L4: Diagnosing Large-scale LLM Training Failures via Automated Log Analysis

Zhihan Jiang
The Chinese University of Hong Kong
Hong Kong SAR, China

Junjie Huang
The Chinese University of Hong Kong
Hong Kong SAR, China

Guangba Yu*
The Chinese University of Hong Kong
Hong Kong SAR, China

Zhuangbin Chen
Sun Yat-sen University
Zhuhai, China

Yichen Li
The Chinese University of Hong Kong
Hong Kong SAR, China

Renyi Zhong
The Chinese University of Hong Kong
Hong Kong SAR, China

Cong Feng
Huawei Cloud
Shenzhen, China

Yongqiang Yang
Zengyin Yang
Huawei Cloud
Shenzhen, China

Michael R. Lyu
The Chinese University of Hong Kong
Hong Kong SAR, China

ABSTRACT

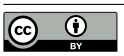
As Large Language Models (LLMs) show their capabilities across various applications, training customized LLMs has become essential for modern enterprises. However, due to the complexity of LLM training, which requires massive computational resources and extensive training time, failures are inevitable during the training process. These failures result in considerable waste of resource and time, highlighting the critical need for effective and efficient failure diagnosis to reduce the cost of LLM training.

In this paper, we present the first empirical study on the failure reports of 428 LLM training failures in our production *Platform-X* between May 2023 and April 2024. Our study reveals that hardware and user faults are the predominant root causes, and current diagnosis processes rely heavily on training logs. Unfortunately, existing log-based diagnostic methods fall short in handling LLM training logs. Considering the unique features of LLM training, we identify three distinct patterns of LLM training logs: cross-job, spatial, and temporal patterns. We then introduce our Log-based Large-scale LLM training failure diagnosis framework, L4, which can automatically extract failure-indicating information (*i.e.*, log events, nodes, stages, and iterations) from extensive training logs, thereby reducing manual effort and facilitating failure recovery. Experimental results on real-world datasets show that L4 outperforms existing approaches in identifying failure-indicating logs and localizing faulty nodes. Furthermore, L4 has been applied in *Platform-X* and demonstrated its effectiveness in enabling accurate and efficient failure diagnosis.

CCS CONCEPTS

• **Software and its engineering → Maintaining software**.

*Guangba Yu is the corresponding author.



FSE Companion '25, June 23–28, 2025, Trondheim, Norway

ACM ISBN 979-8-4007-1276-0/2025/06.
https://doi.org/10.1145/3696630.3728531

KEYWORDS

Large Language Models, Log Analysis, Failure Diagnosis, LLM Training Systems, AIOps

ACM Reference Format:
Zhihan Jiang, Junjie Huang, Guangba Yu, Zhuangbin Chen, Yichen Li, Renyi Zhong, Cong Feng, Yongqiang Yang, Zengyin Yang, and Michael R. Lyu. 2025. L4: Diagnosing Large-scale LLM Training Failures via Automated Log Analysis. In *33rd ACM International Conference on the Foundations of Software Engineering (FSE Companion '25), June 23–28, 2025, Trondheim, Norway.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3696630.3728531

1 INTRODUCTION

Large language models (LLMs) have revolutionized various fields including natural language processing [36, 78] and software engineering [28, 52], enabling breakthrough applications such as code generation [30], document translation [71], and dialogue systems [79]. The superior performance of LLMs is primarily driven by the scaling law [45], which establishes that the model capacity strongly correlates with both the model size and the volume of training data. For instance, recent models like Grok-1 [11] incorporate 314 billion parameters, while training datasets such as RedPajama [74] have reached 30 trillion tokens.

To achieve state-of-the-art model capability, significant efforts have been devoted to training or fine-tuning LLMs, which requires substantial computational resources. For example, the Llama3-405B model was trained using 16,384 H100 GPUs for 54 days [26]. To facilitate these demanding training requirements, IT enterprises have developed multi-tenant LLM development platforms, such as Amazon SageMaker [3] and Google Vertex AI [10]. These platforms allow users to submit LLM training jobs with customized hardware resources and access to specialized software libraries and tools.

LLM training failures have become the norm rather than the exception [31, 73, 75, 85, 88], primarily due to three key factors: the immense scale and complexity of computational resources, the substantial volume of training data, and the extended duration of training processes. For instance, during the training of Llama3-405B, Meta utilized 16,384 H100 GPUs and encountered 466 failures over a 54-day period [26]. These failures result in significant losses in both computational resources and time, requiring substantial human

effort for diagnosis and resolution [31, 42]. A notable example comes from BigScience's training of the BLOOM-176B model using 384 GPUs [1]. During this procedure, each hardware failure resulted in an average loss of 1.5 hours of training time, with the recovery process consuming an additional 5 to 10 hours [1, 37].

As shown in Fig. 1, failure diagnosis is a critical step in the recovery process following a training failure. Rapid and accurate diagnosis allows engineers to identify root causes, implement remediation strategies, and swiftly resume model training. However, diagnosing failures in large-scale LLM training remains a time-consuming and labor-intensive task, primarily due to the challenges posed by both node-level and cluster-level complexities. (1) **Node-level Complexity**: An AI node typically comprises several layers [81], including AI accelerators (*e.g.*, GPUs and NPUs [53]), AI toolkits (*e.g.*, CUDA [6]), AI frameworks (*e.g.*, PyTorch [16] and MindSpore [13]), and AI algorithms (*e.g.*, Transformers [18]). The intricate dependencies and interactions between these layers often result in a multitude of noisy fault manifestations, complicating accurate fault localization due to fault propagation. (2) **Cluster-level Complexity**: Training large-scale LLMs training often involves thousands of AI nodes, utilizing diverse communication paradigms such as Data Parallelism (DP), Pipeline Parallelism (PP), and Tensor Parallelism (TP) [69]. These complex structures make it challenging to quickly pinpoint faulty nodes within the vast network of interconnected components. Therefore, it is imperative to comprehensively characterize LLM training failures and explore automation opportunities to diagnose these failures.

To facilitate this need, we present an empirical study on LLM training failures and their diagnostic procedures. Our analysis examines 428 failure reports collected between May 2023 and April 2024 from *Platform-X*, a large-scale production AI platform operated by *Company-X*, a world-leading cloud vendor. The studied LLM training jobs involve models of considerable scale, with an average size of *72.8 billion parameters*, and require extensive computational resources, utilizing an average of *941 accelerators* per job. Through our study, we have obtained several valuable findings that can benefit future research on ensuring LLM training reliability. The main findings are as follows:

(1) **Failure timing**: The majority (74.1%) of failures occur during iterative model training, indicating that this core training process is prone to failures, often resulting in wasted training time and computational resources (§ 3.2).
(2) **Failure causes**: While the root causes are diverse, the primary culprits are hardware and user-side faults. Notably, hardware faults are more prevalent in LLM training compared to traditional deep learning or data processing workloads [29, 48, 89], highlighting the unique challenges of large-scale LLM training failure diagnosis (§ 3.3).
(3) **Diagnosis methods**: Training logs play a critical role in diagnosing failures, with 89.9% of cases requiring detailed manual log analysis for resolution. This underscores the importance of comprehensive log analysis in LLM training (§ 3.4).

Although Finding 3 emphasizes the importance of logs in diagnosis, an LLM training job can produce an enormous volume of raw logs (*e.g.*, serveral TBs per day), due to the extensive number of nodes and components involved [37]. Within this vast amount

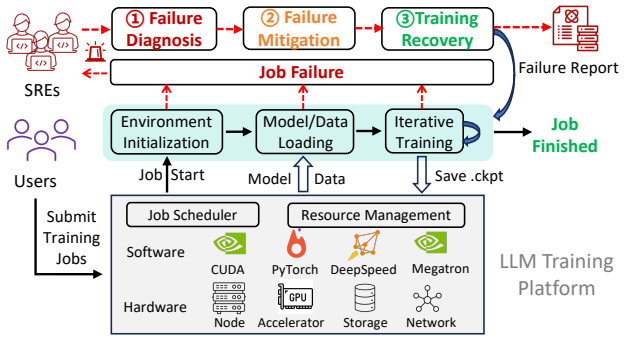


Figure 1: LLM training and failure management process in LLM development platforms.

of log data, only a small subset of logs provides actionable insights for diagnosing failures and improving the resolution efficiency, which we refer to as *failure-indicating logs*. Unfortunately, manual identification of failure-indicating logs akin to finding a needle in a haystack. While many studies [54, 55, 67, 87] have focused on detecting anomalous logs in traditional software systems, we found that existing methods struggle to accurately identify failure-indicating logs in LLM training scenarios. This limitation stems from their reliance on conventional indicators such as logging level [76], event frequency [55], and error semantic [59]. These traditional indicators often prove inapplicable for LLM failure (details in § 4.1).

To address the limitations of existing approaches, we introduce L4, a Log-based Large-scale LLM training failure diagnosis framework designed to automatically identify failure-indicating information and enhance diagnostic efficiency. L4 is designed based on three distinct patterns observed in LLM training logs, *i.e.*, *cross-job, spatial and temporal patterns*. Initially, L4 parses raw training logs into structured logs and performs cross-job filtering to eliminate noisy logs unrelated to failures. Following this, L4 leverages the spatial and temporal patterns of logs to pinpoint failure-indicating information. In the spatial dimension, L4 embeds parsed logs from each node into log event vectors and detects potential failure-indicating nodes and log events. In the temporal dimension, L4 profiles the training stage of logs and discovers distinctive log sequences to localize iterations where faults occur. Finally, these identified failure-indicting log events, nodes, stages and iterations enable engineers to efficiently and precisely understand and diagnose training failures. Furthermore, L4 allows engineers to summarize and confirm fault patterns based on this mined information, which are then archived in the fault library to match future similar failures.

