Bifrost: Empowering Pretrained Language Model with Fallibility Representation for Log-based Fault Diagnosis

Anonymous Author(s)

Abstract

Log-based fault diagnosis is crucial for maintaining the reliability of web systems. Existing fault diagnosis methods use language models pre-trained on natural language (PLMs) for log representation. However, system faults are reflected in the multi-level structure of system logs. PLMs pre-trained on natural language struggle to comprehensively capture multi-level fault information, failing to meet the requirements of fault diagnosis. We refer to this information as fallibility representations. To address this problem, we propose a novel log representation learning method, Bifrost. It draws inspiration from the log analysis experience of Site Reliability Engineers and meticulously designs strategies based on self-supervised contrastive learning to learn the fallibility representations of logs. A comprehensive evaluation on three public systems and one industrial ML-as-a-Service system shows that the log representations produced by Bifrost achieve an average advantage of 9.83%, 18.28%, and 20.88% over existing PLMs.

CCS Concepts

• **Information systems → Web log analysis**.

Keywords

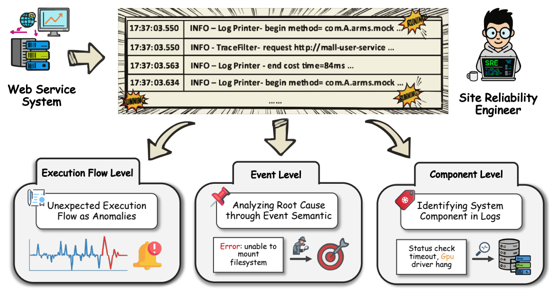
Web System Reliability, Fault Diagnosis, Log Analysis, Root Cause Localization, Log Representation

ACM Reference Format:
Anonymous Author(s). 2018. Bifrost: Empowering Pretrained Language Model with Fallibility Representation for Log-based Fault Diagnosis. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 12 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

"Remember the two audiences for logs: people and programs. Logs must serve both human readability and machine parsability."
— Observability Engineering, 2022

Web systems are becoming increasingly large and complex, with their composition and operational logic growing in intricacy, and they are affected by a greater number of faults [18, 19, 23]. For instance, a one-hour outage for Amazon on Prime Day could potentially lead to a sales loss of up to 100 million USD [35]. In a real-world context where availability is crucial for customer experience, web service providers must devise methods to rapidly and effectively manage anomalous events to enhance web system reliability. Fault diagnosis is a primary approach to achieving this [10, 31, 38].


Web Service System
RUNING
RUNNING
Site Reliability Engineer
SRE
Execution Flow Level
Unexpected Execution Flow as Anomalies
Event Level
Analyzing Root Cause through Event Semantic
Error: unable to mount filesystem
Component Level
Identifying System Component in Logs
Status check timeout, Gpu driver hang

```
17:37:03.550   INFO – Log Printer- begin method= com.A.arms.mock ...
17:37:03.550   INFO - TraceFilter- request http://mall-user-service ...
17:37:03.563   INFO – Log Printer - end cost time=84ms ...
17:37:03.634   INFO – Log Printer- begin method= com.A.arms.mock ...
                              ......
```

Figure 1: Practical log analysis experience of SREs.

Logs record the operational status of a system and are a vital source for diagnosing faults in web systems. In recent years, log-based fault diagnosis for web systems has been extensively studied [9, 26, 33, 37, 41]. Existing log-based fault diagnosis methods are typically divided into three main stages. 1) Log parsing [8, 9, 28], which involves converting semi-structured text logs into structured log events. 2) Log representation [7, 14, 20], where the parsed log events are transformed into numerical vectors; recent research predominantly utilizes Pre-trained Language Models (PLMs) for this purpose. 3) Fault diagnosis [4, 6, 15], a stage that involves constructing specific models to diagnose observed system logs, encompassing three main tasks: Anomaly Detection (AD) [12, 34], Root Cause Analysis (RCA) [1, 30], and Fault Identification (FI) [4, 27]. In this paper, we primarily focus on the second stage: log representation.

Although existing PLM-based log representation methods have achieved some success, PLMs failing to meet the requirements of fault diagnosis tasks [7, 14, 20]. LLMeLog [7] conducted an empirical study which proved that log representations encoded by PLMs suffer from issues such as content missing, semantic deflection, and tendency lacks, leading to a 19.97% decrease in anomaly detection. KnowLog [20] pointed out that PLMs have significant deficiencies in understanding the abbreviations, context, and style of logs. PreLog [14] analyzed the inadequacy of PLMs in understanding log evolution. Overall, leveraging the full potential of PLMs for web system fault diagnosis still faces significant challenges.

