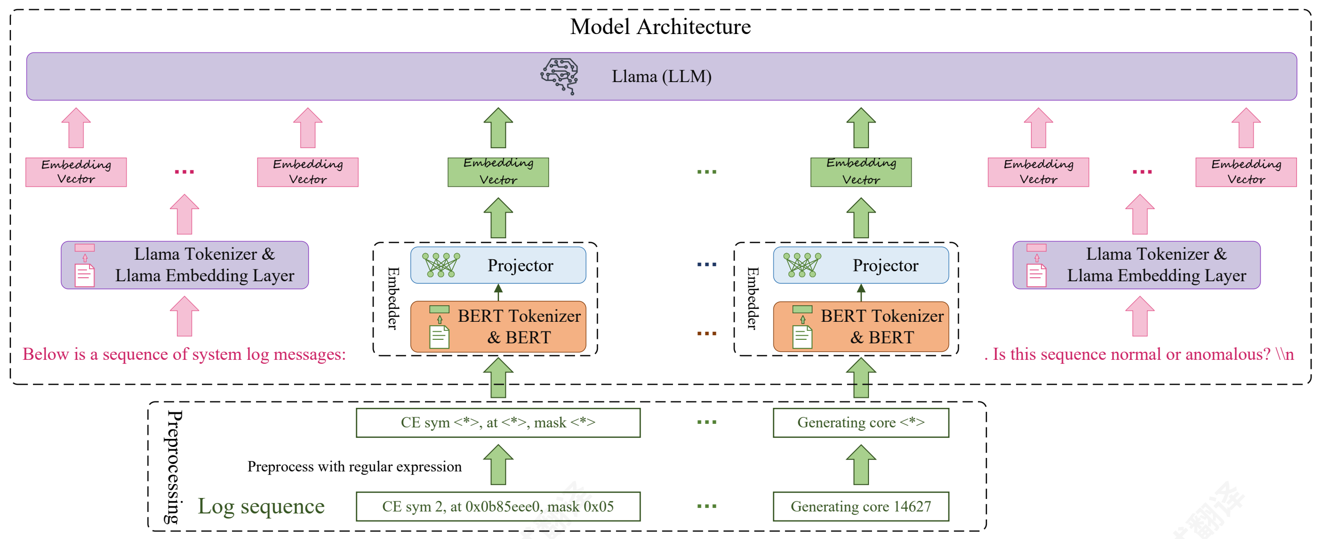

Model Architecture
Llama (LLM)
Embedding Vector
Llama Tokenizer & Llama Embedding Layer
Projector
BERT Tokenizer & BERT
Embedder
Below is a sequence of system log messages:
. Is this sequence normal or anomalous? \\n
Preprocessing
CE sym <*>, at <*>, mask <*>
Generating core <*>
Preprocess with regular expression
Log sequence
CE sym 2, at 0x0b85eee0, mask 0x05
Generating core 14627

Fig. 3: The framework of LogLLM. Notably, the model includes a single instance of BERT and the projector.

resented as $[E_1||E||E_3] \in \mathbb{R}^{(A+N+Q)\times d_{Llama}}$ and fed into Llama.

C. Training

1) Minority Class Oversampling: LogLLM is a supervised anomaly detection method, which means it needs labeled normal and anomalous samples for training. However, supervised anomaly detection methods often face the challenge of data imbalance, which can lead to biased model training. In an anomaly detection task, there are only two classes: normal and anomalous, and the number of instances in each class is uncertain. To cope with data imbalance, we oversample the class with fewer samples, ensuring that the proportion of the minority class is no less than β. Formally, let the proportion of the minority class be α and $\alpha < \beta$, and the total number of samples be $Sample\_num$. To achieve a proportion of β for the minority class, it will be oversampled to the following quantity:

$$\frac{\beta(1-\alpha)}{1-\beta} \times Sample\_num \tag{1}$$

This adjustment will make the proportion of the minority class equal to β.

2) Training Objective: Our objective is to train the deep model to predict whether a given log sequence is normal or anomalous. We fine-tune the model to respond appropriately: if the sequence is anomalous, it outputs '*The sequence is anomalous.*'; if normal, it outputs '*The sequence is normal.*'. We utilize cross-entropy loss [54] as our loss function.

3) Training Procedure: To train our deep model, we follow three main stages.

Stage 1. Fine-tuning Llama to capture the answer template: The first stage involves fine-tuning Llama to capture the answer template. Specifically, we train Llama to respond to the prompt '*Is this sequence normal or anomalous?*' with '*The sequence is anomalous/normal.*'. This stage requires only a few data samples.

Stage 2. Training the embedder of log messages: The second stage involves training the embedder of log messages, specifically BERT and the projector. This stage aims to project each log message to the embedding of the most suitable token in Llama, enabling Llama to discern whether the given log sequence is normal or anomalous.

Stage 3. Fine-tuning the entire model: Finally, we fine-tune the entire model to ensure cohesive and accurate performance across all components.

4) Efficient Fine-Tuning on LLMs: To reduce the costs involved in fine-tuning LLMs (BERT and Llama) with a substantial number of parameters, we utilize QLoRA [55] to minimize memory usage. QLoRA accomplishes this by backpropagating gradients into a frozen 4-bit quantized model, while maintaining the performance levels achieved during the full 16-bit fine-tuning process.

V. Experiments

In this section, we conduct extensive experiments on four real-life logs to investigate the following research questions (RQs):

- **RQ1**: How effective is LogLLM in log-based anomaly detection?
- **RQ2**: How do different preprocessing techniques impact the performance of LogLLM?
- **RQ3**: How effective is the embedder for Llama?
- **RQ4**: How does the size of the Llama model affect the performance of LogLLM?
- **RQ5**: How does each stage of the three-stage training process influence the performance of LogLLM?
- **RQ6**: How do different levels of minority class oversampling, determined by the hyperparameter β, affect the performance of LogLLM?

LogLLM is coded in Python, and the source code is available at https://github.com/guanwei49/LogLLM.

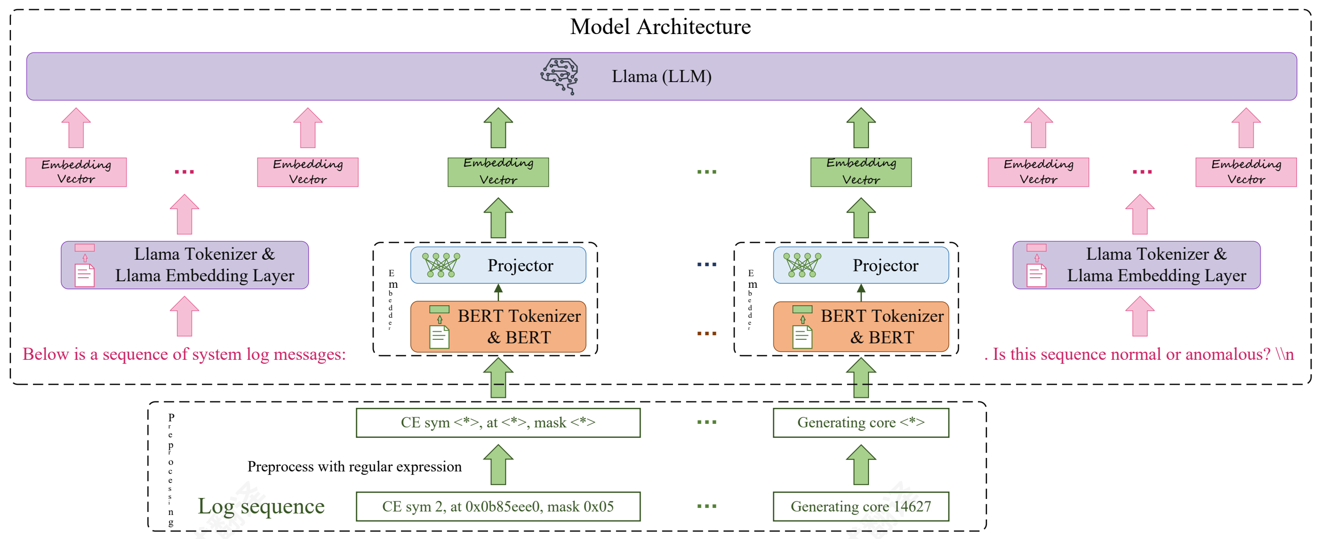

Model Architecture
Llama (LLM)
Embedding Vector
Llama Tokenizer & Llama Embedding Layer
Projector
BERT Tokenizer & BERT
Below is a sequence of system log messages:
. Is this sequence normal or anomalous? \\n
CE sym <*>, at <*>, mask <*>
Generating core <*>
Preprocess with regular expression
Log sequence
CE sym 2, at 0x0b85eee0, mask 0x05
Generating core 14627

图 3：LogLLM 的框架。值得注意的是，该模型包含一个 BERT 实例和投影 tor.

被重命名为 $[E_1||E||E_3] \in \mathbb{R}^{(A+N+Q)\times d_{Llama}}$ 并输入到 Llama 中。

C. 训练

1) 少数类过采样：LogLLM 是一种监督式异常检测方法，这意味着它需要标记的正常和异常样本进行训练。然而，监督式异常检测方法通常面临数据不平衡的挑战，这可能导致模型训练偏差。在异常检测任务中，只有两个类别：正常和异常，每个类别的实例数量是不确定的。为了应对数据不平衡，我们对样本数量较少的类别进行过采样，确保少数类别的比例不少于 β。形式上，设少数类别的比例为 α 和 $\alpha < \beta$，样本总数为 $Sample\_num$。为了使少数类别的比例为 β，需要过采样到以下数量：

$$\frac{\beta(1-\alpha)}{1-\beta} \times Sample\_num \tag{1}$$

这种调整将使少数类别的比例等于 β。

2) 训练目标：我们的目标是训练深度模型以预测给定的日志序列是正常还是异常。我们微调模型以做出适当的响应：如果序列是异常的，它输出 ‘ 该序列是异常的。’ ；如果是正常的，它输出 ‘ 该序列是正常的。’ 。我们使用交叉熵损失 [54] 作为我们的损失函数。

3) 训练流程：为了训练我们的深度模型，我们遵循三个主要阶段。

第一阶段：微调 Llama 以捕捉答案模板：第一阶段涉及微调 Llama 以捕捉答案模板。具体来说，我们训练 Llama 对提示 “ 这个序列正常还是异常？ ” 做出 “ 这个序列是异常 / 正常的。 ” 的回应。这一阶段只需要少量数据样本。

阶段 2：训练日志消息的嵌入器：第二阶段涉及训练日志消息的嵌入器，特别是 BERT 和投影器。这一阶段的目的是将每个日志消息投影到 Llama 中最合适的标记的嵌入中，使 Llama 能够判断给定的日志序列是正常还是异常。

第 3 阶段。微调整个模型：最后，我们将整个模型进行微调，以确保所有组件的性能一致且准确。

4) 在 *LLMs* 上的高效微调：为了降低使用大量参数对 LLMs （ BERT 和 Llama ）进行微调的成本，我们利用 QLoRA [55] 来最小化内存使用。QLoRA 通过将梯度反向传播到冻结的 4 位量化模型中，同时保持在全 16 位微调过程中达到的性能水平。

V. 实验

在本节中，我们对四个真实日志进行了广泛的实验，以研究以下研究问题（ RQs ）：

RQ1：LogLLM 在基于日志的异常检测中效果如何?