We evaluated and deployed L4 on *Platform-X*. Evaluation using real-world large-scale training log datasets shows that L4 achieves high accuracy in identifying failure-indicating logs (87.3% F1-score) and detecting faulty nodes (80% top-5 accuracy). These results surpasses all compared approaches, with a large improvements ranging from 50.7% to 66.6% for log identification and 18.5% to 43.1% for node detection. In addition, L4 has been successfully applied in *Platform-X* since June 2024, where it has demonstrated effectiveness in facilitating the diagnosis of LLM training failures.

The main contributions of this paper are as follows:

- We present an empirical study on large-scale distributed LLM training failures, which offers valuable findings that can benefit future research on ensuring LLM training reliability (§ 3).
- We introduce our deployed log-based large-scale LLM training failure diagnosis framework, L4, which automatically extracts failure-indicating information (*i.e.*, log events, nodes, stages, and iterations) from extensive training logs, thereby facilitating efficient and effective failure diagnosis (§ 5).
- We evaluate L4 using real-world datasets from production LLM training jobs, demonstrating that L4 outperforms other state-of-the-art baselines. We also share our experience from over six months of industrial application of L4 on *Platform-X* (§ 6).

2 BACKGROUND

2.1 LLM Development *Platform-X*

Platform-X is a multi-tenant LLM development platform at our *Company-X*, supporting LLM training jobs for hundreds of internal users and partner companies. The platform processes hundreds of LLM training jobs daily, leveraging comprehensive hardware and software infrastructure. Specifically, *Platform-X* is equipped with substantial computing resources, including heterogeneous accelerators (*e.g.*, GPUs and NPUs), distributed storage systems, and high-performance networks (*e.g.*, RDMA over Converged Ethernet and InfiniBand). Besides, *Platform-X* provides comprehensive software support for LLM training, incorporating commonly used architecture (*e.g.*, Ascend CANN [4] and NVIDIA CUDA [6]), popular training frameworks (*e.g.*, Meagtron-LM [15] and DeepSpeed [7]), and essential libraries (*e.g.*, Pytorch [16] and Transformer [18].

The LLM training job submission and execution workflow of *Platform-X* closely resembles that of public platforms such as Amazon SageMaker [3] and Google Vertex AI [10]. As depicted in Fig. 1, when users submit an LLM training job, it first allocates the necessary resources (*e.g.*, nodes and storage) and initializes the training environment based on user requirements (*e.g.*, container images and dependent libraries). After environment initialization, datasets and models are loaded from remote storage, and the iterative training process begins (*e.g.*, fetching data, forward passing, computing loss, back-propagation, communication and saving checkpoints).

2.2 LLM Training Failure Management

Failures are frequent and can occur at any stage during the lifecycle of an LLM training job [37, 73]. When encountering a job failure, users can submit a failure report to the failure management system in *Platform-X* to seek assistance from site reliability engineers (SREs). Fig. 2 illustrated a failure report example in *Platform-X*, which mainly comprises five fields: *job metadata*, *hardware resource*, *software environment*, *failure description*, *monitoring data* and *diagnostic information*. Each field includes several detailed sub-fields to provide comprehensive descriptions of the failed training job. Particularly, *monitoring data* are uploaded by users when they seek diagnosis help. Training logs are one of the most commonly used monitoring data types, enabling SREs to gain an in-depth understanding of the job's status. Additionally, if users have enabled additional monitors (*e.g.*, performance and network monitors), the recorded data can also be uploaded to aid diagnosis.

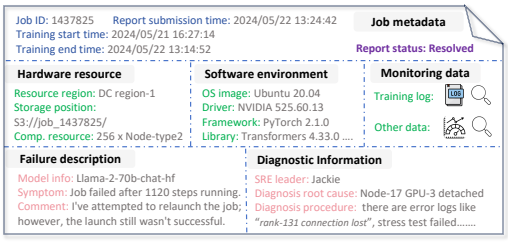


Figure 2: An example of failure reports in *Platform-X*.

Once failure reports are submitted, they are automatically assigned to appropriate SREs for handling. The SREs carefully examine the failure reports and begin the fault diagnosis process, typically involving manual inspection of monitoring data (*e.g.*, training logs). These diagnostic processes are complex and time-consuming, requiring SREs to communicate with users and other teams to identify the root cause and provide recommended solutions. After receiving feedback, users execute the suggested fixes and restart the training job. Upon successful resolution, SREs add the fault diagnostic process and root causes to the corresponding failure report and archive it within the management system, building a knowledge base for more efficient diagnosis of recurring failures.

3 LLM TRAINING FAILURE STUDY

To better characterize and understand LLM training failures and their diagnosis procedures, we conduct the first empirical study on these failures in *Platform-X*. To ensure generalizability, we avoid drawing conclusions that are specific or ambiguous. In Sec. 8.1, we provide a detailed discussion of the generalizability of our findings.

3.1 Study Design

Study Subject. We collect and study 428 failure reports of failed LLM training jobs in *Platform-X* from May 2023 to April 2024, after eliminating duplicated reports. These LLM training jobs encompass a diverse range of trained models (*e.g.*, LLaMA [26] and Vicuna [24] series), training frameworks (*e.g.*, PyTorch [16] and Transformer [18]), and underlying hardware. Furthermore, all jobs in our study were all large-scale, characterized by substantial model sizes and significant computational resource usage. Specifically, the average model size is 72.8B parameters, and the average number of accelerators utilized per job is 941.

Study Method. In this study, we comprehensively analyze all 428 LLM training failures and their diagnostic procedures by addressing the following research questions (RQs):

- **RQ1:** What are the common symptoms of LLM training failures?
- **RQ2:** What are the common root causes of LLM training failures?
- **RQ3:** What monitor data sources are typically used to diagnose LLM training failures?

We developed a taxonomy for each RQ and categorized each failure report. To avoid potential bias, a team of five experienced SREs and Ph.D. students conduct the classification process. Each annotator independently labeled the categories for three factors of each failure by thoroughly reviewing the documented diagnostic information. We used Cohen's kappa [25] to assess inter-annotator

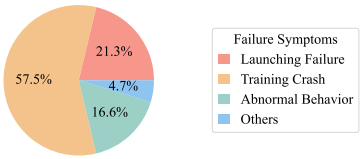


Figure 3: Classification of LLM training failure symptoms.

agreement, achieving near-perfect agreement for each taxonomy, with all scores exceeding 0.95. For cases with discrepancies, annotators engaged in discussions and, when necessary, consulted the submitters and corresponding SREs. Ultimately, consensus was reached for the categorization of all 428 failure reports.

3.2 RQ1: Failure Symptoms

We first study common symptoms of LLM training failures. A symptom is the subjective manifestation of a failure observed by users, which is manually categorized by engineers. These symptoms are divided into four categories: *launching failure*, *training crash*, *abnormal behavior* and *others*. The distribution of these failure symptoms across all 428 LLM training failures is illustrated in Fig. 3.

Failures occurring prior to the iterative training stage, such as during environment initialization, are classified as *launching failures*. These failures account for 21.3% of all reported issues, as shown in Fig.3. For instance, mismatches between the GPU driver and CUDA toolkit versions can cause failures during environment initialization, and misconfigurations in model parallelism can lead to errors during model loading. The complexity and large scale of LLM training jobs pose great challenges for users and engineers in preventing these launching failures.

After the iterative training starts, the job may crash due to various reasons such as hardware faults. These issues are frequent in LLM training due to its strong synchronization properties [37, 73, 75], *i.e.*, a local fault in a specific component (*e.g.*, a GPU or network router), can disrupt the entire training job. According to Fig. 3, these *training crash* failures represent 57.5% of all failures. Such crashes often result in waste of training time and computational resources, as they typically occur after prolonged training periods [37, 42, 85]. Even with the checkpointing mechanisms [27, 31, 73, 88], the time required for failure recovery remains substantial [31, 63].

Moreover, certain training jobs may exhibit *abnormal behaviors* like hanging or slowing down [42, 46]. For instance, an epoch might take twice as long, or training could stall at a specific iteration with no RDMA network traffic. In such cases, users can submit failure reports to SREs for helping diagnose these issues. Notably, such abnormal behaviors account for 16.6% of the total failure reports.

The final category is *Others*, which includes failures not directly related to a specific LLM training job, such as unavailability of platform and remote storage. This type of failure accounts for only 4.7% of the total failures.

> **Finding 1:** Most LLM training failures (74.1%) occur during the iterative training stage, which can waste significant computational resources and training time.

3.3 RQ2: Failure Root Causes

In this section, we study the common root causes of all 428 LLM training failures and their manifestations in monitor data, categorizing them into four categories: *hardware fault*, *user fault*, *platform fault* and *framework fault*. Due to privacy concerns, specific distribution proportions are withheld.

3.3.1 Hardware Fault. Similar to other LLM platforms, *Platform-X* is built with heterogeneous hardware, including nodes (*e.g.*, physical servers and virtual machines), accelerators (*e.g.*, GPUs, TPUs and NPUs), networks (*e.g.*, RoCE and InfiniBand), and remote distributed storage. As LLM training jobs scale increase, the required hardware resources also grow, raising the probability of hardware faults [42, 73, 85]. Due to the synchronous properties of LLM training, a single-point hardware fault can cause the training failure, making hardware fault the most common failure root cause in our study. This proportion is much higher than that reported in previous studies on data processing and deep learning failures [29, 40, 48, 60, 89], indicating that LLM training procedures are more susceptible to hardware faults.