In our view, the root cause of these issues is the inherent structural differences between logs and natural language. The characteristics of system faults do not exist in a singular semantic form but are manifested through the multi-level structure of logs. Therefore, relying solely on models pre-trained on natural language makes it difficult to comprehensively capture and understand this complex structural information, thus impeding effective fault diagnosis.

In fact, in real-world web system reliability maintenance practices, Site Reliability Engineers (SREs) typically analyze logs from three levels: execution flow, event, and component, to gain insights into system faults, as shown in Figure 1. 1) Execution Flow Level: SREs focus on scrutinizing execution flows that deviate from historical patterns. Such unexpected execution paths are considered strong indicators of nomalies [3, 22]. 2) Event Level. In production practice, SREs abstract a single log entry into an independent system event. The specific semantics carried by each event serve as the key basis for them to determine the cause of a fault [7]. 3) Component Level: Log records contain fine-grained information about system components (e.g., GPU, disk). By deeply analyzing this information, SREs can localize the fault to specific hardware or software units [4].

We term the information that SREs obtain from the multi-level analysis of logs as fallibility representations. PLMs, operating in the form of masked language modeling, can only encode the natural language information of system logs while neglecting the fault-related fallibility representations, leading to poor performance of the encoded log representations in various fault diagnosis tasks. To investigate the capability of PLMs in encoding fallibility representations, this paper conducts an empirical study on existing PLMs using three log datasets. The results indicate that existing PLMs are indeed incapable of encoding these fallibility representations, which leads to unsatisfactory fault diagnosis performance.

Solving this problem requires a comprehensive learning of the fallibility representations of system logs. However, this is not straightforward. Firstly, fallibility cannot be directly obtained from system logs. How can learning objectives be designed to effectively learn different types of fallibility representations? Secondly, the learning processes for different types of fallibility representations may interfere with each other. How can a learning strategy be designed to comprehensively learn fallibility representations?

To address these issues and unlock the potential of PLMs in fault diagnosis tasks, in this paper, we propose *Bifrost*, an innovative log representation learning method designed to enhance the capabilities of PLMs in fault diagnosis by learning fallibility representations from the continuously generated system logs of web systems. To tackle the first problem, *Bifrost* first meticulously designs three training tasks: Execution Flow Prediction (EFP), Abnormal Event Discrimination (AED), and System Component Perception (SCP). These tasks, based on the paradigm of self-supervised contrastive learning, are capable of learning different types of fallibility representations from system logs. To address the second problem, we first conducts an error analysis of the learning process for fallibility representations. We discovered that due to the inherent random sampling process in the fallibility learning process, an error is generated at each step of the optimization trajectory, which ultimately accumulates in the model's weight parameters, a phenomenon we term the fallibility jitter problem. To solve the fallibility jitter problem, *Bifrost* proposes the Co-Anchored Fallibility Representation Learning (CARL) strategy. The key idea of this strategy is to use system logs from real execution flows as representation space anchors for multi-objective representation learning, thereby reducing weight errors during the learning process.

To evaluate the effectiveness of *Bifrost*, we conducted exhaustive evaluations on three of the most widely used public datasets [24, 42] for log-based fault diagnosis—BGL, Hadoop, and Thunderbird—as well as on dataset collected from an industrial ML-as-a-Service (MLaaS) system, Platform-X, covering the three fault diagnosis tasks of AD, RCA, and FI. The experimental results demonstrate that the log representations generated by *Bifrost* achieve advanced performance in fault diagnosis tasks. Compared to PLMs with a similar number of parameters, *Bifrost* achieved an average advantage of 20.81% across the three tasks. Compared to Large Language Models (LLMs) with tens of times more parameters than *Bifrost*, *Bifrost* also achieved an average advantage of 5.80%. This evidence underscores the potent role of *Bifrost* and fallibility representations in fault diagnosis tasks.

In summary, our contributions are as follows:

- We analyzed the log analysis experience of SREs and identified three types of information closely related to failure, which we term fallibility representations.
- We conducted an empirical study, the results of which show that existing PLM-based log representation methods perform poorly in encoding fallibility representations, leading to suboptimal failure diagnosis.
- We propose *Bifrost*, a representation learning method that enhances the failure diagnosis capabilities of PLMs by learning fallibility representations.
- Quantitative, qualitative and theoretical evaluations on four datasets demonstrate the effectiveness of *Bifrost* and fallibility representations in failure diagnosis.