RQ2：不同的预处理技术如何影响 LogLLM 的性能?

RQ3：Llama 的嵌入器效果如何?
RQ4：Llama 模型的大小如何影响 LogLLM 的性能?

RQ5：三个阶段训练过程中的每个阶段如何影响 LogLLM 的性能?

RQ6：由超参数 β 确定的少数类过采样不同级别如何影响 LogLLM 的性能?

LogLLM 是用 Python 编写的，源代码可在 https://github.com/guanwei49/LogLLM 上找到。

A. Benchmark Methods

To verify the superiority of the proposed method, we compare LogLLM with five state-of-the-art semi-supervised methods: DeepLog [8], LogAnomaly [9], PLELog [22], FastLogAD [34], and LogBERT [32]. We also compare it with three supervised methods: LogRobust [19], CNN [18] and NeuralLog [3], and one method that does not require training a deep model but needs some normal samples for retrieval: RAPID [39].

Notably, FastLogAD, LogBERT, NeuralLog, and RAPID adopt LLMs for anomaly detection.

B. Experimental Settings

We conduct all experiments on a server equipped with an Intel Xeon Gold 6330 CPU (38 cores), 256GB of memory, and an NVIDIA A40 GPU with 48 GB of memory.

In our experiment, we utilize the BERT-base model[1] and Llama-3-8B model[2] as backbones. The hyperparameter β, which is described in Section IV-C1, is set to 30%. We use the AdamW optimizer [56] to train the model with a mini-batch size of 16. Unless otherwise specified, the training procedure is configured as follows: In the first stage, only 1,000 samples are involved with a learning rate of 5e-4. The second and third stages each consist of two epochs with a learning rate of 5e-5.

For a fair comparison, we configure the hyperparameters for all compared methods according to the values provided in their original articles.

C. Metrics

We evaluate the performance of these methods using the widely adopted $Precision$, $Recall$ and $F_1-score$. These metrics are calculated as follows:

$$Precision = \frac{TP}{TP+FP} \tag{2}$$

$$Recall = \frac{TP}{TP+FN} \tag{3}$$

$$F_1-score = \frac{2*Precision*Recall}{Precision+Recall} \tag{4}$$

, where TP, FN, FP represent true positives, false negatives and false positives respectively.

Precision refers to the percentage of correctly detected anomalies among all anomalies identified by the model, while recall represents the percentage of anomalies that are correctly identified from all real anomalies. The F_1-score combines these two metrics into a single measure, providing a balanced assessment of the model's performance in detecting anomalies.

D. Dataset

To evaluate our method for log-based anomaly detection, we selected four public datasets [57]: HDFS, BGL, Liberty, and Thunderbird. The details for each dataset are provided below:

HDFS (Hadoop Distributed File System) dataset [49] is generated by running Hadoop-based mapreduce jobs on over 200 Amazon EC2 nodes and contains a total of 11,175,629 log messages. These log messages are grouped into different log windows based on their *block_id*, which reflect program executions in the HDFS, resulting in 575,061 blocks. Among these, 16,838 blocks (2.93%) indicate system anomalies.

BGL (Blue Gene/L) dataset [50] is a supercomputing system log dataset collected from a BlueGene/L supercomputer system at lawrence livermore national labs (LLNL). The dataset contains 4,747,963 log messages, each of which has been manually labeled as either normal or anomalous. There are 348,460 log messages (7.34%) that are labeled as anomalous.

Thunderbird dataset [50] is a publicly accessible collection of log data sourced from the Thunderbird supercomputer at sandia national laboratories (SNL). This dataset consists of both normal and anomalous messages, each of which has been manually categorized. Although the dataset contains over 200 million log messages, we focus on a subset of 10 million continuous log messages for computational efficiency. This subset includes 4,937 anomalous log messages, representing approximately 0.049% of the total.

Liberty dataset [50] comprises system logs from the Liberty supercomputer at sandia national labs (SNL) in Albuquerque. This supercomputer features 512 processors and 944 GB of memory, and the dataset contains over 200 million log messages. For computational efficiency, we sample 5 million consecutive log messages, among which 1,600,525 are identified as anomalous, constituting approximately 32.01% of the total sampled messages.

In the context of HDFS, we adopt a session window strategy, which involves grouping log messages into sequences based on the *block_id* present in each log message. Each session is labeled using ground truth. For other datasets, including BGL, Thunderbird, and Liberty, we utilize a sliding window strategy to group log messages, with a window size of 100 messages and a step size of 100 messages. A log sequence is deemed anomalous if it contains at least one anomalous log message according to the ground truth.

Similar to existing work [8], [9], [19], [22], [34], [39], we split each dataset into a training set and a testing set with a ratio of 8:2 to evaluate the performance of a log-based anomaly detection approach. For the HDFS dataset, we randomly split the log sequences into training and testing data. In contrast, for the BGL, Thunderbird, and Liberty datasets, we adhere to a chronological split [6]. This strategy ensures that all log sequences in the training set precede those in the testing set, reflecting real-world conditions and mitigating potential data leakage from unstable log data.

Table I summarizes the statistics of the datasets used in the experiments.

E. Performance Evaluation (RQ1)

Table II presents the experimental results of various log-based anomaly detection methods on the HDFS, BGL, Liberty, and Thunderbird datasets. The best results are highlighted in bold. We have the following observations:

[1]https://huggingface.co/google-bert/bert-base-uncased
[2]https://huggingface.co/meta-llama/Meta-Llama-3-8B

A. 基准方法

为了验证所提方法的优越性，我们将 LogLLM 与五种最先进的半监督方法进行了比较：DeepLog [8], LogAnomaly [9], PLELog [22], Fast-LogAD [34], 和 LogBERT [32]。我们还将其与三种监督方法进行了比较：LogRobust [19],CNN [18] 和 NeuralLog [3], ，以及一种不需要训练深度模型但需要一些正常样本用于检索的方法：RAPID [39]。

值得注意的是，FastLogAD 、LogBERT 、 NeuralLog 和 RAPID 采用了 LLM 进行异常检测。

B. 实验设置

我们在配备 Intel Xeon Gold 6330 CPU （38 核）、256GB 内存和 NVIDIA A40 GPU （48GB 内存）的服务器上进行了所有实验。

在我们的实验中，我们利用 BERT-base 模型 [1] 和 Llama-3-8B 模型 [2] 作为骨干。第 IV-C1 节中描述的超参数 β 设置为 30%。我们使用 AdamW 优化器 [56] 以 16 个样本的小批量大小训练模型。除非另有说明，否则训练过程配置如下：在第一阶段，只涉及 1,000 个样本，学习率为 5e-4。第二和第三阶段各包含两个 epoch，学习率为 5e-5。

为了进行公平的比较，我们将所有比较方法的超参数配置为它们原始文章中提供的值。

C. 评估指标

我们使用广泛采用的 $Precision$、 $Recall$ 和 $F_1-score$ 来评估这些方法的性能。这些指标的计算方法如下：

$$Precision = \frac{TP}{TP+FP} \tag{2}$$

$$Recall = \frac{TP}{TP+FN} \tag{3}$$

$$F_1-score = \frac{2*Precision*Recall}{Precision+Recall} \tag{4}$$

, where TP, FN, FP represent true positives, false negatives and false positives respectively.

精确率指的是模型识别出的所有异常中正确检测到的异常所占的百分比，而召回率表示从所有真实异常中正确识别出的异常所占的百分比。F₁- 分数将这两个指标结合成一个单一指标，为模型在检测异常方面的性能提供平衡评估。

D. 数据集

为了评估我们的基于日志的异常检测方法，我们选择了四个公开数据集 [57]：HDFS 、BGL 、Liberty 和 Thunderbird。每个数据集的详细信息如下：

HDFS （Hadoop 分布式文件系统）数据集是通过在超过 200 个 Amazon EC2 节点上运行基于 Hadoop 的 mapreduce 作业生成的。这些日志消息根据它们的块 _id 分组，这些 id 反映了 HDFS 中的程序执行，从而产生了 575,061 个块。在这些块中，有 16,838 个块（2.93%）表示系统异常。

BGL （Blue Gene/L）数据集是从劳伦斯利弗莫尔国家实验室（LLNL）的 BlueGene/L 超级计算机系统收集的超级计算机系统日志数据集。该数据集包含 4,747,963 条日志消息，每条消息都已被人工标记为正常或异常。其中有 348,460 条日志消息（7.34%）被标记为异常。

Thunderbird 数据集是从桑迪亚国家实验室（SNL）的 Thunderbird 超级计算机获取的公开可访问的日志数据集。该数据集包含正常和异常消息，每条消息都已被人工分类。尽管该数据集包含超过 2 亿条日志消息，但我们关注的是 1000 万条连续日志消息的子集，以提高计算效率。这个子集包括 4,937 条异常日志消息，占总数的约 0.049%。

Liberty 数据集由位于阿尔伯克基的桑迪亚国家实验室（SNL）的 Liberty 超级计算机的系统日志组成。这台超级计算机具有 512 个处理器和 944 GB 的内存，数据集包含超过 2 亿条日志消息。为了提高计算效率，我们采样了 500 万条连续的日志消息，其中 1,600,525 条被识别为异常，占总采样消息的约 32.01%。