We identified four primary sub-types of hardware faults:

Network Fault. Training LLMs demands extensive computational resources, typically involving tens to thousands of compute nodes interconnected via high-speed networks like RoCE. The training process utilizes various parallelism paradigms, which necessitate communication between compute nodes during each iteration. As a result, network issues can impact the training process, potentially causing performance degradation, hang and failures. Therefore, among different types of hardware faults, network faults are the most common cause of training failures. When such faults occur, error log messages such as "`NIC port link down`" and "`increased pcs_err_cnt`" [14] may indicate network port failures.

Accelerator Fault. Accelerators, including GPUs, TPUs, and NPUs, are the primary computing devices for LLM training. Although the fault probability for a single accelerator is low, the overall fault probability during the training procedure is high due to the large number of accelerators involved [73, 75]. Similar to traditional GPU systems [35, 70], accelerators can experience memory faults such as error correcting code (ECC) errors and stuck-at errors caused by circuit malfunctions. Power faults can also render accelerators unavailable. Common error log messages like "`double bit ecc error`" indicating ECC memory errors [8] and "`Aicore kernel execute failed`" signifying computational faults [2].

Node Fault. A node (*e.g.*, a virtual machine) is an allocated unit for training jobs, containing CPUs, memory, and other resources. In large-scale clusters, node faults such as mainboard damage, power leakage, and disk errors are inevitable and can cause training failures in *Platform-X*. When a node fails, it typically becomes inaccessible, making it impossible to retrieve logs for direct fault diagnosis. In these cases, *Platform-X* relies on heartbeat mechanisms to detect node failures. The absence of regular heartbeat signals from a node is a key indicator of a node fault.

Storage Fault. The datasets, models, and checkpoints used in LLM training can be extremely large, often exceeding hundreds of gigabytes [85]. Users typically apply for remote distributed storage and store their data there. All nodes load data from the remote storage to start training, and checkpoints are periodically generated

and stored there during the training process. Hence, any faults in remote storage can cause training failures at different stages. For example, an error log message "`Failed to load checkpoint`" [9] may indicate issues with accessing stored model states.

> **Finding 2:** LLM training procedures are vulnerable to hardware faults due to the extensive computing resources required. These faults can occur at network, accelerator, node, and storage, with network and accelerator faults being the most prevalent.

3.3.2 User Fault. Before submitting a failure report, users typically review their operations to attempt to resolve the issue themselves. Despite these efforts, *user fault* remains the second largest root cause of failures among all four categories, due to the complexity of user-side settings for LLM training jobs, including configurations, code, scripts, and more. Specifically, we identified four major subtypes of user faults in our study:

Configuration Error. Some LLM training failures are caused by misconfigurations in system environments and frameworks. When submitting LLM training jobs, users must manually configure a series of configurations. Even a minor misconfiguration can lead to training failures. For example, a user mistakenly set a low timeout threshold for Notify register, resulting in a timeout log message "`The wait execution of the Notify register times out.`" and subsequent training process failure [19].

Program/Script Bug. Similar to traditional software, buggy code can exist in LLM training programs and scripts, as comprehensively studied in previous empirical research [58, 72, 84]. For instance, using inappropriate sub-process creation during training can cause the training process to get stuck [5]. Since LLM training programs and scripts are typically adopted and modified from existing projects, most bugs occur in the modified parts, caused by inconsistencies between the original and modified code, such as error paths, null references and inconsistent model parameters.

Software Incompatibility. LLM training requires specific software such as operating system images, drivers, training frameworks, libraries, and toolkits, specified by users before submitting training jobs. Version incompatibility is common due to independent component development [29, 37], with even minor mismatches potentially causing build or compilation failures. Typically, such failures could be reflected in the logs with an inappropriate version (*e.g.*,"`Stream mode cannot be set in current driver version`" [17]) Consequently, users need to carefully verify the compatibility of the relevant software versions or utilize pre-configured version information when submitting their LLM training tasks.

Misoperation. While *Platform-X* simplifies the LLM training process, users still need to learn the operational procedures. Hence, users' misoperations can also result in LLM training failures. For example, using external remote storage for checkpoints without configuring proper access permissions can cause checkpoint writing to fail, resulting in a training crash.

> **Finding 3:** User faults constitute the second most prevalent cause of LLM training failures due to the complexity of the settings. These faults include configuration error, program/script bug, software incompatibility and misoperation.

3.3.3 Framework Fault. *Platform-X* supports various open-source LLM training frameworks and libraries, including widely used options such as PyTorch [16] and DeepSpeed [7], as well as customized frameworks like CNTK [68]. Like other software systems, these training frameworks are susceptible to various bugs. Consequently, framework faults account for a small proportion of the 428 failures we studied, often arising from buggy code and inconsistencies during software iterations. We have identified that, compared to widely used LLM training frameworks like PyTorch, customized LLM training frameworks are more prone to faults due to their relative immaturity. These framework faults are particularly challenging to diagnose because they require a deep understanding of the specific training framework and significant expert effort to locate the buggy code and logic. Moreover, fixing these bugs often requires version updates, so temporary mitigation strategies, such as version rollback, are commonly adopted until the bugs are fixed.

3.3.4 Platform Fault. *Platform-X* is a large-scale, multi-tenant platform that provides comprehensive support for LLM training. Despite careful design and iterative updates, platform-side faults are inevitable, causing the least proportion of LLM training failures. These faults arise from various system defects, with the most common type involving resource management issues, such as logical bugs in isolating abnormal nodes and mounting remote storage. Other defects can occur in modules like job scheduling (*e.g.*, abnormal preemption) and platform configurations (*e.g.*, network settings). These platform failures are highly severe and prioritized, requiring SREs to spend a significant amount of time resolving them promptly.

> **Finding 4:** Although framework and platform faults cause relatively fewer LLM training failures, their diagnosis and mitigation are more challenging. Thus, the reliability of the LLM training frameworks and platforms deserves attention.

3.4 RQ3: Data Sources of Failure Diagnosis

Troubleshooting LLM training failures is challenging due to the complexity and scale of components, stages, and resources involved. Based on the analysis of diagnostic procedures documented in 428 failure reports, **the average time to diagnose LLM training failures is 34.7 hours, with approximately 41.9% of failures requiring more than 24 hours for diagnosis.** This highlights the time-consuming and labor-intensive nature of the diagnostic process. To support failure diagnosis and better understand runtime behavior, LLM platforms, including *Platform-X*, are typically equipped with a variety of monitors that collect runtime information, such as training logs, performance metrics, and network traffic data. In this research question, we investigate the monitoring data sources typically used in the diagnosis process to better understand and improve failure diagnosis in LLM training platforms.

Similar to traditional software, training logs from different components capture detailed runtime information about the training procedure, offering valuable insights for users and engineers to understand the system's status [34, 82, 83]. Consequently, logs are a top priority for diagnosing LLM training failures in *Platform-X* in practice. We meticulously reviewed the diagnosis processes

Table 1: LLM Training Failures Across Diagnosis Types

Diagnosis Type	Log-only	Non-log	Hybrid
Percentage	53.9%	10.1%	36.0%

recorded in all 428 failure reports and categorized each training failure into three diagnostic types: **(1) Log-only diagnosable**: Only training logs are involved in the diagnosis process. **(2) Non-log diagnosable**: Training logs do not provide useful clues for diagnosis. **(3) Hybrid diagnosable**: Both training logs and other monitoring data (*e.g.*, performance metrics) are jointly used for diagnosis.

The distribution of failure diagnostic types is shown in Tab. 1. Notably, 53.9% of LLM training failures can be diagnosed using training logs alone, without additional monitoring data. This is because these logs, which include information from the training process, framework, hardware, and platform, provide comprehensive runtime details essential for diagnosing various types of faults in many cases. For instance, if there is an error log "`The ranktable or rank is invalid,Reason:[%s].`" [12], SREs can immediately notice the issues with the parallelism rank configurations and manually inspect the configuration files to determine the root causes.

However, there are also 10.1% of training failures where training logs do not aid in the diagnosis. In these cases, the logs either lack the necessary failure-indicating information or do not reflect the failure at all. Consequently, SREs cannot rely on the LLM training logs to localize the faults. Additionally, 36.0% of training failures fall into the category of hybrid diagnosable failures. In diagnosing these failures, SREs typically begin by examining the training logs to identify potential faulty components. If the logs do not provide sufficient information to pinpoint the exact faulty components and root causes, SREs must then investigate other monitoring data to aid in the diagnosis. These additional monitoring data typically include metrics of operator delay, GPU utilization rate, network packet loss, disk I/O rate and node heartbeats, which can help identify issues that are not explicitly reflected in the training logs.

In conclusion, training logs are the most crucial data source for diagnosing LLM training failures, with approximately 90% of such failures requiring the information contained within these logs for diagnosis. However, the volume of training logs can be substantial, as each distributed process rank generates logs independently, and training durations are often extensive. **The average size of training logs for the failures we studied is 16.92GB.** Consequently, manually checking and identifying the failure-indicating logs within this volume of data is time-consuming and labor-intensive. Moreover, 46.1% of training failures cannot be diagnosed solely through training logs and require supplementary system monitoring data. This finding highlights the importance of developing comprehensive monitoring systems for LLM training platforms to improve the efficiency of failure diagnosis and facilitate rapid failure recovery.