2 Task Formulation

Our objective is to conduct fault diagnosis for web systems based on system logs. Consider a large-scale web system, within an observation window of length w, we have a log sequence defined as $\mathbf{x} = [l_1, l_2, \ldots, l_w]$, where l_i is a log event observed at a certain time step. The fault diagnosis aims to establish a framework that processes the input log sequence and outputs 1) the operational status of the system, and upon detection of an anomaly, further outputs 2) the root cause logs and 3) the fault type. This task consists of three sub-tasks:

- **Anomaly Detection (AD)**: Given an observed log sequence $\mathbf{x}$, a detector $\mathcal{D}$ is used to predict the presence of an anomaly. Its output, y, is a binary variable indicating whether the system is normal (0) or anomalous (1), i.e., $y = \mathcal{D}(\mathbf{x}) \in \{0,1\}$.
- **Root Cause Analysis (RCA)**: If the system state is determined to be anomalous, a localizer $\mathcal{R}$ is activated to evaluate the probability of each log l_i in the input sequence $\mathbf{x}$ being the root cause. That is, $\mathcal{R}(\mathbf{x}) = [p_1, p_2, \ldots, p_w]$, where $p_i \in [0,1]$ represents the probability that log l_i is the root cause.
- **Fault Identification (FI)**: After confirming a system anomaly, a multi-class classifier $\mathcal{M}$ is employed to identify the specific type of the fault. It is assumed that the potential fault types belong to a predefined, finite set $\mathcal{U} = \{u_1, u_2, \ldots, u_N\}$. The objective of the $\mathcal{M}$ is to predict the fault type corresponding to the current log sequence, i.e., $\mathcal{M}(\mathbf{x}) \in \mathcal{U}$.

3 Empirical Study

In section, we conduct an empirical study on existing PLM-based log representation methods on three systems [24, 42]: Thunderbird, BGL, and an industrial MLaaS system Platform-X. We manually examined the capability of the four most widely used PLMs [2, 11, 16, 25] to encode fallibility representations. **Appendix A** provides a detailed introduction to the baseline models and the systems.

3.1 How effective are existing PLMs in identifying unexpected execution flows?

In this section, we evaluate the capability of existing PLMs to identify unexpected execution flows in the form of predicting the next event. We first collect a normal execution flow of the system, denoted as $L=[l_1,l_2,...,l_T]$. Then, we use a time window w to partition L into execution flows of length w, represented as $\mathbf{x}_t=[l_{t-w},...,l_{t-1}]$. We use PLMs, denoted as $f_\theta(\cdot)$, for encoding to obtain a sequence of log representations $f_\theta(\mathbf{x}_t)$. This sequence is subsequently fed into a sequential neural network $g_\phi(\cdot)$ to predict the next log in the execution flow: $\hat{e}_t=g_\phi(f_\theta(\mathbf{x}_t))$. Our training objective is to maximize the cosine similarity $s(\hat{e}_t,e_t)$ between $\hat{e}_t$ and the embedding of the next true log in the execution flow, $e_t=f_\theta(l_t)$. Here, the parameters θ of the PLM f_θ are frozen to evaluate the representation quality of the original PLMs. After training is complete, we use unexpected execution flows for evaluation. For an unexpected event representation $\tilde{e}_t$, a smaller value of $s(\hat{e}_t,\tilde{e}_t)$ indicates a stronger capability of the PLM to identify unexpected execution flows.

We use the absolute cosine similarity, *Avg_Sim*, and the relative rank similarity, *Avg_Rank*, to quantify the similarity:

- $Avg\_Sim=\mathbb{E}[s(\tilde{e}_i,\hat{e}_i)]$
- $Avg\_Rank=\mathbb{E}\left[1+\sum_{e_j\in \mathrm{E}(L)}\mathbb{I}\big(s(\hat{e}_i,\tilde{e}_i)\le s(\hat{e}_i,e_j)\big)\right]$

where $\mathbb{E}$ denotes the expected value and $\mathbb{I}$ is the indicator function. A lower *Avg_Sim* suggests a stronger ability of the PLM to identify unexpected execution flows, whereas for *Avg_Rank*, the opposite is true. The experimental results on the Thunderbird system logs are shown in Table 1. It is readily apparent that existing methods consider unexpected execution flows to have a high similarity to historical execution flows, which indicates that the capability of existing PLMs to identify unexpected execution flows is not ideal.

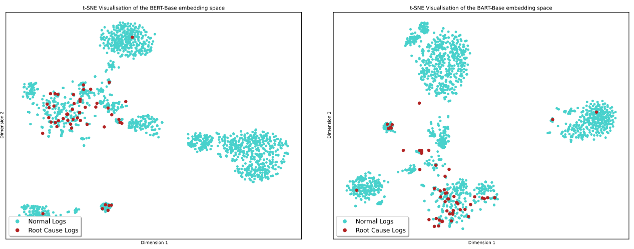
Table 1: Representational similarity between expected and unexpected events in PLMs.