在 HDFS 的背景下，我们采用会话窗口策略，该策略涉及根据每个日志消息中存在的块 _id 将日志消息分组为序列。每个会话都使用真实标签进行标记。对于其他数据集，包括 BGL 、Thunderbird 和 Liberty，我们使用滑动窗口策略来分组日志消息，窗口大小为 100 条消息，步长为 100 条消息。如果一个日志序列包含至少一条根据真实标签标记为异常的日志消息，则认为该日志序列是异常的。

与现有工作 [8], [9], [19], [22], [34], [39]，类似，我们将每个数据集分为训练集和测试集，比例为 8:2，以评估基于日志的异常检测方法的性能。对于 HDFS 数据集，我们随机将日志序列分为训练数据和测试数据。相比之下，对于 BGL 、Thunderbird 和 Liberty 数据集，我们遵循时间顺序分割 [6]。这种策略确保训练集中的所有日志序列都早于测试集中的日志序列，反映了现实世界的情况，并减轻了不稳定日志数据可能引起的数据泄露。

表 I 总结了实验中使用的各个数据集的统计数据。

E. 性能评估（*RQ1*）

表 II 展示了各种基于日志的异常检测方法在 HDFS 、BGL 、Liberty 和 Thunderbird 数据集上的实验结果。最佳结果以粗体显示。我们观察到以下几点：

[1]https://huggingface.co/google-bert/bert-base-uncased
[2]https://huggingface.co/meta-llama/Meta-Llama-3-8B

TABLE I: The statistics of datasets used in the experiments.

	# Log messages	# Log sequences	Training Data			Testing Data		
			# Log sequences	# Anomalies	Anomaly ratio	# Log sequences	# Anomalies	Anomaly ratio
HDFS	11,175,629	575,061	460,048	13,497	2.93%	115,013	3,341	2.90%
BGL	4,747,963	47,135	37,708	4,009	10.63%	9,427	817	8.67%
Liberty	5,000,000	50,000	40,000	34,144	85.36%	10,000	651	6.51%
Thunderbird	10,000,000	99,997	79,997	837	1.05%	20,000	29	0.15%

TABLE II: Experimental results on HDFS, BGL, Liberty, and Thunderbird datasets. The best results are highlighted in bold.

Datasets / Methods	Log parser	HDFS			BGL			Liberty			Thunderbird			Avg. F_1
		Prec.	Rec.	F_1	Prec.	Rec.	F_1	Prec.	Rec.	F_1	Prec.	Rec.	F_1	
DeepLog	✔	0.835	0.994	0.908	0.166	0.988	0.285	0.751	0.855	0.800	0.017	0.966	0.033	0.506
LogAnomaly	✔	0.886	0.893	0.966	0.176	0.985	0.299	0.684	0.876	0.768	0.025	0.966	0.050	0.521
PLELog	✔	0.893	0.979	0.934	0.595	0.880	0.710	0.795	0.874	0.832	0.808	0.724	0.764	0.810
FastLogAD	✔	0.721	0.893	0.798	0.167	**1.000**	0.287	0.151	**0.999**	0.263	0.008	0.931	0.017	0.341
LogBERT	✔	0.989	0.614	0.758	0.165	0.989	0.283	0.902	0.633	0.744	0.022	0.172	0.039	0.456
LogRobust	✔	0.961	1.000	0.980	0.696	0.968	0.810	0.695	0.979	0.813	0.318	**1.000**	0.482	0.771
CNN	✔	0.966	1.000	0.982	0.698	0.965	0.810	0.580	0.914	0.709	0.870	0.690	0.769	0.818
NeuralLog	✗	0.971	0.988	0.979	0.792	0.884	0.835	0.875	0.926	0.900	0.794	0.931	0.857	0.893
RAPID	✗	**1.000**	0.859	0.924	**0.874**	0.399	0.548	0.911	0.611	0.732	0.200	0.207	0.203	0.602
LogLLM	✗	0.994	**1.000**	**0.997**	0.861	0.979	**0.916**	**0.992**	0.926	**0.958**	**0.966**	0.966	**0.966**	**0.959**

The proposed LogLLM achieves the highest F_1-score across all datasets. On average, LogLLM's F_1-scores are 6.6% better than the best existing method, NeuralLog, demonstrating its effectiveness in log-based anomaly detection. Despite the adoption of LLMs in FastLogAD, LogBERT, NeuralLog, and RAPID for anomaly detection, their performance remains unsatisfactory. FastLogAD and LogBERT utilize BERT, a transformer encoder-based model, for detecting anomalies based on log sequence reconstruction errors. Their inputs consist of sequences of log template IDs (IDs of log string templates) extracted from log messages via log parsers, lacking semantic information. In contrast, NeuralLog and RAPID utilize transformer encoder-based models to extract semantic vectors from log messages. However, NeuralLog utilizes smaller models, whereas RAPID relies on distance-based comparison for anomaly sequence classification. LogLLM, on the other hand, leverages both BERT for extracting semantic vectors and Llama, a transformer decoder-based LLM, for anomaly detection. The representation spaces of BERT and Llama are aligned via a projector, fully harnessing the potential of LLMs for log-based anomaly detection.

Moreover, LogLLM achieves a balance between precision and recall, indicating that it maintains low false alarm rates and minimizes missed reports. In contrast, methods like FastLogAD are excessively sensitive to anomalies, often resulting in numerous false alarms. For example, on the BGL dataset, despite FastLogAD having a recall of 1, it only achieves a precision of 0.167, making it impractical for real-world use. Similarly, methods such as DeepLog, LogAnomaly and LogBERT exhibit similar issues. On the other hand, RAPID is not sensitive enough to anomalies, leading to many undetected anomalies. For instance, on the BGL dataset, RAPID achieves a precision of 0.874 but a recall of only 0.399.

Effect of labeled anomalies: As illustrated in Table II, in contrast to methods such as DeepLog, LogAnomaly, FastLogAD, LogBERT, and RAPID, which require clean datasets devoid of anomalies to build anomaly detection models, methods like PLELog, LogRobust, CNN, NeuralLog, and LogLLM demonstrate superior performance. These models are trained using not only normal samples but also labeled anomalies. For instance, these five methods achieve an average F_1-score above 0.771 across four datasets, whereas others that do not utilize labeled anomalies perform poorly, with an average F_1-score below 0.602 across four datasets. This demonstrates that incorporating labeled anomalies can provide a significant advantage to anomaly detection methods.

TABLE III: Computational cost.

	Training time (Minutes)	Testing time (Minutes)
DeepLog	72.17	3.42
LogAnomaly	156.16	7.25
PLELog	315.47	33.59
LogRobust	108.42	2.48
CNN	98.16	2.16
FastLogAD	254.17	0.29
LogBERT	429.04	43.77
NeuralLog	267.46	21.44
RAPID	63.98	38.43
LogLLM	1,065.15	64.48

Computational cost: The time consumption of each method is presented in Table III. These results have been averaged across all the datasets.

Although RAPID does not require training a deep model, the extraction and retrieval of vector representations remain time-consuming. In comparison to other methods, FastLogAD requires relatively high training time, but it has the shortest testing time because it uses only the discriminator of the model during testing. As anticipated, while our proposed LogLLM demonstrates the best performance, it also incurs the highest

表 I：实验中使用的数据集的统计信息。

	日志消息	日志序列	训练数据			测试数据		
			# Log sequences	异常	异常比率	# Log sequences	# 异常	Anomaly ratio
HDFS	11,175,629	575,061	460,048	13,497	2.93%	115,013	3,341	2.90%
BGL	4747963	47135	37708	4009	10.63%	9427	817	8.67%
自由	500,000	50,000	40,000	34,144	85.36%	10,000	651	6.51%
Thunderbird	10,000,000	99,997	79,997	837	1.05%	20,000	29	0.15%

T表 2：在 HDFS 、 BGL 、 Liberty 和 Thunderbird 数据集上的实验结果。最佳结果已突出显示 n bold.

数据集 / 方法	Log parser	HDFS			BGL			自由			Thunderbird			Avg. F_1
		精度	Rec.	F_1	精度	Rec.	F_1	精度	Rec.	F_1	精度	Rec.	F_1	
DeepLog	✔	0.835	0.994	0.908	0.166	0.988	0.285	0.751	0.855	0.800	0.017	0.966	0.033	0.506
日志异常	✔	0.886	0.893	0.966	0.176	0.985	0.299	0.684	0.876	0.768	0.025	0.966	0.050	0.521
PLELog	✔	0.893	0.979	0.934	0.595	0.880	0.710	0.795	0.874	0.832	0.808	0.724	0.764	0.810
FastLogAD	✔	0.721	0.893	0.798	0.167	**1.000**	0.287	0.151	**0.999**	0.263	0.008	0.931	0.017	0.341
LogBERT	✔	0.989	0.614	0.758	0.165	0.989	0.283	0.902	0.633	0.744	0.022	0.172	0.039	0.456
LogRobust	✔	0.961	1.000	0.980	0.696	0.968	0.810	0.695	0.979	0.813	0.318	**1.000**	0.482	0.771
CNN	✔	0.966	1.000	0.982	0.698	0.965	0.810	0.580	0.914	0.709	0.870	0.690	0.769	0.818
NeuralLog	✗	0.971	0.988	0.979	0.792	0.884	0.835	0.875	0.926	0.900	0.794	0.931	0.857	0.893
RAPID	✗	**1.000**	0.859	0.924	**0.874**	0.399	0.548	0.911	0.611	0.732	0.200	0.207	0.203	0.602
LogLLM	✗	0.994	**1.000**	**0.997**	0.861	0.979	**0.916**	**0.992**	0.926	**0.958**	**0.966**	0.966	**0.966**	**0.959**