> **Finding 5:** Training logs are invaluable for diagnosing most (89.9%) LLM training failures, but their large volume underscores the need for advanced log diagnostic tools for identifying the failure-indicating logs.

4 AUTOMATION OPPORTUNITIES

Our study results show that the automated identification of failure-indicating logs from large-scale training logs is crucial to enhancing the efficiency of failure diagnosis. Therefore, in this section, we explore the automated opportunities for diagnosing LLM training failures based on training logs.

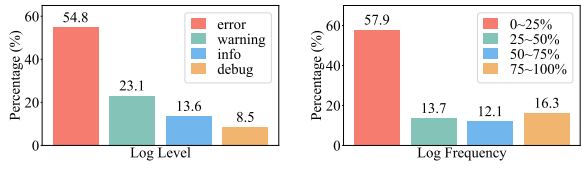
4.1 Limitation of Existing Approaches

Numerous studies [22, 39, 54, 55, 59, 67] have focused on detecting anomalous logs in software systems. These methods leverage features such as logging level [76], event frequency [55], and error semantic [47] to distinguish anomalous logs. The anomalous logs detected serve as a potential failure indicator for fault diagnosis.

We have applied existing log-based anomaly detectors to LLM training logs on *Platform-X*, but our SREs reported that these methods struggle to distinguish failure-indicating logs from unrelated ones. This issue stems from inherent limitations of the characteristics utilized by these detectors, rendering them ineffective for LLM training logs. To better understand these characteristics, we randomly sampled 100 failures and manually labeled the failure-indicating logs within their training logs according to the documented failure diagnosis procedure. Then, we analyzed the logging level, event frequency, and error semantics of these training logs.

Logging Level. Logging levels (*e.g.*, error, warning, info, and debug) indicate log importance. Traditional log analysis methods [21, 82] prioritize more serious logs, such as those at the error level. We examined the distribution of failure-indicating logs across different levels in our sample dataset, as shown in Fig. 4(a). It is evident that about half (54.8%) of these logs are at the error level. The rest are spread across all log levels, including info (13.6%) and debug (8.5%), showing that logs at various levels can provide valuable insights for failure diagnosis. Furthermore, we observed that many error-level logs are not related to training failures. These discrepancies arise because log levels, which are determined by individual LLM training component, do not always reflect the overall severity and urgency in the training process. For example, a log with failed checkpoint writing to remote storage might be logged in the error level by the checkpointing module, but if the fault-tolerance design allows for successful rewriting, this log does not impact the training process and thus is unrelated to failures.

Event Frequency. Event frequency is commonly used to detect anomalous logs, based on the intuition that infrequent log events are more likely to be anomalous [55, 77]. However, this assumption does not hold for LLM training logs. On the one hand, most infrequent logs are not failure-indicating logs. Due to the numerous stages and steps in the LLM training procedure, many logs occur infrequently or even once during the entire process. However, most of these infrequent logs are unrelated to diagnosing training failures. On the other hand, failure-indicating logs are not necessarily infrequent, especially those during the training iteration phase. We analyzed the event frequencies of failure-indicating logs in our sample dataset, as shown in Fig. 4(b). We categorized these logs into four groups based on their occurrence frequency, corresponding to the percentiles of lowest 0-25%, 25%-50%, 50%-75%, and 75%-100%. The results show that although more than half (57.9%) of the failure-indicating logs fall within the lowest 25% frequency, a notable portion still



(a) Different logging levels. (b) Different event frequencies.

Figure 4: Distributions of LLM failure-indicating logs.

occur frequently, *e.g.*, 16.3% of these logs are within the highest 25% frequency. Therefore, relying solely on frequency to identify failure-indicating logs is infeasible.

Error Semantic. Recent work [47, 49] has leveraged deep learning models, such as language models, to detect anomalies by analyzing the semantics of logs. Logs with error semantics are flagged as anomalous and potential failure indicators. However, these methods fall short in detecting failure-indicating logs in LLM training logs. Firstly, not all logs with error semantics indicate failures. Logs with error semantics from specific components or stages may not affect the training process or lead to failures. We have observed that even in some successful training jobs, there are logs with errors in building wheels or recording hardware status, which are unrelated to failures. Secondly, not all failure-indicating logs exhibit error semantics. Some training failures manifest through abnormal behaviors rather than explicit error messages, making them undetectable by current semantic-based methods.

> **Finding 6:** Existing log-based anomaly detectors struggle to effectively identify failure-indicating logs, as traditional anomalous log characteristics (*i.e.*, level, frequency and semantic) are not suitable for LLM training log scenarios. More effective approaches tailored for LLM training logs are needed.

4.2 Distinct Patterns of LLM Training Logs

Despite the inherent limitations of existing log analysis methods, we have observed three distinct patterns that can be used to automatically pinpoint failure-indicating logs.

Cross-job Pattern. In practice, each failed training job is usually associated with a series of successful jobs with identical settings (*e.g.*, models and frameworks). For example, users typically validate configurations on a small scale of nodes before scaling up. Therefore, when analyzing a failed training job's logs, it is useful to review the logs of historical successful jobs with the same settings. As discussed in Sec. 4.1, even normal training jobs can produce numerous noisy error logs. Comparing logs from successful and failed jobs can help filter out unrelated noise and identify cross-job patterns.

Spatial Pattern. Different from traditional software, the workflow of nodes in LLM training systems is highly synchronized and nearly identical. Consequently, the distributed log sequences generated by different nodes are very similar. As noted in Sec. 3.3, local faults such as hardware faults cause a significant proportion of training failures. In such cases, logs from the faulty node can exhibit different patterns compared to others. Therefore, this spatial pattern in LLM training logs allows for comparison across nodes, enabling identification of differential logs that may indicate potential failures.

Temporal Pattern. The LLM training procedure consists of multiple stages, each with distinct log characteristics. These features can help identify the stage where a failure occurred and filter out unrelated logs. For instance, if the iterative training stage has successfully started, error logs from the data and model loading stage are likely irrelevant. Furthermore, as discussed in Sec. 3.2, most LLM training failures occur during the iterative training stage, where the workflow of all nodes is periodic and identical for each iteration. This temporal pattern can be used to compare logs across different iterations to automatically identify failure-indicating logs.

> **Finding 7:** LLM training logs display special cross-job, spatial and temporal patterns, which can be leveraged to automatically identify failure-indicating logs.

5 OUR FRAMEWORK: L4

Building on these observations, we propose L4, a Log-based Large-scale LLM training failure diagnosis framework designed to automatically extract failure-indicating information from extensive training logs and improve diagnostic efficiency. The overall framework is illustrated in Fig. 5, containing four main phases.

5.1 Log Preprocessing

The raw logs of a training job are substantial (*e.g.*, tens of gigabytes) and unstructured, making them unsuitable for analysis. To address this issue, we first preprocess the unstructured training logs through log parsing, transforming them into structured log events. Specifically, log parsing aims to extract the constant parts of logs as templates and the variable parts as parameters, which has been widely studied [38, 43, 44]. This structured information facilitates the identification of identical log events with varying parameters, enabling subsequent automated log analysis. In this work, we adopt the widely-used and most efficient log parser, Drain [33], for log preprocessing. Following this parsing process, all raw logs are converted into sequential log events for further analysis.

5.2 Cross-job Filtering

As outlined in Sec. 4.2, it is common for users to submit LLM training jobs with large-scale training nodes after successfully experimenting with smaller-scale ones [84]. This practice generates a history of successful jobs that share similar settings (*e.g.*, trained models and environments) with the failed job. Intuitively, log events that are present in both successful and failed jobs are unlikely to indicate the failure root causes. Therefore, these log events can be filtered out to reduce noise and enhance the analysis efficiency.

To achieve this filtering process, we first gather all log events from the parsed logs of the historical successful jobs to construct a normal log event pool, denoted as $\mathcal{N} = \{e_{n1}, e_{n2}, \cdots\}$. Next, we examine the parsed logs of the failed job and remove all log events that are frequently present in $\mathcal{N}$ while preserving the original chronological order of the logs. This step typically results in a significant reduction in the volume of logs compared to the original unfiltered logs, which is beneficial for subsequent analysis as it reduces the amount of extraneous information. In cases where historical successful jobs are not available, we proceed by directly using the parsed logs of the failed training job for further processing.

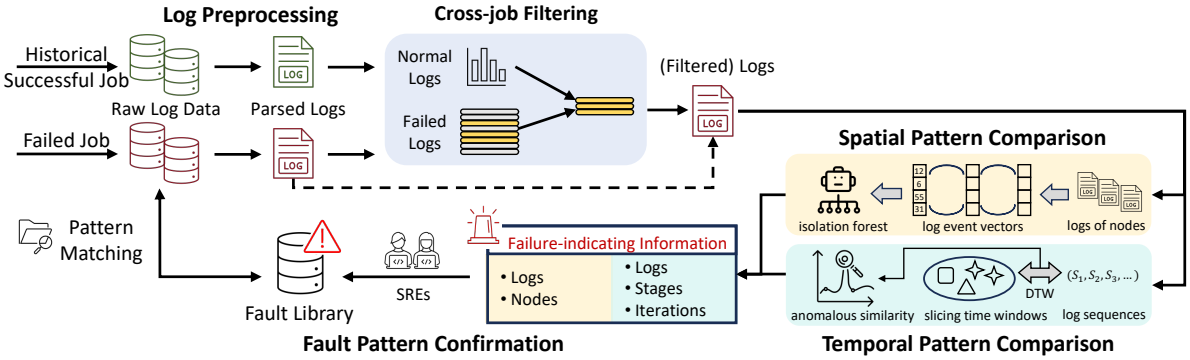


Figure 5: The overall framework of L4.