所提出的 LogLLM 在所有数据集上实现了最高的 F1-分数。平均而言，LogLLM 的 F1- 分数比现有最佳方法 NeuralLog 高出 6.6%，证明了其在基于日志的异常检测中的有效性。尽管 FastLogAD 、 LogBERT 、 NeuralLog 和 RAPID 等方法采用了 LLM 进行异常检测，但它们的性能仍然不尽人意。FastLogAD 和 LogBERT 利用基于 BERT 的 transformer 编码器模型，通过日志序列重建错误来检测异常。它们的输入由日志解析器从日志消息中提取的日志模板 ID （日志字符串模板的 ID ）序列组成，缺乏语义信息。相比之下，NeuralLog 和 RAPID 利用基于 transformer 编码器的模型从日志消息中提取语义向量。然而，NeuralLog 使用较小的模型，而 RAPID 则依赖于基于距离的比较进行异常序列分类。另一方面，LogLLM 利用 BERT 提取语义向量，并利用 Llama （一种基于 transformer 解码器的 LLM ）进行异常检测。通过投影器对齐 BERT 和 Llama 的表示空间，充分利用 LLM 在基于日志的异常检测中的潜力。

此外，LogLLM 在精确度和召回率之间取得了平衡，表明它保持了低误报率并最大限度地减少了漏报。相比之下，像 Fast-LogAD 这样的方法对异常过于敏感，往往会导致许多误报。例如，在 BGL 数据集上，尽管 FastLogAD 的召回率为 1，但其精确率仅为 0.167，使其在实际应用中不切实际。类似地，DeepLog 、 LogAnomaly 和 LogBERT 等方法也存在类似问题。另一方面，RAPID 对异常不够敏感，导致许多异常未被检测到。例如，在 BGL 数据集上，RAPID 的精确率为 0. 874，但召回率仅为 0.399。

标签异常的影响：如图表 II 所示，在与 DeepLog 、 LogAnomaly 、 FastLogAD 、 LogBERT 和 RAPID 等方法相比，这些方法需要无异常的干净数据集来构建异常检测模型，而像 PLELog 、 LogRobust 、 CNN 、 NeuralLog 和 LogLLM 这样的方法则表现出更优越的性能。这些模型不仅使用正常样本进行训练，还使用标记的异常样本进行训练。例如，这五种方法在四个数据集上实现了平均 F1-score 超过 0.771，而那些没有利用标记异常的方法在四个数据集上的平均 F3-score 低于 0.602。这表明，结合标记的异常可以为异常检测方法提供显著的优点。

表 III：计算成本。

	训练时间（分钟）	Testing time (Minutes)
DeepLog	72.17	3.42
LogAnomaly	156.16	7.25
PLELog	315.47	33.59
LogRobust	108.42	2.48
CNN	98.16	2.16
FastLogAD	254.17	0.29
LogBERT	429.04	43.77
NeuralLog	267.46	21.44
RAPID	63.98	38.43
LogLLM	1,065.15	64.48

计算成本：每种方法的耗时在表 III 中给出。这些结果是在所有数据集上平均得出的。

尽管 RAPID 不需要训练深度模型，但向量和表示的提取和检索仍然耗时。与其他方法相比，FastLogAD 需要相对较高的训练时间，但因为它在测试时只使用模型的判别器，所以测试时间最短。正如预期的那样，虽然我们提出的 LogLLM 表现出最佳性能，但它也产生了最高的

TABLE IV: Effects of different preprocessing techniques on HDFS, BGL, Liberty, and Thunderbird datasets. The best results are highlighted in bold.

	HDFS			BGL			Liberty			Thunderbird			Avg. F_1
	Prec.	Rec.	F_1	Prec.	Rec.	F_1	Prec.	Rec.	F_1	Prec.	Rec.	F_1	
Raw	0.994	0.991	0.993	**0.943**	0.767	0.846	0.911	0.908	0.909	0.806	0.862	0.833	0.895
Template ID	**0.995**	0.945	0.969	0.775	0.286	0.418	**0.994**	0.270	0.425	**1.000**	0.379	0.550	0.591
Template	0.991	1.000	0.995	0.861	0.919	0.889	0.968	**0.931**	0.949	0.950	0.655	0.776	0.902
RE (LogLLM)	0.994	**1.000**	**0.997**	0.861	**0.979**	**0.916**	0.992	0.926	**0.958**	0.966	**0.966**	**0.966**	**0.959**

TABLE V: Effects of the embedder (BERT & adapter) and LLaMA model size, where ‘Mem.’ indicates GPU memory usage (GB), and ‘Tim.’ indicates training time (Minutes). ‘-’ indicates an out-of-memory (OOM) error.

	HDFS					BGL					Liberty					Thunderbird				
	Prec.	Rec.	F_1	Mem.	Tim.	Prec.	Rec.	F_1	Mem.	Tim.	Prec.	Rec.	F_1	Mem.	Tim.	Prec.	Rec.	F_1	Mem.	Tim.
L.-1B	0.986	0.995	0.991	16.5	1022.1	-	-	-	-	-	0.960	0.699	0.809	42.6	443.2	**1.000**	0.724	0.840	44.5	1732.1
Emb. & L.-1B	**0.996**	0.996	0.996	8.0	1412.2	0.734	0.944	0.825	32.4	187.1	0.950	0.905	0.927	29.3	173.2	0.875	0.966	0.918	32.4	715.1
L.-8B	0.988	0.997	0.992	43.0	4712.1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Emb. & L.-8B	0.994	**1.000**	**0.997**	16.6	2168.2	**0.861**	**0.979**	**0.916**	38.0	396.2	**0.992**	**0.926**	**0.958**	36.1	412.1	0.966	**0.966**	**0.966**	38.2	1284.2

computational cost due to its large number of parameters. However, the testing time of LogLLM remains acceptable when compared to other methods that utilize LLMs, such as LogBERT, NeuralLog, and RAPID.

F. Different Preprocessing Techniques (RQ2)

We evaluate the effectiveness of the different preprocessing techniques. The results are shown in Table IV. In this table, ‘*Raw*’ indicates that the content of log messages is not preprocessed and is directly input into the proposed deep model. ‘*Template*’ indicates that sequences of log templates produced by Drain [51], a log parser, are used as input for the proposed deep model. ‘*Template ID*’ signifies that the IDs of log templates, obtained by Drain, are simply encoded into numeric vectors using an embedding layer instead of BERT. The preprocessing technique ‘Template ID’ renders the model unable to capture the semantic information within log messages. Notably, the parser Drain is applied to the entire dataset, rather than only the training dataset, to avoid performance degradation due to the OOV problem. ‘*RE*’ indicates that regular expressions, as introduced in Section IV-A, are used for preprocessing log messages.

As anticipated, the preprocessing technique ‘RE’ yields the highest F_1-score across all datasets. Conversely, the preprocessing technique ‘Template ID’ consistently results in the lowest F_1-score across all datasets, averaging 36.8% lower than that of ‘RE’. This can be attributed to the fact that ‘Template ID’ hinders the model’s ability to capture the semantic information within log messages, thereby impairing its capability to detect anomalies from a natural language perspective. The preprocessing techniques ‘Raw’ and ‘Template’ result in relatively good performance, but their F_1-scores are still 6.4% and 5.7% lower than that of ‘RE’, respectively. For the preprocessing technique ‘Raw’, the variable parts (parameters that carry dynamic runtime information) within the content of each log message have little influence on anomaly detection. However, due to their high randomness, they can confuse the model, making it difficult to discern anomalies. For the preprocessing technique ‘Template’, the parser is not always reliable, sometimes incorrectly removing the constant parts or retaining the variable parts, which can lead to information loss or confusion for the model, making it difficult to discern anomalies.

G. Effect of the Embedder (RQ3)

We investigate whether the embedder (BERT and adapter) is necessary for LogLLM. The results are presented in Table V. ‘L.-1B’ refers to directly inputting the log sequence (by concatenating log messages with semicolons (;) as separators into a long string) into the ‘Llama-3.2-1B’ model [3]. ‘Emb. & L.-1B’ represents LogLLM based on ‘Llama-3.2-1B’.

As expected, with the assistance of the embedder, the model requires less GPU memory, thereby avoiding out-of-memory (OOM) errors. Additionally, it enhances model performance by clarifying the boundaries between messages within a sequence. This improved representation enables the LLM to capture sequential dependencies better.

H. Effect of the Llama Model Size (RQ4)

As shown in Table V, larger LLaMA model sizes lead to better performance, at the cost of increased GPU memory usage and longer training times.

On average, compared to using Llama-3.2-1B, adopting Llama-3-8B improves the F_1-score by 4.3%, but increases GPU memory usage by 7.7 GB and extends training time by 443.2 minutes.

I. Ablation Study of the Training Procedure (RQ5)

We investigate the effect of each training procedure through an ablation study. The results are presented in Table VI, where ‘*W/O*’ denotes ‘*without*’. We have the following observations:

Skipping any training stage results in a decrease in the F_1-score across all datasets, demonstrating the effectiveness of our

[3]https://huggingface.co/meta-llama/Llama-3.2-1B

表 IV：不同预处理技术对 HDFS 、 BGL 、 Liberty 和 Thunderbird 数据集的影响。最佳结果以粗体显示。

	HDFS			BGL			Liberty			Thunderbird			Avg. F_1
	准确度	Rec.	F_1	精度	Rec.	F_1	精度	Rec.	F_1	精度	Rec.	F_1	
Raw	0.994	0.991	0.993	**0.943**	0.767	0.846	0.911	0.908	0.909	0.806	0.862	0.833	0.895
模板 ID	**0.995**	0.945	0.969	0.775	0.286	0.418	**0.994**	0.270	0.425	**1.000**	0.379	0.550	0.591
模板	0.991	1.000	0.995	0.861	0.919	0.889	0.968	**0.931**	0.949	0.950	0.655	0.776	0.902
RE (LogLLM)	0.994	**1.000**	**0.997**	0.861	**0.979**	**0.916**	0.992	0.926	**0.958**	0.966	**0.966**	**0.966**	**0.959**

TABLEV：嵌入器（ BERT & adapter ）和 LLaMA 模型大小的影响，其中 ‘Mem。’ 表示 GPU 内存使用量（ GB ）， ‘Tim。’ 表示训练时间（分钟）。 ‘-’ 表示内存不足（ OOM ）错误。