5.3 Spatial and Temporal Pattern Comparison

Inspired by the spatial and temporal pattern discussed in Sec. 4.2, this phase aims to identify failure-indicating information, such as log events, nodes, stages and iterations, by analyzing the spatial distribution and temporal sequence of training logs. These types of interpretable information can be easily understood by engineers, facilitating more effective troubleshooting and in-depth analysis.

5.3.1 Spatial pattern comparison. Since the workloads distributed across nodes during the LLM training are nearly identical, it is expected that the training logs from these nodes will display similar patterns. Therefore, examining the divergences in these patterns can reveal potential failure-indicating logs and suspicious nodes. To facilitate this analysis, we first transform the parsed log events of each node into log event vectors, following previous studies [61, 77]. Specifically, the log event vector for each node is denoted as $V = [c_1, c_2, \cdots, c_n]$, where n represents the total number of distinct log events in the training logs and c_i denotes the count of events of the i -th log event. This representation effectively captures both the occurrence and frequency information of training log events, providing an overview of training logs for each node.

After vectorization, we employ the Isolation Forest (iForest) [57] algorithm to detect deviant log event vectors across these nodes. The iForest algorithm constructs a collection of isolation trees that isolate anomalies by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature. The depth of a sample, averaged across the forest, serves as its anomaly score, with samples exhibiting noticeably shorter average path lengths being more likely to be anomalies. We select the iForest algorithm for two primary reasons: (1) It is an unsupervised method that does not require labeled data, making it highly adaptable to various training jobs. (2) It provides interpretability by indicating the anomalous degree of each log event vector, which allows us to identify not only anomalous nodes but also specific log events contributing to the anomalies. Finally, our spatial pattern comparison module utilizes the results from the iForest algorithm to pinpoint suspicious nodes and potential failure-indicating logs, which are then used for further troubleshooting.

5.3.2 Temporal pattern comparison. As discussed in Sec. 2.1, LLM training can be divided into multiple stages. These stages are typically recorded in the training logs, as LLM training frameworks often generate logs that denote the commencement and conclusion of each stage like data loading, model initialization. Consequently, in this module, L4 first employs predefined rules to categorize identified logs in earlier phases into different stages. This stage information helps SREs in broadly determining the stage at which the root causes may occur, thereby facilitating more efficient diagnosis.

Furthermore, the findings in Sec. 3.2 reveal that most LLM training failures occur during the iterative training phase, where each iteration involves executing a similar workload. Thus, log event sequences across iterations are expected to follow consistent patterns. Consequently, a log sequence that significantly deviates from those of preceding iterations could indicate a problematic iteration where a fault may occur. To identify such failure-indicating iterations, we initially transform the parsed log events from each iteration into sequences of events, denoted as $S_i = [e_1, e_2, \ldots, e_k]$. We then choose the dynamic time warping (DTW) distance [64], which can dynamically align similar patterns in log sequences, to assess the similarity between log sequences from different iterations. Subsequently, we select the slicing time window with ten iterations to calculate the average similarity scores between the log sequences of the current iteration and those of its antecedent iterations within this window. Upon computing the similarity scores for all iterations, we apply the three-sigma rule [66] to detect any anomalous scores that are significantly lower than the average. If such anomalies are detected, we identify and recommend these suspicious iterations and the logs within them as potentially failure-indicating information.

5.4 Fault Pattern Confirmation

Following the aforementioned three phases of log analysis, L4 can automatically extract potential failure-indicating information, including suspicious logs, nodes, stages, and iterations from substantial training logs. This automation circumvents the need for manual examination of extensive raw training logs, thereby significantly enhancing the diagnostic efficiency of training failures. Additionally, the recommended information offers valuable insight for SREs, enabling them to swiftly understand the behavior and status of failures and accurately pinpoint their root causes. After

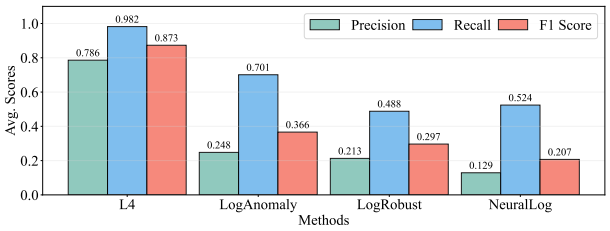


Figure 6: Effectivenss for identifying failure-indicating logs.

completing the diagnosis procedure, SREs summarize the log-based fault pattern of the failure and archive it in the historical fault library. These confirmed fault patterns can be later used to directly match the training logs of new failed jobs, thereby improving the efficiency of handling similar failures in the future.

6 EVALUATION

6.1 Experiment Designs

Evaluation Objective. We evaluate the effectiveness of L4 by answering the following two research questions (RQs):
RQ4: How effective is L4 in identifying failure-indicating logs?
RQ5: How effective is L4 in locating faulty nodes?
Dataset. Our dataset comprises log files from 100 randomly sampled failed distributed LLM training jobs in our study dataset in Sec. 3, each averaging 632 accelerators. Each accelerator corresponds to an individual process rank, thus generating a distinct log file. As a result, the average log volume of each failed job is 12.3 GB. We manually labeled the failure-indicating log events for each case based on the recorded diagnostic procedure outlined in the corresponding failure report. Specifically, within these cases, 42 were caused by hardware faults related to specific network devices, accelerators, and nodes. For these cases, we marked the machines directly associated with the faulty component as the faulty nodes.
Baselines. (1) In RQ4, we compare L4 with three state-of-the-art, open-source log anomaly detection methods: LogAnomaly [62], LogRobust [86], and NeuralLog [23]. These baseline methods rely on labeled log data for training to achieve high accuracy. Accordingly, we train the models using log data from 30% of failed LLM training jobs and test their performance on the remaining 70%. (2) In RQ5, since no existing log-based methods are designed for faulty node localization, we compare L4 with two simple baselines, *Error_time* and *Error_count*, based on the practical intuition that nodes with earlier or more frequent error logs are more likely to be faulty. In detail, *Error_time* ranks nodes by the time of their first error log, while *Error_count* ranks nodes by the total number of error logs.
Metrics. For RQ4, we compare the labeled failure-indicating logs with the detected anomalous logs, using precision, recall, and F1-score as evaluation metrics, following previous work [23, 62]. For RQ5, since L4 can rank the detected suspicious nodes based on the anomaly degree, we use top-k accuracy to assess the results, is calculated by the proportion of the labeled faulty nodes that are included in the top-k candidates.

6.2 Experiments Results

RQ4: The effectiveness of identifying failure-indicating logs. We first conduct experiments to measure the capabilities of L4 and other cutting-edge log-based anomaly detection methods in LLM training failure scenarios. Specifically, precision, recall, and F1-score are calculated by contrasting the ground-truth failure-indicating log events with the detected anomalous log events.

The experimental results are presented in Fig. 6. It is noteworthy that the existing log-based anomaly detectors exhibit poor performance when analyzing LLM training logs. Evidently, the F1-scores of all baselines are considerably lower, ranging from 0.207 to 0.366. This is primarily due to the low precision of these methods, as they fail to account for the specific patterns within distributed training logs, thereby resulting in false positive reports of anomaly logs. Furthermore, their recall scores are also inferior to that of L4, and there are two main reasons for this. First, as investigated in Sec.4.1, many failure-indicating logs do not conform to the traditional anomalous characteristics (*e.g.*, low frequency) utilized by these methods. Second, these methods rely heavily on labeled training data, and when dealing with new log data (*e.g.*, LLM training jobs with different frameworks), their accuracies are restricted.

In contrast, L4 consistently surpasses other baselines across all metrics. In particular, L4 attains a high average recall of 0.982, demonstrating its robust ability to precisely identify failure-indicating logs without omission. Although L4 's precision (0.786) is marginally lower than its recall due to the existence of noisy logs, it still remains substantially higher than that of the other baselines. Given that the detected results are intended for further examination by SREs, recall is prioritized over precision. The high F1-score of L4, exceeding baseline methods by over 0.507, further validates its superior balanced performance.
RQ5: The effectiveness of locating faulty nodes. As discussed in Sec. 3.3, hardware fault is the most common root cause of LLM training failures. L4 can unsupervisedly rank potentially faulty nodes based on their anomaly degrees through spatial pattern comparison. In this RQ, we evaluate the accuracy of L4 and two baselines in finding faulty nodes in cases of hardware-related training failures.

As shown in Fig. 7, L4 significantly outperforms two baselines across all metrics, with improvements ranging from 18.5% to 43.1%. For example, the top-1 accuracy of L4 in localizing faulty nodes is 65.8%, compared to 47.3% and 36.7% for *Error_time* and *Error_count*, respectively. This demonstrates that relying solely on error log time and count is insufficient for identifying faulty nodes. Furthermore, L4 's accuracy increases with the number of candidate nodes, reaching 91.2% when considering the top 8 candidates. These results indicate that L4 effectively detects anomalous log patterns in faulty nodes under hardware-related failure scenarios by leveraging spatial patterns. It is also worth noting that not all faulty nodes will exhibit the most anomalous log patterns. In some cases, nodes associated with the faulty node (*e.g.*, those in the same communication area) may also produce outlier log patterns. Consequently, L4 recommends multiple top-ranked anomalous nodes (the top 8 by default) that exceed an anomalous degree threshold for further investigation by SREs. In large-scale LLM training scenarios involving thousands of nodes, such recommendations greatly enhance diagnostic efficiency by narrowing the scope of investigation.