	HDFS					BGL					Liberty					雷鸟				
	Prec.	Rec.	F_1	Mem.	Tim.	精度	Rec.	F_1	Mem.	Tim.	精度	Rec.	F_1	Mem.	Tim.	精度	Rec.	F_1	Mem.	Tim.
L.-1B	0.986	0.995	0.991	16.5	1022.1	-	-	-	-	-	0.960	0.699	0.809	42.6	443.2	**1.000**	0.724	0.840	44.5	1732.1
Emb. & L.-1B	**0.996**	0.996	0.996	8.0	1412.2	0.734	0.944	0.825	32.4	187.1	0.950	0.905	0.927	29.3	173.2	0.875	0.966	0.918	32.4	715.1
L.-8B	0.988	0.997	0.992	43.0	4712.1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Emb. & L.-8B	0.994	**1.000**	**0.997**	16.6	2168.2	**0.861**	**0.979**	**0.916**	38.0	396.2	**0.992**	**0.926**	**0.958**	36.1	412.1	0.966	**0.966**	**0.966**	38.2	1284.2

由于其参数数量庞大，计算成本较高。然而，与使用 LLM 的其他方法（如 LogBERT 、 NeuralLog 和 RAPID ）相比，LogLLM 的测试时间仍然可接受。

F. 不同预处理技术（ *RQ2* ）

我们评估了不同预处理技术的有效性。结果如表 IV 所示。在此表中， ‘*Raw*’ 表示日志消息的内容未经预处理，直接输入到所提出的深度模型中。 ‘*Template*’ 表示由 Drain （一个日志解析器）生成的日志模板序列被用作所提出的深度模型的输入。 ‘*TemplateID*’ 表示由 Drain 获得的日志模板 ID 被简单地编码为数值向量，而不是使用 BERT。预处理技术 ‘Template ID’ 使模型无法捕获日志消息中的语义信息。值得注意的是，Drain 解析器应用于整个数据集，而不仅仅是训练数据集，以避免由于 OOV 问题导致的性能下降。 ‘*RE*’ 表示使用正则表达式进行日志消息的预处理。

正如预期的那样，预处理技术 ‘RE’ 在所有数据集中产生了最高的 F_1- 分数。相反，预处理技术 ‘Template ID’ 在所有数据集中始终产生最低的 F_1- 分数，平均比 ‘RE’ 低 36.8%。这可以归因于 ‘Template ID’ 阻碍了模型捕获日志消息中语义信息的能力，从而损害了其从自然语言角度检测异常的能力。预处理技术 ‘Raw’ 和 ‘Template’ 的结果相对较好，但它们的 F_1- 分数仍然比 ‘RE’ 低 6.4% 和 5.7%。对于预处理技术 ‘Raw’，每个日志消息内容中的可变部分（携带动态运行时信息的参数）对异常检测的影响很小。然而，由于它们的高度随机性，它们可能会混淆模型，使得难以辨别异常。对于预处理技术 ‘ 模板 ’，解析器并不总是可靠的，有时会错误地删除常数部分或保留变量部分，这可能导致信息丢失或模型混淆，使得难以辨别异常。

G. Embedder 的影响（ *RQ3* ）

我们研究了嵌入器（ BERT 和适配器）对 LogLLM 是否必要。结果如表 V 所示。 ‘L.-1B’ 指的是直接将日志序列（通过将日志消息以分号（ ; ）作为分隔符连接成一个长字符串）输入到 ‘Llama-3.2-1B’ 模型 [3] 中。 ‘Emb. & L.-1B’ 表示基于 ‘Llama-3.2-1B’ 的 LogLLM。

不出所料，在嵌入器的帮助下，模型需要的 GPU 内存更少，从而避免了内存不足（ OOM ）错误。此外，它通过阐明序列中消息之间的边界来提高模型性能。这种改进的表示使得 LLM 能够更好地捕捉序列依赖关系。

H. Llama 模型大小的影响（ *RQ4* ）

如表 V 所示，更大的 LLaMA 模型尺寸会导致更好的性能，但代价是增加了 GPU 内存使用量和更长的训练时间。

平均而言，与使用 Llama-3.2-1B 相比，采用 Llama-3-8B 将 F1 分数提高 4.3%，但增加了 7.7 GB 的 GPU 内存使用量，并将训练时间延长了 443.2 分钟。

I. 训练过程消融研究（ *RQ5* ）

我们通过消融研究调查了每个训练过程的影响。结果如表 VI 所示，其中 “*W/O*” 表示 “*without*”。我们的观察如下：

跳过任何训练阶段都会导致所有数据集上的 F1 分数下降，证明了我们方法的有效性。

[3]https://huggingface.co/meta-llama/Llama-3.2-1B

TABLE VI: Ablation study of the training procedure on HDFS, BGL, Liberty, and Thunderbird datasets. The best results are highlighted in bold.

	HDFS			BGL			Liberty			Thunderbird			Avg. F_1
	Prec.	Rec.	F_1	Prec.	Rec.	F_1	Prec.	Rec.	F_1	Prec.	Rec.	F_1	
W/O Stage 1	0.991	1.000	0.995	0.578	0.971	0.725	0.685	0.290	0.408	0.381	0.828	0.522	0.662
W/O Stage 2	0.994	1.000	0.997	0.858	0.920	0.888	0.995	0.906	0.949	0.848	0.966	0.903	0.934
W/O Stage 1&2	0.992	1.000	0.996	0.853	0.882	0.868	0.995	0.906	0.949	0.897	0.897	0.897	0.927
W/O Stage 3	0.993	0.999	0.996	0.704	0.776	0.738	**1.000**	0.684	0.812	0.958	0.793	0.868	0.854
LogLLM	**0.994**	**1.000**	**0.997**	**0.861**	**0.979**	**0.916**	0.992	**0.926**	**0.958**	**0.966**	**0.966**	**0.966**	**0.959**

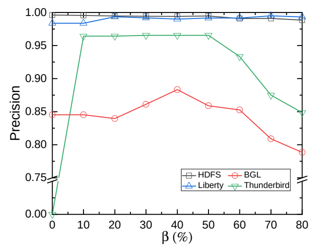
three-stage training procedure. It is noteworthy that training without stage 1 leads to the worst performance, with the F_1-score averaged across all datasets decreasing by as much as 29.7%. However, training without stages 1&2 (only adopting training stage 3: fine-tuning the entire model) yields acceptable performance, with only a 3.2% decrease in the average F_1-score. This demonstrates that fine-tuning Llama to capture the answer template (Stage 1) is essential before training the embedder (BERT and projector) of log messages (Stage 2). Without stage 1 (i.e., directly training the embedder), the embedder may be misdirected, resulting in incorrect semantic capture of log messages and model failure. Training without stage 3 yields relatively poor performance, with an average F_1-score decrease of 10.5%. This indicates that sequentially fine-tuning Llama and training the embedder alone is insufficient for the model to capture anomalous patterns; cohesive fine-tuning of the entire model is essential. Training without stages 2 and 1&2 also results in a performance decrease, with average F_1-score reductions of 2.5% and 3.2%, respectively. This demonstrates that individually training the embedder before fine-tuning the entire model can also enhance performance. This stage allows the embedder to generate better semantic vectors of log messages for Llama to discern anomalies.

In summary, our proposed three-stage training procedure is well-suited for our deep model in log-based anomaly detection.

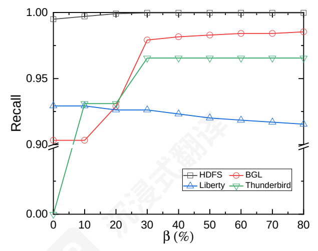
J. Impact of Minority Class Oversampling (RQ6)

Note that normal and anomalous samples in the training dataset are imbalanced, as shown in Table I. For the HDFS, BGL, and Thunderbird datasets, normal samples outnumber anomalous samples. Conversely, in the Liberty dataset, anomalous samples exceed normal samples. As described in Section IV-C1, the hyper-parameter β controls the proportion of the minority class by oversampling to address the data imbalance problem. In this section, we investigate the impact of β by varying its value. Fig. 4 illustrates the performance of LogLLM on the four datasets under different magnitudes of β. When $\beta = 0$, the samples are not oversampled; instead, the original datasets are utilized directly for training.

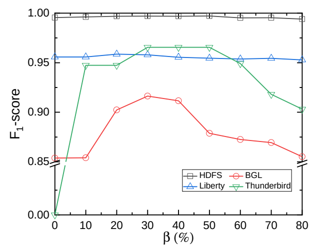
As illustrated in Fig. 4b, for the HDFS, BGL, and Thunderbird datasets, the recall always increases, while for the Liberty dataset, recall decreases as β increases. This can be attributed to the fact that for the HDFS, BGL, and Thunderbird datasets, when β increases, anomalies are oversampled, making the model more prone to identifying samples as anomalies. In contrast, for the Liberty dataset, when β increases, normal samples are oversampled, making the model more prone to identifying samples as normal.



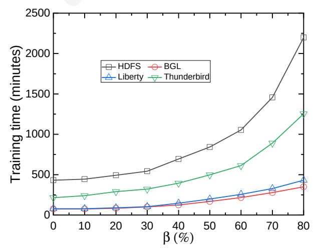
(a) Precision



(b) Recall



(c) F_1-score



(d) Training time

Fig. 4: Impact of minority class oversampling.

As illustrated in Fig. 4c, the trend of the F_1-score is basically the same across all datasets. The F_1-score increases and then decreases as β increases. However, the LogLLM seems not to be sensitive to β; when β is between 10% and 80%, the variation in the F_1-score is no more than 0.07. Thanks to the substantial semantic knowledge embedded in LLMs, a trained model can effectively learn anomalous patterns and detect anomalies, even when the minority class constitutes only 10% of the dataset. However, LogLLM appears unable to effectively handle extremely imbalanced scenarios. For instance, in the Thunderbird dataset, anomalies constitute only 1.05% of the samples, causing the trained model to be biased and classify all samples as normal. As a result, precision, recall, and F_1-score are all equal to 0.

Compared to the BGL and Thunderbird datasets, the precision, recall and F_1-score for the HDFS and Liberty datasets exhibit minimal variation with respect to β. This consistency arises from the more distinct patterns between abnormal and normal samples in the HDFS and Liberty datasets, allowing LogLLM to easily differentiate them, regardless of the ratio

表 VI：在 HDFS 、 BGL 、 Liberty 和 Thunderbird 数据集上对训练过程的消融研究。最佳结果以粗体显示。

	HDFS			BGL			Liberty			Thunderbird			Avg. F_1
	准确率	Rec.	F_1	精确度	Rec.	F_1	精确度	Rec.	F_1	精确度	Rec.	F_1	
无第 1 阶段	0.991	1.000	0.995	0.578	0.971	0.725	0.685	0.290	0.408	0.381	0.828	0.522	0.662
不包含阶段 2	0.994	1.000	0.997	0.858	0.920	0.888	0.995	0.906	0.949	0.848	0.966	0.903	0.934
无阶段 1&2	0.992	1.000	0.996	0.853	0.882	0.868	0.995	0.906	0.949	0.897	0.897	0.897	0.927
无阶段 3	0.993	0.999	0.996	0.704	0.776	0.738	**1.000**	0.684	0.812	0.958	0.793	0.868	0.854
LogLLM	**0.994**	**1.000**	**0.997**	**0.861**	**0.979**	**0.916**	0.992	**0.926**	**0.958**	**0.966**	**0.966**	**0.966**	**0.959**

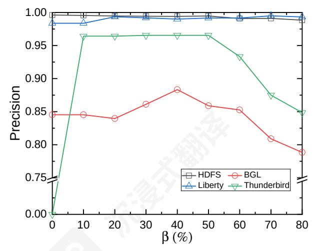
三阶段训练流程。值得注意的是，没有第一阶段训练会导致最差的表现，所有数据集上平均的 F1- 分数下降了高达 29.7%。然而，没有第一阶段和第二阶段训练（仅采用第三阶段：对整个模型进行 ne-tuning ）仍然可以获得可接受的表现，平均 F1- 分数仅下降了 3.2%。这表明在训练日志消息的嵌入器（ BERT 和 projector ）（第二阶段）之前，先对 Llama 进行 ne-tuning 以捕获答案模板（第一阶段）是至关重要的。如果没有第一阶段（即直接训练嵌入器），嵌入器可能会被误导，导致对日志消息的语义捕获不正确，从而导致模型失败。没有第三阶段训练会导致相对较差的表现，平均 F1- 分数下降了 10.5%。这表明仅仅对 Llama 进行 ne-tuning 和单独训练嵌入器不足以让模型捕获异常模式；对整个模型进行连贯的 ne-tuning 是必要的。没有第二阶段和第一阶段和第二阶段训练也会导致性能下降，平均 F1- 分数分别下降了 2.5 % 和 3.2%。这表明在 ne-tuning 整个模型之前单独训练嵌入器也可以提高性能。这一阶段允许嵌入器为 Llama 生成更好的日志消息语义向量，以便其能够识别异常。

总之，我们提出的基于日志的深度模型三阶段训练流程非常适合我们的异常检测。

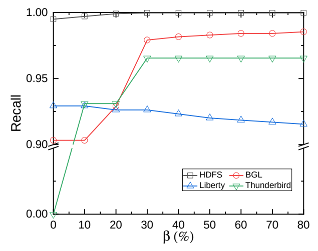
J. Impact of Minority Class Oversampling (RQ6)

请注意，训练数据集中的正常样本和异常样本是不平衡的，如表 I 所示。对于 HDFS 、 BGL 和 Thunderbird 数据集，正常样本的数量多于异常样本。相反，在 Liberty 数据集中，异常样本的数量超过正常样本。如第 IV-C1 节所述，超参数 β 通过过采样来控制少数类的比例，以解决数据不平衡问题。在本节中，我们通过改变其值来研究 β 的影响。图 4 展示了在不同 β 幅度下 LogLLM 在四个数据集上的性能。当 $\beta = 0$ 时，样本不进行过采样；而是直接使用原始数据集进行训练。

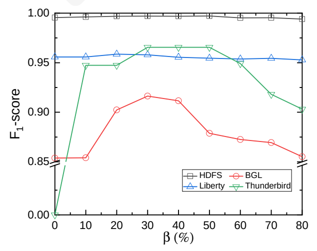
如图 4b 所示，对于 HDFS 、 BGL 和 Thunderbird 数据集，召回率始终在增加，而对于 Liberty 数据集，随着 β 的增加，召回率在下降。这可以归因于，对于 HDFS 、 BGL 和 Thunderbird 数据集，当 β 增加时，异常样本被过采样，使得模型更容易将样本识别为异常。



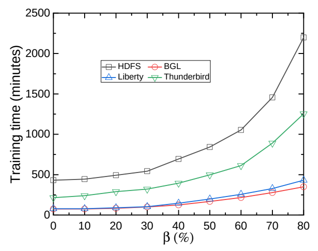
(a) Precision



(b) Recall



(c) F_1-score



(d) Training time

图 4：少数类过采样的影响。

相比之下，对于 Liberty 数据集，当 β 增加时，正常样本被过采样，使得模型更容易将样本识别为正常。

如图 4c 所示，F1- 分数的趋势在所有数据集中基本上是相同的。F1- 分数随着 β 的增加先上升后下降。然而，LogLLM 似乎对 β 不敏感；当 β 在 10% 到 80% 之间时，F1- 分数的变化不超过 0.07。得益于 LLM 中嵌入的丰富语义知识，训练好的模型可以有效地学习异常模式并检测异常，即使少数类仅占数据集的 10%。然而，LogLLM 似乎无法有效地处理极端不平衡的场景。例如，在 Thunderbird 数据集中，异常样本仅占样本的 1.05%，导致训练好的模型产生偏差，将所有样本分类为正常。因此，精确度、召回率和 F1- 分数都等于 0。

与 BGL 和 Thunderbird 数据集相比，HDFS 和 Liberty 数据集的精确度、召回率和 F1 分数在 β 方面表现出最小变化。这种一致性源于 HDFS 和 Liberty 数据集中异常样本和正常样本之间更明显的模式，使得 LogLLM 能够轻松区分它们，无论比例如何

of normal and abnormal samples.

As anticipated, as β increases, the training time also increases, as shown in Fig. 4d. This relationship arises because a higher β leads to more oversampled data samples, as indicated by equation (1), thereby enlarging the training dataset.

To summarize, minority class oversampling is essential; however, the value of the hyperparameter β does not significantly impact the performance of LogLLM, making careful selection unnecessary. Moreover, excessively large values of β are undesirable, as they result in prolonged training times. Values between 30% and 50% are deemed acceptable.

VI. Conclusion

In this paper, we propose LogLLM, a novel log-based anomaly detection framework that leverages LLMs. LogLLM employs both transformer encoder-based and decoder-based LLMs, specifically BERT and Llama, for log-based anomaly detection. BERT is utilized to extract semantic vectors from log messages, while Llama is used to classify log sequences. To ensure coherence in log semantics, we introduce a projector that aligns the vector representation spaces of BERT and Llama. LogLLM is trained using an innovative three-stage procedure designed to enhance both performance and adaptability. Extensive experiments conducted on four public real-world datasets demonstrate that LogLLM achieves remarkable performance. Subsequent ablation studies further confirm the effectiveness of our three-stage training procedure.

References

[1] R. S. Kazemzadeh and H.-A. Jacobsen, "Reliable and highly available distributed publish/subscribe service," in *2009 28th IEEE International Symposium on Reliable Distributed Systems*. IEEE, 2009, pp. 41–50.

[2] E. Bauer and R. Adams, *Reliability and availability of cloud computing*. John Wiley & Sons, 2012.

[3] V.-H. Le and H. 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.

[4] W. Guan, J. Cao, H. Zhao, Y. Gu, and S. Qian, "Survey and benchmark of anomaly detection in business processes," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–23, 2024.

[5] S. Zhang, Y. Ji, J. Luan, X. Nie, Z. Chen, M. Ma, Y. Sun, and D. Pei, "End-to-end automl for unsupervised log anomaly detection," *Automated Software Engineering (ASE'24)*, 2024.

[6] V.-H. Le and H. 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.

[7] J. Qi, S. Huang, Z. Luan, S. Yang, C. Fung, H. Yang, D. Qian, J. Shang, Z. Xiao, and Z. 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.

[8] M. Du, F. Li, G. Zheng, and V. 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.

[9] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun *et al.*, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs." in *IJCAI*, vol. 19, no. 7, 2019, pp. 4739–4745.

[10] L. Zhang, W. Li, Z. Zhang, Q. Lu, C. Hou, P. Hu, T. Gui, and S. Lu, "Logattn: Unsupervised log anomaly detection with an autoencoder based attention mechanism," in *International conference on knowledge science, engineering and management*. Springer, 2021, pp. 222–235.

[11] M. Catillo, A. Pecchia, and U. Villano, "Autolog: Anomaly detection by deep autoencoding of system logs," *Expert Systems with Applications*, vol. 191, p. 116263, 2022.

[12] Y. Xie and K. Yang, "Log anomaly detection by adversarial autoencoders with graph feature fusion," *IEEE Transactions on Reliability*, 2023.

[13] X. Zhang, X. Chai, M. Yu, and D. Qiu, "Anomaly detection model for log based on lstm network and variational autoencoder," in *2023 4th International Conference on Information Science, Parallel and Distributed Systems (ISPDS)*. IEEE, 2023, pp. 239–244.

[14] X. Duan, S. Ying, W. Yuan, H. Cheng, and X. Yin, "A generative adversarial networks for log anomaly detection." *Comput. Syst. Sci. Eng.*, vol. 37, no. 1, pp. 135–148, 2021.

[15] Z. He, Y. Tang, K. Zhao, J. Liu, and W. Chen, "Graph-based log anomaly detection via adversarial training," in *International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*. Springer, 2023, pp. 55–71.

[16] C. Zhang, X. Wang, H. Zhang, J. Zhang, H. Zhang, C. Liu, and P. Han, "Layerlog: Log sequence anomaly detection based on hierarchical semantics," *Applied Soft Computing*, vol. 132, p. 109860, 2023.

[17] S. Hashemi and M. Mäntylä, "Onelog: towards end-to-end software log anomaly detection," *Automated Software Engineering*, vol. 31, no. 2, p. 37, 2024.

[18] S. Lu, X. Wei, Y. Li, and L. Wang, "Detecting anomaly in big data system logs using convolutional neural network," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. IEEE, 2018, pp. 151–158.