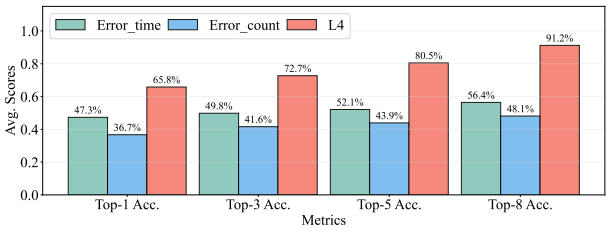


Figure 7: Effectivenss for locating faulty nodes.

7 DEPLOYMENT EXPERIENCE

L4 has been successfully applied in the failure management system of *Platform-X* to analyze the training logs of failed LLM training jobs since June 2024. Once a failure report is reported, L4 first attempts to match its training logs with historically confirmed fault patterns in the fault library. If the match is successful, the recorded root cause and fix solutions are retrieved and recommended to SREs. Otherwise, L4 automatically analyzes the extensive training logs by identifying failure-indicating log events, nodes, stages and iterations, which are then recommended to SREs for further understanding and diagnosis. This procedure significantly reduces the manual effort required to investigate substantial raw training logs. In the following, we depict two real-world cases illustrating the effectiveness of L4 in diagnosing LLM training failures.

Case 1: Fine-grained Localization of Hardware Fault. As studied in Sec. 3.3, hardware faults are the most common cause of LLM training failures. Localizing these faults to specific hardware (*e.g.*, nodes, accelerators, and network links) was previously labor-intensive and time-consuming due to the large scale of training resources. However, with L4, SREs can promptly pinpoint the faulty hardware causing training failures. For instance, an LLM training job involving 1024 nodes with 4096 accelerators on *Platform-X* failed after 20 hours of training, producing 71 GBs of latest training logs. L4 was employed to analyze these substantial logs and it identified eight nodes with log event patterns that diverged from those of the other nodes through spatial pattern comparison. Consequently, L4 recommended these suspicious nodes and log events such as `'ROCE(,hccp_service.bin):error cqe status.'`. Based on this information, SREs inspected the indicated nodes and logs, uncovering a hardware fault in an accelerator on one of these nodes. This fault also caused the nodes directly communicating with the faulty node to generate relevant error logs. Following this discovery, the SREs isolated and replaced the faulty node, restored the training procedure, and subsequently summarized this fault pattern into the fault library for diagnosing similar failures in the future.

Case 2: Issue Identification during Iterative Training. According to Finding 1, most LLM training failures occur during training iterations. L4 can effectively identify the failure-indicating iteration to aid in diagnosis. For example, in *Platform-X*, an LLM training job hanged after around two thousand epochs. L4 was applied to analyze the training logs, and through temporal pattern comparison, L4 identified an anomalous log event sequence in one iteration. This sequence included additional logs, such as `'notify wait from rank_<*> timeout'`, which were present in this iteration but absent in preceding ones. This failure-indicating information inspired SREs to suspect a network fault, prompting them to check the network packet loss rate for this ranked node. They found a spike during the time frame of the anomalous iteration, confirming intermittent network faults in the communication links, which caused timeouts and stalled the training. Consequently, SREs isolated and repaired the faulty network links for failure recovery, incorporating this fault pattern into the fault library for future failure diagnosis.

These two typical cases demonstrate that L4 can effectively contribute to diagnosing the majority of LLM training failures. However, there remains a small portion of cases where L4 cannot directly pinpoint the exact root cause. For instance, in rare cases involving logic bugs within LLM training frameworks, L4 is unable to locate the specific faulty code. Nonetheless, in such situations, L4 can still provide valuable diagnostic information (*e.g.*, the failed stage) and filter out failure-unrelated noisy logs. This process reduces the manual effort required to analyze training logs and enhances the diagnostic efficiency of LLM training failures for SREs.

8 DISCUSSION

8.1 Generalizability

Our study investigates 428 diverse LLM training job failures on the *Platform-X*. However, we believe that the studied LLM failures are common and representative, and that the findings can be generalized to other LLM platforms.

On the one hand, *Platform-X* employs widely adopted technologies and shares architectural similarities with other leading platforms [41, 42, 46], which utilize similar job management mechanisms. Besides, most training jobs on *Platform-X* involve diverse open-source and commonly used models (*e.g.*, LLaMA [26] and Vicuna [24] series), frameworks (*e.g.*, PyTorch [16] and Transformer [18]) and hardware, underscoring the commonality of our analysis.

On the other hand, our study avoids drawing conclusions that are overly specific or narrowly applicable to *Platform-X*. In addition, reports from industry practitioners have identified similar issues in large-scale LLM training systems. For instance, Kokolis et al. [46] classify LLM training failures into domains such as user programs, system software, and hardware infrastructure, aligning closely with our observations. Similarly, Hu et al. [37] and Jiang et al. [42] highlight the diagnostic challenges of LLM training failures and underscore the critical role of training logs in the diagnosis process. Our study provides a more comprehensive analysis of failure root causes and diagnostic procedures, along with the discussion of automation opportunities, offering actionable insights for other LLM platforms to improve system reliability.

Regarding our proposed framework, L4, its applicability extends beyond *Platform-X* and can enhance failure diagnosis efficiency across different LLM platforms. The three key log patterns leveraged by L4 —cross-job, spatial, and temporal—are not specific to *Platform-X* and are broadly applicable to various LLM training scenarios. The successful deployment of L4 on a large number of LLM training jobs involving diverse models, frameworks, and hardware configurations further demonstrates its generalizability.

8.2 Future Directions

LLM Training Monitoring. According to Finding 5, 10.1% of failures could not be diagnosed using the current platform's monitoring

data and logs, indicating that existing monitoring systems for LLM training are insufficient and could be improved. Future enhancements could incorporate additional signals like training data lineage and program profiling to offer a more holistic view of the training process. Tracking the provenance of training data and analyzing the performance characteristics of training programs could enhance the platform's monitoring capabilities, enabling earlier detection and diagnosis of failures. Additionally, our research highlighted issues with the quality of LLM training logs, characterized by excessive noise and low correlation between logging levels/semantics and failure relevance. Future work can focus on providing recommendations on logging levels, locations, and contents to improve log standardization and informativeness in LLM training frameworks, ensuring that logged information is more closely tied to potential failures and facilitating efficient failure diagnosis [50, 51].

Multi-modal Failure Diagnosis. Our L4 is designed to extract failure-indicating information from extensive training logs, enhancing the efficiency of diagnosing LLM training failures. However, analyzing logs solely cannot handle all failure types. As shown in Sec. 3.4, 36.0% of failures require hybrid monitoring data for accurate diagnosis. Future research could further integrate multi-modal monitoring data, similar to cloud system failure diagnosis [32, 80, 90]. Combining diverse data modalities such as logs, performance metrics, and network profiling data can provide a comprehensive view of the training process, allowing for a more accurate and comprehensive failure diagnosis.

9 RELATED WORK

Model Training Failure Study. Recently, a series of studies have focused on failures in deep learning (DL) platforms. [29] studied 360 quality issues of DL jobs, categorizing common symptoms, root causes and fixes, while [84] focused on program failures and reviewed current testing and debugging practices in DL platforms. [35] presented an in-depth study on hardware faults of accelerators in DL systems. However, as previous DL jobs differ significantly from LLM training jobs in model sizes, architectures, training resources, and software stacks [37], new research on large-scale LLM training jobs is necessary to uncover their unique characteristics and associated failures.

The closest related work is by [37], which examined a six-month LLM development workload and probed discrepancies between LLMs and prior DL workloads. However, their analysis did not focus on training failures and diagnosis procedures. In contrast, our work comprehensively examines failure symptoms and root causes, as well as automated opportunities for log-based failure diagnosis, offering valuable insights for both practitioners and researchers.

Log-based Failure Diagnosis. Logs are essential for ensuring software reliability and diagnosing issues by providing critical runtime information [20, 54, 83]. A significant body of research focuses on diagnosing by identifying anomalous logs for further investigation. For instance, [77] pioneered the application of Principal Component Analysis (PCA) to detect system issues from console logs based on log event frequencies. LogCluster [56] employs clustering to group similar logs, thereby identifying atypical log events. SBLD [67] applied spectrum-based techniques to transform logs into events and identify key logs by evaluating event coverage. Moreover, some supervised approaches, such as LogRobust[86], leverages log semantics for more effective analysis. Recently, LLMs have been applied to analyze log semantics, enhancing the identification of anomalous logs with error semantics for diagnosis [59, 65].

However, as outlined in Sec. 4.1, these methods assume certain characteristics of anomalous logs that are inapplicable to the context of LLM training logs. In contrast, our L4 utilizes the distinct patterns in LLM training logs to accurately identify failure-indicating information. Our study can also provide guidance and opportunities for future log-based LLM training failure diagnosis research.

10 CONCLUSION

In conclusion, this work provides the first comprehensive study of LLM training failures on *Platform-X*. Our investigation reveals diverse root causes and diagnostic challenges of these failures, with existing log analysis methods being inadequate due to their reliance on traditional log characteristics. We also identify three distinct patterns (*i.e.*, cross-job, spatial and temporal) within LLM training logs, which inspire the design of our L4, a log-based large-scale LLM training failure diagnosis framework. L4 significantly enhances diagnostic efficiency and accuracy by automatically extracting failure-indicating information from extensive training logs. Our findings and L4 can offer valuable information for diagnosing LLM training failures and ensuring the reliability of LLM training systems.

11 ACKNOWLEDGMENT

The work described in this paper was supported by the Research Grants Council of the Hong Kong Special Administrative Region, China (No. SRFS2425-4S03 of the Senior Research Fellow Scheme and No. CUHK 14209124 of the General Research Fund). We are also grateful to Rui Ren, Dong Wang, and Hongliang Xiang, for their invaluable help at various stages of this project, as well as the support of HUAWEI CLOUD Enterprise Intelligence.

REFERENCES

[1] 2022. The Technology Behind BLOOM Training. https://huggingface.co/blog/bloom-megatron-deepspeed
[2] 2024. AI Core failure. https://www.hiascend.com/document/detail/zh/canncommercial/80RC3/developmentguide/maintenref/troubleshooting/troubleshooting_0004.html
[3] 2024. Amazon SageMaker. https://aws.amazon.com/sagemaker/
[4] 2024. CANN Toolkit. https://www.hiascend.com/en/software/cann
[5] 2024. Creating a child process in the fork method causes the application process to get stuck. https://www.hiascend.com/document/detail/zh/canncommercial/80RC3/developmentguide/maintenref/troubleshooting/troubleshooting_0075.html
[6] 2024. CUDA Toolkit. https://developer.nvidia.com/cuda-toolkit
[7] 2024. DeepSpeed. https://www.deepspeed.ai/
[8] 2024. ECC failure. https://support.huaweicloud.com/trouble-ecs/ecs_trouble_1626.html
[9] 2024. Failed to load checkpoint after write failure to S3 backend. https://github.com/pulumi/pulumi/issues/2801
[10] 2024. Google Vertex AI. https://console.cloud.google.com/vertex-ai?hl=en&inv=1&invt=Abkx0g&project=fine-effect-362306
[11] 2024. Grok-1. https://github.com/xai-org/grok-1
[12] 2024. Invalid Ranktable Configuration. https://www.hiascend.com/document/detail/zh/canncommercial/80RC3/developmentguide/maintenref/troubleshooting/atlaserrorcode_15_0244.html
[13] 2024. MindSpore LLM Platform. https://xihe.mindspore.cn/en
[14] 2024. NPU network port Link failure. https://www.hiascend.com/document/caselibrary/detail/topic_0000001792986414
[15] 2024. Ongoing research training transformer models at scale. https://github.com/NVIDIA/Megatron-LM
[16] 2024. PyTorch. https://pytorch.org

[17] 2024. Stream mode cannot be set in current driver version. https://www.hiascend.com/document/caselibrary/detail/case_9542

[18] 2024. Transformers. https://huggingface.co/docs/transformers/en/index

[19] 2024. The wait execution of the Notify register times out. https://www.hiascend.com/document/caselibrary/detail/case_9526

[20] Anunay Amar and Peter C Rigby. 2019. Mining historical test logs to predict bugs and localize faults in the test logs. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 140–151.

[21] Jasmin Bogatinovski, Gjorgji Madjarov, Sasho Nedelkoski, Jorge Cardoso, and Odej Kao. 2022. Leveraging log instructions in log-based anomaly detection. In *2022 IEEE International Conference on Services Computing (SCC)*. IEEE, 321–326.

[22] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM computing surveys (CSUR)* 41, 3 (2009), 1–58.

[23] Zeming Chen, Qiyue Gao, and Lawrence S Moss. 2021. NeuralLog: Natural language inference with joint neural and logical reasoning. *arXiv preprint arXiv:2105.14167* (2021).

[24] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)* 2, 3 (2023), 6.

[25] Jacob Cohen. 1968. Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit. *Psychological bulletin* 70, 4 (1968), 213.

[26] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).

[27] Assaf Eisenman, Kiran Kumar Matam, Steven Ingram, Dheevatsa Mudigere, Raghuraman Krishnamoorthi, Krishnakumar Nair, Misha Smelyanskiy, and Murali Annavaram. 2022. {Check-N-Run}: A checkpointing system for training deep learning recommendation models. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 929–943.

[28] Shuzheng Gao, Xin-Cheng Wen, Cuiyun Gao, Wenxuan Wang, Hongyu Zhang, and Michael R Lyu. 2023. What makes good in-context demonstrations for code intelligence tasks with llms?. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 761–773.

[29] Yanjie Gao, Xiaoxiang Shi, Haoxiang Lin, Hongyu Zhang, Hao Wu, Rui Li, and Mao Yang. 2023. An Empirical Study on Quality Issues of Deep Learning Platform. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 455–466.

[30] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. 2024. DeepSeek-Coder: When the Large Language Model Meets Programming–The Rise of Code Intelligence. *arXiv preprint arXiv:2401.14196* (2024).

[31] Tanmaey Gupta, Sanjeev Krishnan, Rituraj Kumar, Abhishek Vijeev, Bhargav Gulavani, Nipun Kwatra, Ramachandran Ramjee, and Muthian Sivathanu. 2024. Just-In-Time Checkpointing: Low Cost Error Recovery from Deep Learning Training Failures. In *Proceedings of the Nineteenth European Conference on Computer Systems*. 1110–1125.

[32] Jingzhu He, Yuhang Lin, Xiaohui Gu, Chin-Chia Michael Yeh, and Zhongfang Zhuang. 2022. Perfsig: extracting performance bug signatures via multi-modality causal analysis. In *Proceedings of the 44th International Conference on Software Engineering*. 1669–1680.

[33] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. 2017. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE international conference on web services (ICWS)*. IEEE, 33–40.

[34] Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R Lyu. 2021. A survey on automated log analysis for reliability engineering. *ACM computing surveys (CSUR)* 54, 6 (2021), 1–37.

[35] Yi He, Mike Hutton, Steven Chan, Robert De Gruijl, Rama Govindaraju, Nishant Patil, and Yanjing Li. 2023. Understanding and mitigating hardware failures in deep learning training systems. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–16.

[36] Zhankui He, Zhouhang Xie, Rahul Jha, Harald Steck, Dawen Liang, Yesu Feng, Bodhisattwa Prasad Majumder, Nathan Kallus, and Julian McAuley. 2023. Large language models as zero-shot conversational recommenders. In *Proceedings of the 32nd ACM international conference on information and knowledge management*. 720–730.

[37] Qinghao Hu, Zhisheng Ye, Zerui Wang, Guoteng Wang, Meng Zhang, Qiaoling Chen, Peng Sun, Dahua Lin, Xiaolin Wang, Yingwei Luo, et al. 2024. Characterization of large language model development in the datacenter. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 709–729.

[38] Junjie Huang, Zhihan Jiang, Zhuangbin Chen, and Michael R Lyu. 2024. LUNAR: Unsupervised LLM-based Log Parsing. *arXiv preprint arXiv:2406.07174* (2024).

[39] Junjie Huang, Zhihan Jiang, Jinyang Liu, Yintong Huo, Jiazhen Gu, Zhuangbin Chen, Cong Feng, Hui Dong, Zengyin Yang, and Michael R Lyu. 2024. Demystifying and Extracting Fault-indicating Information from Logs for Failure Diagnosis. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 511–522.

[40] Junjie Huang, Jinyang Liu, Zhuangbin Chen, Zhihan Jiang, Yichen Li, Jiazhen Gu, Cong Feng, Zengyin Yang, Yongqiang Yang, and Michael R Lyu. 2024. Faultprofit: Hierarchical fault profiling of incident tickets in large-scale cloud systems. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*. 392–404.

[41] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. 2019. Analysis of {Large-Scale}{Multi-Tenant}{GPU} clusters for {DNN} training workloads. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. 947–960.

[42] Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, et al. 2024. {MegaScale}: Scaling Large Language Model Training to More Than 10,000 {GPUs}. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 745–760.

[43] Zhihan Jiang, Jinyang Liu, Zhuangbin Chen, Yichen Li, Junjie Huang, Yintong Huo, Pinjia He, Jiazhen Gu, and Michael R Lyu. 2024. Lilac: Log parsing using llms with adaptive parsing cache. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 137–160.

[44] Zhihan Jiang, Jinyang Liu, Junjie Huang, Yichen Li, Yintong Huo, Jiazhen Gu, Zhuangbin Chen, Jieming Zhu, and Michael R Lyu. 2024. A Large-Scale Evaluation for Log Parsing Techniques: How Far Are We?. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*.

[45] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361* (2020).

[46] Apostolos Kokolis, Michael Kuchnik, John Hoffman, Adithya Kumar, Parth Malani, Faye Ma, Zachary DeVito, Shubho Sengupta, Kalyan Saladi, and Carole-Jean Wu. 2024. Revisiting Reliability in Large-Scale Machine Learning Research Clusters. *arXiv preprint arXiv:2410.21680* (2024).

[47] Van-Hoang Le and Hongyu Zhang. 2022. Log-based anomaly detection with deep learning: How far are we?. In *Proceedings of the 44th international conference on software engineering*. 1356–1367.

[48] Sihan Li, Hucheng Zhou, Haoxiang Lin, Tian Xiao, Haibo Lin, Wei Lin, and Tao Xie. 2013. A characteristic study on failures of production distributed data-parallel programs. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 963–972.

[49] Xiaoyun Li, Pengfei Chen, Linxiao Jing, Zilong He, and Guangba Yu. 2020. SwissLog: Robust and Unified Deep Learning Based Log Anomaly Detection for Diverse Faults. In *Proceedings of the31st International Symposium on Software Reliability Engineering*. 92–103.