[19] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. 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.

[20] Y. Xie, H. Zhang, and M. A. Babar, "Loggd: Detecting anomalies from system logs with graph neural networks," in *2022 IEEE 22nd International conference on software quality, reliability and security (QRS)*. IEEE, 2022, pp. 299–310.

[21] Z. Zhao, W. Niu, X. Zhang, R. Zhang, Z. Yu, and C. Huang, "Trine: Syslog anomaly detection with three transformer encoders in one generative adversarial network," *Applied Intelligence*, vol. 52, no. 8, pp. 8810–8819, 2022.

[22] L. Yang, J. Chen, Z. Wang, W. Wang, J. Jiang, X. Dong, and W. 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.

[23] S. Hochreiter, "Long short-term memory," *Neural Computation MIT-Press*, 1997.

[24] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[25] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[26] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[27] T. GLM, A. Zeng, B. Xu, B. Wang, C. Zhang, D. Yin, D. Rojas, G. Feng, H. Zhao, H. Lai *et al.*, "Chatglm: A family of large language models from glm-130b to glm-4 all tools," *arXiv preprint arXiv:2406.12793*, 2024.

[28] W. Guan, J. Cao, J. Gao, H. Zhao, and S. Qian, "Dabl: Detecting semantic anomalies in business processes using large language models," *arXiv preprint arXiv:2406.15781*, 2024.

[29] Y. Liu, S. Tao, W. Meng, F. Yao, X. Zhao, and H. Yang, "Logprompt: Prompt engineering towards zero-shot and interpretable log analysis," in *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, 2024, pp. 364–365.

[30] C. Egersdoerfer, D. Zhang, and D. Dai, "Early exploration of using chatgpt for log-based anomaly detection on parallel file systems logs," in *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing*, 2023, pp. 315–316.

[31] J. Pan, W. S. Liang, and Y. Yidi, "Raglog: Log anomaly detection using retrieval augmented generation," in *2024 IEEE World Forum on Public Safety Technology (WFPST)*. IEEE, 2024, pp. 169–174.

正常和异常样本。

正如预期的那样，随着 β 的增加，训练时间也随之增加，如图 4d 所示。这种关系产生的原因是更高的 β 导致更多的过采样数据样本，如方程（1）所示，从而扩大了训练数据集。

总结来说，少数类过采样是必要的；然而，超参数 β 的值对 LogLLM 的性能没有显著影响，因此无需仔细选择。此外，过大的 β 值是不理想的，因为它们会导致训练时间延长。30% 到 50% 之间的值被认为是可接受的。

VI. 结论

在本文中，我们提出了 LogLLM，这是一种基于日志的、利用 LLMs 的异常检测新框架。LogLLM 使用基于 transformer 编码器和解码器的 LLMs，特别是 BERT 和 Llama，进行基于日志的异常检测。BERT 用于从日志消息中提取语义向量，而 Llama 用于分类日志序列。为了确保日志语义的一致性，我们引入了一个投影器，以对齐 BERT 和 Llama 的向量表示空间。LogLLM 使用一个创新的三阶段程序进行训练，旨在提高性能和适应性。在四个公开的真实世界数据集上进行的广泛实验表明，LogLLM 取得了显著的性能。随后的消融研究进一步证实了我们的三阶段训练程序的有效性。

参考文献

[1] R. S. Kazemzadeh 和 H.-A. Jacobsen，‘可靠且高度可用的分布式发布 / 订阅服务’，载于 *2009 第 28 届 IEEE* 国际可靠分布式系统研讨会 . IEEE，2009，第 41–50 页。

[2] E. Bauer 和 R. Adams，云计算的可靠性和可用性 . 约翰 · 威利与 Sons，2012。

[3] V.-H. Le 和 H. Zhang，‘基于日志的无日志解析异常检测’，载于 *2021 第 36 届 IEEE/ACM* 国际自动软件工程会议 *(ASE)*. IEEE，2021，第 492–504 页。

[4] W. Guan，J. Cao，H. Zhao，Y. Gu 和 S. Qian，‘业务流程异常检测综述与基准测试’，*IEEE* 知识和数据工程杂志，第 1–23 页，2024。

[5] S. Zhang，Y. Ji，J. Luan，X. Nie，Z. Chen，M. Ma，Y. Sun 和 D. Pei，‘面向无监督日志异常检测的端到端自动机器学习’，自动软件工程 *(ASE'24)*，2024。

[6] V.-H. Le 和 H. Zhang，‘基于深度学习的日志异常检测：我们还有多远？’，载于 第 *44* 届国际软件工程会议论文集，2022，第 1356–1367 页。

[7] J. Qi, S. Huang, Z. Luan, S. Yang, C. Fung, H. Yang, D. Qian, J. Shang, Z. Xiao, and Z. Wu, "Loggpt: Exploring chatgpt for log-based anomaly 基于日志的大语言模型异常检测，在 *2023 IEEE* 国际高性能计算与通信、数据科学及系统、智能城市与传感器、云及大数据系统与应用（*HPCC/DSS/SmartCity/DependSys*）会议论文集中，*IEEE*，*2023*，第 *273–280* 页。

[8] M. Du, F. Li, G. Zheng, 和 V. Srikumar，在 2017 年 ACM SIGSAC 计算机与通信安全会议论文集中，‘Deeplog：通过深度学习从系统日志中进行异常检测与诊断’，2017，第 1285–1298 页。

[9] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun 等人，‘Loganomaly：无监督检测非结构化日志中的顺序和定量异常’，在 IJCAI 会议论文集中，第 19 卷第 7 期，2019，第 4739–4745 页。

[10] L. Zhang, W. Li, Z. Zhang, Q. Lu, C. Hou, P. Hu, T. Gui, 和 S. Lu，‘Logattn：基于自动编码器的注意力机制的日志无监督异常检测’，在知识科学、工程与管理国际会议上，Springer，2021，第 222–235 页。

[11] M. Catillo, A. Pecchia, 和 U. Villano，“通过系统日志的深度自动编码进行异常检测的 Autolog”，在 Expert Systems with Applications 期刊中，第 191 卷，p. 116263, 2022。

[12] Y. Xie 和 K. Yang，“通过图特征融合的对抗性自动编码器进行日志异常检测”，在 IEEE Transactions on Reliability 期刊中，2023。

[13] 张 X，蔡 X，余 M，邱 D，“基于 LSTM 网络和变分自编码器的日志异常检测模型”，载于 2023 年第 4 届信息科学、并行与分布式系统国际会议 (ISPDS)，IEEE，2023，第 239–244 页。

[14] 段 X，英 S，袁 W，程 H，尹 X，“用于日志异常检测的生成对抗网络。”计算机系统科学与工程，第 37 卷，第 1 期，第 135–148 页，2021 年。

[15] 何 Z，唐 Y，赵 K，刘 J，陈 W，“基于对抗训练的图日志异常检测”，载于国际可靠软件工程研讨会：理论、工具和应用，Springer，2023，第 55–71 页。

[16] 张 C，王 X，张 H，张 J，张 H，刘 C，韩 P，“基于层次语义的日志序列异常检测：Layerlog”，应用软件计算，第 132 卷，第 109860 页，2023 年。

[17] 哈希米 S，马恩蒂拉 M，“Onelog：面向端到端软件日志异常检测”，自动化软件工程，第 31 卷，第 2 期，第 37 页，2024 年。

[18] *卢 S，魏 X，李 Y，王 L*，“使用卷积神经网络在大数据系统日志中检测异常”，载于 *2018* 年第 *16* 届 *IEEE* 可靠、自主和安全的计算国际会议，第 *16* 届国际普适智能计算会议，第 *4* 届大数据智能计算和网络安全技术大会 *(DASC/PiCom/DataCom/CyberSciTech)*，*IEEE*，*2018*，第 *151–158* 页。

[19] 张 X，徐 Y，林 Q，乔 B，张 H，当 Y，谢 C，杨 X，程 Q，李 Z 等 .（v1）“基于日志的鲁棒不稳定日志数据异常检测”，载于 v3 第 27 届 ACM 欧洲软件工程会议和软件工程基础研讨会联合会议论文集（v4），2019 年，第 807–817 页。

[20] 谢 Y，张 H，巴巴尔 M.A.，“Loggd：利用图神经网络从系统日志中检测异常”，载于 v1 第 22 届 IEEE 国际软件质量、可靠性和安全性会议（v2），IEEE，2022 年，第 299–310 页。

[21] 赵 Z，牛 W，张 X，张 R，余 Z，黄 C，“Trine：在生成对抗网络中使用三个 Transformer 编码器进行 syslog 异常检测”，载于 v1《应用智能》（v2），第 52 卷第 8 期，第 8810–8819 页，2022 年。

[22] 杨 L，陈 J，王 Z，王 W，江 J，董 X，张 W，“通过概率标签估计进行半监督日志异常检测”，载于 v1 第 43 届 IEEE/ACM 国际软件工程会议（ICSE）（v2），IEEE，2021 年，第 1448–1460 页。

[23] 霍克瑞特 S，“长短期记忆”，载于 v1《神经计算》（MIT-Press）（v2），1997 年。

[24] *瓦斯瓦尼 A*，“注意力即是所需”，载于 *v2*《神经信息处理系统进展》（*v3*），*2017* 年。

[25] J. Achiam, S. Adler, S.Agarwal, L. Ahmad, I. Akkaya, F.L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat 等人，“Gpt-4 技术报告”，arXiv 预印本 arXiv:2303.08774，2023 年。

[26] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan 等人，“Llama 3 模型群”，arXiv 预印本 arXiv:2407.21783，2024 年。

[27] T.GLM, A. Zeng, B. Xu, B. Wang, C. Zhang, D. Yin, D. Rojas, G. Feng, H. Zhao, H. Lai 等人，“ChatGLM：从 glm-130b 到 glm-4 的全系列大语言模型工具”，arXiv 预印本 arXiv:2406.12793，2024 年。

[28] W. Guan, J. Cao, J. Gao, H. Zhao, 和 S. Qian，“Dabl：使用大语言模型检测业务流程中的语义异常”，arXiv 预印本 arXiv:2406.15781，2024 年。

[29] Y. Liu, S. Tao, W. Meng, F. Yao, X. Zhao, 和 H. Yang，“Logprompt：面向零样本和可解释日志分析的提示工程”，在 2024 IEEE/ACM 第 46 届国际软件工程会议：补充会议论文集，2024 年，第 364–365 页。

[30] C. Egersdoerfer, D. Zhang, 和 D. Dai，“早期探索使用 ChatGPT 对并行文件系统日志进行基于日志的异常检测”，在 第 32 届高性能并行和分布式计算国际研讨会，2023 年，第 315–316 页。

[31] 潘杰，梁伟胜，以及叶一笛，“Raglog：基于检索增强生成的日志异常检测”，载于 2024 年 IEEE 公共安全技术世界论坛（WFPST）。IEEE，2024 年，第 169-174 页。

[32] H. Guo, S. Yuan, and X. Wu, "Logbert: Log anomaly detection via bert," in *2021 international joint conference on neural networks (IJCNN)*. IEEE, 2021, pp. 1–8.

[33] Y. Lee, J. Kim, and P. Kang, "Lanobert: System log anomaly detection based on bert masked language model," *Applied Soft Computing*, vol. 146, p. 110689, 2023.

[34] Y. Lin, H. Deng, and X. Li, "Fastlogad: Log anomaly detection with mask-guided pseudo anomaly generation and discrimination," *arXiv preprint arXiv:2404.08750*, 2024.

[35] C. Almodovar, F. Sabrina, S. Karimi, and S. Azad, "Logfit: Log anomaly detection using fine-tuned language models," *IEEE Transactions on Network and Service Management*, 2024.

[36] S. Chen and H. Liao, "Bert-log: Anomaly detection for system logs based on pre-trained language model," *Applied Artificial Intelligence*, vol. 36, no. 1, p. 2145642, 2022.

[37] J. L. Adeba, D.-H. Kim, and J. Kwak, "Sarlog: Semantic-aware robust log anomaly detection via bert-augmented contrastive learning," *IEEE Internet of Things Journal*, 2024.

[38] Y. Fu, K. Liang, and J. Xu, "Mlog: Mogrifier lstm-based log anomaly detection approach using semantic representation," *IEEE Transactions on Services Computing*, vol. 16, no. 5, pp. 3537–3549, 2023.

[39] G. No, Y. Lee, H. Kang, and P. Kang, "Training-free retrieval-based log anomaly detection with pre-trained language model considering token-level information," *Engineering Applications of Artificial Intelligence*, vol. 133, p. 108613, 2024.

[40] F. Hadadi, Q. Xu, D. Bianculli, and L. Briand, "Anomaly detection on unstable logs with gpt models," *arXiv preprint arXiv:2406.07467*, 2024.

[41] M. Burtsev, M. Reeves, and A. Job, "The working limitations of large language models," *MIT Sloan Management Review*, vol. 65, no. 2, pp. 8–10, 2024.

[42] A. Joulin, "Fasttext. zip: Compressing text classification models," *arXiv preprint arXiv:1612.03651*, 2016.

[43] J. Devlin, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[44] Y. Liu, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[45] M. Joshi, D. Chen, Y. Liu, D. S. Weld, L. Zettlemoyer, and O. Levy, "Spanbert: Improving pre-training by representing and predicting spans," *Transactions of the association for computational linguistics*, vol. 8, pp. 64–77, 2020.

[46] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020.

[47] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. 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.

[48] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "An evaluation study on log parsing and its use in log mining," in *2016 46th annual IEEE/IFIP international conference on dependable systems and networks (DSN)*. IEEE, 2016, pp. 654–661.

[49] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Online system problem detection by mining patterns of console logs," in *2009 ninth IEEE international conference on data mining*. IEEE, 2009, pp. 588–597.

[50] A. Oliner and J. 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.

[51] P. He, J. Zhu, Z. Zheng, and M. 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.

[52] M. Du and F. Li, "Spell: Streaming parsing of system event logs," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 2016, pp. 859–864.

[53] V.-H. Le and H. 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.

[54] J. S. Bridle, "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition," in *Neurocomputing: Algorithms, architectures and applications*. Springer, 1990, pp. 227–236.

[55] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[56] I. Loshchilov, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[57] J. Zhu, S. He, P. He, J. Liu, and M. 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.

[32] H. Guo, S.Yuan, 和 X. Wu, "Logbert：基于 BERT 的日志异常检测", 载于 *2021* 年国际神经网络联合会议（*IJCNN*）. IEEE，2021，第 1-8 页。

[33] Y. Lee, J. Kim, 和 P. Kang, "Lanobert：基于 BERT 掩码语言模型的系统日志异常检测", 应用软计算，第 146 卷，第 110689 页，2023 年。

[34] Y. Lin, H. Deng, 和 X.Li, "Fastlogad：基于掩码引导的伪异常生成和判别的日志异常检测", *arXiv* 预印本 *arXiv:2404.08750*，2024 年。

[35] C. Almodovar, F. Sabrina, S. Karimi, 和 S. Azad, "Log t：使用 NE 调优的语言模型的日志异常检测", *IEEE* 网络和服务管理杂志，2024 年。

[36] S. Chen 和 H. Liao, "Bert-log：基于预训练语言模型的系统日志异常检测", 应用人工智能，第 36 卷，第 1 期，第 2145642 页，2022 年。

[37] J.L. Adeba, D.-H. Kim, 和 J.Kwak, "Sarlog：通过 BERT 增强的对比学习实现的语义感知鲁棒日志异常检测", *IEEE* 物联网杂志，2024 年。

[38] Y. Fu, K. Liang, 和 J. Xu, "Mlog: 基于语义表示的 Mogri er LSTM 日志异常检测方法 ," *IEEE Transactionson Services Computing*, 第16卷，第 5 期，第 3537–3549 页，2023 年。

[39] G. No, Y.Lee, H. Kang, 和 P. Kang, "考虑 token 级信息的预训练语言模型的无监督检索日志异常检测," *Engineering Applications of Artificial Intelligence*, 第 133 卷，第 108613 页，2024 年。

[40] F. Hadadi, Q. Xu, D. Bianculli, 和 L. Briand, "使用 gptmodels 在不稳定日志上的异常检测," *arXiv* 预印本 *arXiv:2406.07467*，2024 年。

[41] M. Burtsev, M. Reeves, 和 A. Job, "大型语言模型的工作局限性," *MIT Sloan Management Review*, 第65 卷，第 2 期，第 8–10 页，2024 年。

[42] A. Joulin, "Fasttext. zip: 压缩文本分类模型," *arXiv* 预印本 *arXiv:1612.03651*, 2016 年。

[43] J. Devlin, "Bert: 用于语言理解的深度双向变换器预训练," *arXiv* 预印本 *arXiv: 1810.04805*，2018 年。

[44] 刘宇, "Roberta：一种鲁棒优化的 BERT 预训练方法", 预印本 arXiv:1907.11692, 2019 年。

[45] M. Joshi，D. Chen，Y. Liu，D. S. Weld，L. Zettlemoyer，和 O. Levy, "通过表示和预测跨度来改进预训练：SpanBERT", 计算语言学协会会刊，第 8 卷，第 64-77 页，2020 年。

[46] P. Lewis，E. Perez，A. Piktus，F. Petroni，V. Karpukhin，N. Goyal，H. Küttler, M. Lewis，W.-t. Yih，T. Rocktäschel 等人, "用于知识密集型 NLP 任务的检索增强生成", 神经信息处理系统进展，第 33 卷，第 9459-9474 页，2020 年。

[47] 朱珠，何思，刘杰，何平，谢奇，郑振，和吕明仁, "用于自动日志解析的工具和基准", 在 2019 IEEE/ACM 第 41 届国际软件工程会议：软件工程实践（ICSE-SEIP）中，IEEE，2019 年，第 121-130 页。

[48] 何平，朱珠，何思，李杰，和吕明仁, "关于日志解析及其在日志挖掘中的应用的评估研究", 在 2016 第 46 届 IEEE/IFIP 国际可靠系统与网络会议（DSN）中，IEEE，2016 年，第 654-661 页。

[49] Xu W，Huang L，Fox A，Patterson D，和 Jordan M, "通过挖掘控制台日志模式进行在线系统问题检测", 在 2009 第 9 届 IEEE 国际数据挖掘会议中，IEEE，2009 年，第 588-597 页。

[50] A. Oliner 和 J. Stearley, "超级计算机说了什么：对五个系统日志的研究", 载于 第 *37* 届 *IEEE/IFIP* 国际可靠系统与网络会议（*DSN' 07*）. IEEE，2007 年，第 575– 584 页。

[51] P. He, J. Zhu, Z. Zheng 和 M. R. Lyu, "Drain：一种具有固定深度的在线日志解析方法", 载于 *2017* 年 *IEEE* 国际 *Web* 服务会议（*ICWS*）. IEEE，2017 年，第 33–40 页。

[52] M. Du 和 F. Li, "Spell：系统事件日志的流式解析", 载于 *2016* 年 *IEEE* 第 *16* 届国际数据挖掘会议（*ICDM*）. IEEE，2016 年，第 859–864 页。

[53] V.-H. Le 和 H. Zhang, "基于提示的少样本学习的日志解析", 载于 *2023* 年 *IEEE/ ACM* 第 *45* 届国际软件工程会议（*ICSE*）. IEEE，2023 年，第 2438–2449 页。

[54] J. S. Bridle, "前馈分类网络输出的概率解释，与统计模式识别的关系", 载于 《神经计算：算法、架构和应用》. Springer, 1990 年，第 227–236 页。

[55] T. Dettmers, A. Pagnoni, A. Holtzman 和 L. Zettlemoyer, "Qlora：量化 LLM 的有效微调", 《神经信息处理系统进展》, 第 36 卷，2024 年。6] I. Loshchilov, "解耦权重衰减正则化", *arXiv* 预印本 *arXiv:1711.05101*，2017 年。7] J. Zhu, S. He, P. He,
[5
[5J. Liu 和 M. R. Lyu, "Loghub：用于 AI 驱动日志分析的大型系统日志数据集", 载于 *2023* 年 *IEEE* 第 *34* 届国际软件可靠性工程研讨会（*ISSRE*）. IEEE，2023 年，第 355–366 页。