[50] Yichen Li, Yintong Huo, Zhihan Jiang, Renyi Zhong, Pinjia He, Yuxin Su, and Michael R Lyu. 2023. Exploring the effectiveness of llms in automated logging generation: An empirical study. *arXiv preprint arXiv:2307.05950* (2023).

[51] Yichen Li, Yintong Huo, Renyi Zhong, Zhihan Jiang, Jinyang Liu, Junjie Huang, Jiazhen Gu, Pinjia He, and Michael R Lyu. 2024. Go Static: Contextualized Logging Statement Generation. *arXiv preprint arXiv:2402.12958* (2024).

[52] Yichen Li, Yulun Wu, Jinyang Liu, Zhihan Jiang, Zhuangbin Chen, Guangba Yu, and Michael Lyu. 2025. COCA: Generative Root Cause Analysis for Distributed Systems with Code Knowledge. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 770–770.

[53] Heng Liao, Jiajin Tu, Jing Xia, Hu Liu, Xiping Zhou, Honghui Yuan, and Yuxing Hu. 2021. Ascend: a scalable and unified architecture for ubiquitous deep neural network computing: Industry track paper. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 789–801.

[54] Fred Lin, Keyur Muzumdar, Nikolay Pavlovich Laptev, Mihai-Valentin Curelea, Seunghak Lee, and Sriram Sankar. 2020. Fast dimensional analysis for root cause investigation in a large-scale service environment. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 4, 2 (2020), 1–23.

[55] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. 2016. Log clustering based problem identification for online service systems. In *Proceedings of the 38th International Conference on Software Engineering Companion*. 102–111.

[56] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. 2016. Log clustering based problem identification for online service systems. In *Proceedings of the 38th International Conference on Software Engineering Companion*. 102–111.

[57] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation forest. In *2008 eighth ieee international conference on data mining*. IEEE, 413–422.

[58] Haopeng Liu, Shan Lu, Madan Musuvathi, and Suman Nath. 2019. What bugs cause production cloud incidents?. In *Proceedings of the Workshop on Hot Topics in Operating Systems*. 155–162.

[59] Jinyang Liu, Junjie Huang, Yintong Huo, Zhihan Jiang, Jiazhen Gu, Zhuangbin Chen, Cong Feng, Minzhi Yan, and Michael R Lyu. 2023. Scalable and adaptive log-based anomaly detection with expert in the loop. *arXiv preprint arXiv:2306.05032* (2023).

[60] Jinyang Liu, Zhihan Jiang, Jiazhen Gu, Junjie Huang, Zhuangbin Chen, Cong Feng, Zengyin Yang, Yongqiang Yang, and Michael R Lyu. 2023. Prism: Revealing hidden functional clusters from massive instances in cloud systems. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*.

IEEE, 268–280.

[61] Jian-Guang Lou, Qiang Fu, Shenqi Yang, Ye Xu, and Jiang Li. 2010. Mining invariants from console logs for system problem detection. In *2010 USENIX Annual Technical Conference (USENIX ATC 10)*.

[62] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, et al. 2019. Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs.. In *IJCAI*, Vol. 19. 4739–4745.

[63] Jayashree Mohan, Amar Phanishayee, and Vijay Chidambaram. 2021. CheckFreq: Frequent, Fine-Grained DNN Checkpointing. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*. 203–216.

[64] Meinard Müller. 2007. Dynamic time warping. *Information retrieval for music and motion* (2007), 69–84.

[65] Jonathan Pan, Swee Liang Wong, and Yidi Yuan. 2023. RAGLog: Log Anomaly Detection using Retrieval Augmented Generation. *arXiv preprint arXiv:2311.05261* (2023).

[66] Friedrich Pukelsheim. 1994. The three sigma rule. *The American Statistician* 48, 2 (1994), 88–91.

[67] Carl Martin Rosenberg and Leon Moonen. 2020. Spectrum-based log diagnosis. In *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 1–12.

[68] Frank Seide and Amit Agarwal. 2016. CNTK: Microsoft's open-source deep-learning toolkit. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2135–2135.

[69] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053* (2019).

[70] Devesh Tiwari, Saurabh Gupta, James Rogers, Don Maxwell, Paolo Rech, Sudharshan Vazhkudai, Daniel Oliveira, Dave Londo, Nathan DeBardeleben, Philippe Navaux, et al. 2015. Understanding GPU errors on large-scale HPC systems and the implications for system design and operation. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 331–342.

[71] Longyue Wang, Chenyang Lyu, Tianbo Ji, Zhirui Zhang, Dian Yu, Shuming Shi, and Zhaopeng Tu. 2023. Document-level machine translation with large language models. *arXiv preprint arXiv:2304.02210* (2023).

[72] Tao Wang, Qingxin Xu, Xiaoning Chang, Wensheng Dou, Jiaxin Zhu, Jinhui Xie, Yuetang Deng, Jianbo Yang, Jiaheng Yang, Jun Wei, et al. 2022. Characterizing and detecting bugs in WeChat mini-programs. In *Proceedings of the 44th International Conference on Software Engineering*. 363–375.

[73] Zhuang Wang, Zhen Jia, Shuai Zheng, Zhen Zhang, Xinwei Fu, TS Eugene Ng, and Yida Wang. 2023. Gemini: Fast failure recovery in distributed training with in-memory checkpoints. In *Proceedings of the 29th Symposium on Operating Systems Principles*. 364–381.

[74] Maurice Weber, Daniel Y. Fu, Quentin Anthony, Yonatan Oren, Shane Adams, Anton Alexandrov, Xiaozhong Lyu, Huu Nguyen, Xiaozhe Yao, Virginia Adams, Ben Athiwaratkun, Rahul Chalamala, Kezhen Chen, Max Ryabinin, Tri Dao, Percy Liang, Christopher Ré, Irina Rish, and Ce Zhang. 2024. RedPajama: an Open Dataset for Training Large Language Models. *NeurIPS Datasets and Benchmarks Track* (2024).

[75] Baodong Wu, Lei Xia, Qingping Li, Kangyu Li, Xu Chen, Yongqiang Guo, Tieyao Xiang, Yuheng Chen, and Shigang Li. 2023. Transom: An efficient fault-tolerant system for training llms. *arXiv preprint arXiv:2310.10046* (2023).

[76] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. 2009. Largescale system problem detection by mining console logs. In *Proceedings of SOSP*, Vol. 9. Citeseer, 1–17.

[77] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. 2009. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles (SOSP)*. 117–132.

[78] Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Shaochen Zhong, Bing Yin, and Xia Hu. 2024. Harnessing the power of llms in practice: A survey on chatgpt and beyond. *ACM Transactions on Knowledge Discovery from Data* 18, 6 (2024), 1–32.

[79] Zihao Yi, Jiarui Ouyang, Yuwen Liu, Tianhao Liao, Zhe Xu, and Ying Shen. 2024. A Survey on Recent Advances in LLM-Based Multi-turn Dialogue Systems. *arXiv preprint arXiv:2402.18013* (2024).

[80] Guangba Yu, Pengfei Chen, Yufeng Li, Hongyang Chen, Xiaoyun Li, and Zibin Zheng. 2023. Nezha: Interpretable Fine-Grained Root Causes Analysis for Microservices on Multi-modal Observability Data. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 553–565.

[81] Guangba Yu, Gou Tan, Haojia Huang, Zhenyu Zhang, Pengfei Chen, Roberto Natella, and Zibin Zheng. 2024. A Survey on Failure Analysis and Fault Injection in AI Systems. arXiv:2407.00125 [cs.SE] https://arxiv.org/abs/2407.00125

[82] Ding Yuan, Soyeon Park, Peng Huang, Yang Liu, Michael M Lee, Xiaoming Tang, Yuanyuan Zhou, and Stefan Savage. 2012. Be conservative: Enhancing failure diagnosis with proactive logging. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. 293–306.

[83] Wei Yuan, Shan Lu, Hailong Sun, and Xudong Liu. 2020. How are distributed bugs diagnosed and fixed through system logs? *Information and Software Technology* 119 (2020), 106234.

[84] Ru Zhang, Wencong Xiao, Hongyu Zhang, Yu Liu, Haoxiang Lin, and Mao Yang. 2020. An empirical study on program failures of deep learning jobs. In *Proceedings of the ACM/IEEE 42nd international conference on software engineering*. 1159–1170.

[85] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2023. Opt: Open pre-trained transformer language models, 2022. *URL https://arxiv.org/abs/2205.01068* 3 (2023), 19–0.

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

[87] Xu Zhang, Yong Xu, Si Qin, Shilin He, Bo Qiao, Ze Li, Hongyu Zhang, Xukun Li, Yingnong Dang, Qingwei Lin, et al. 2021. Onion: identifying incident-indicating logs for cloud systems. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1253–1263.

[88] Yuchen Zhong, Guangming Sheng, Juncheng Liu, Jinhui Yuan, and Chuan Wu. 2023. Swift: Expedited Failure Recovery for Large-Scale DNN Training. In *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*. 447–449.

[89] Hucheng Zhou, Jian-Guang Lou, Hongyu Zhang, Haibo Lin, Haoxiang Lin, and Tingting Qin. 2015. An empirical study on quality issues of production big data platform. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 2. IEEE, 17–26.

[90] Zhouruixing Zhu, Cheryl Lee, Xiaoying Tang, and Pinjia He. 2024. HeMiRCA: Fine-Grained Root Cause Analysis for Microservices with Heterogeneous Data Sources. *ACM Transactions on Software Engineering and Methodology* (2024).