Embedding Model	Thunderbird	
Metrics	**Avg_Sim ↓**	**Avg_Rank ↑**
Glove-300d [25]	78.08	13.49
BERT-Base [2]	89.62	21.69
ALBERT-Base [11]	92.22	21.05
BART-Base [16]	81.03	17.16
Average	88.10	20.94

3.2 Can existing PLMs discriminate anomalous system events?

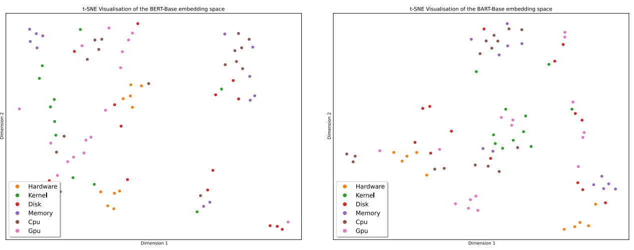
This section investigates the discriminative ability of existing PLMs for anomalous system events. Limited by space constraints, we select the most representative BERT-Base [2] and BART-Base [16] as baseline models, and choose the widely-used BGL as the system for our study. We use PLMs to encode the logs from the BGL system and then visualize these representations using t-SNE in Figure 2.

It is easy to observe that in the log representations obtained by existing PLMs, normal system events and anomalous system events are heavily stacked together, and the distribution of anomalous system events is relatively dispersed. This evidence indicates that existing PLMs do not have the ability to distinguish anomalous semantics within logs, resulting in non-ideal log representations.



(a) BERT-Base **(b) BART-Base**

Figure 2: Representations of anomalous events generated by PLMs.



(a) BERT-Base **(b) BART-Base**

Figure 3: Representations of logs from different system components generated by PLMs.

3.3 Can existing PLMs perceive different system components?

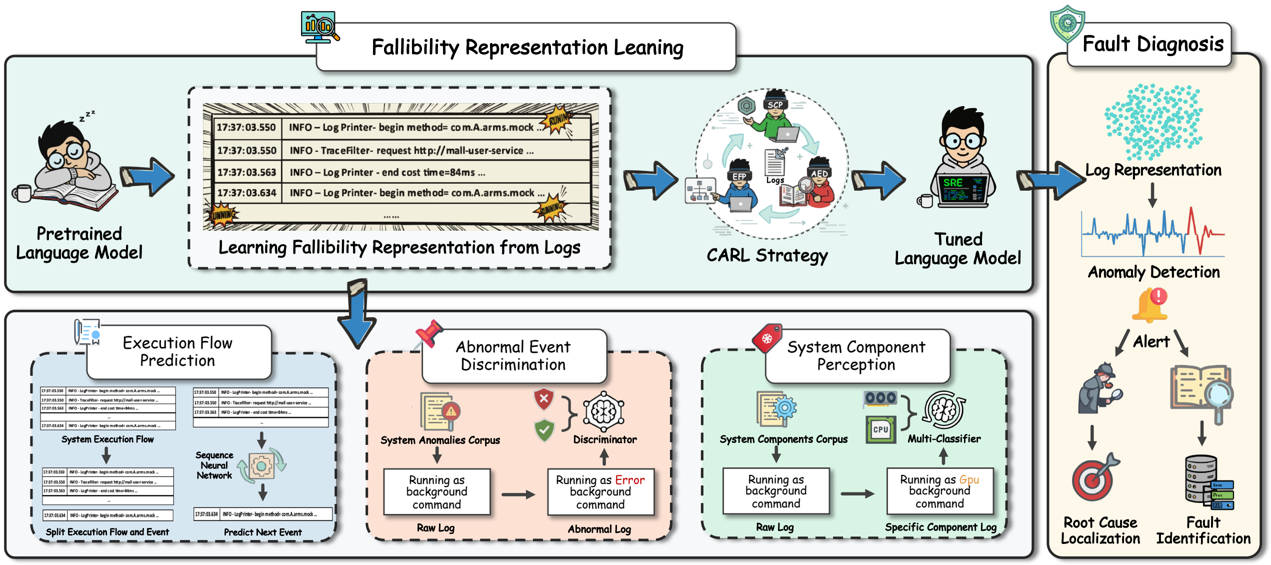
This section investigates the capability of existing PLMs to perceive different system components. Due to space constraints, we select the most representative BERT-Base and BART-Base as baseline models and choose Platform-X as the system for our study. Similar to Section 3.2, we use PLMs to encode the logs from the Platform-X system and then visualize these representations using t-SNE, with the results shown in Figure 3. It is easy to observe that in the log representations obtained by existing PLMs, the representations of different system components are heavily stacked together. This indicates that it is difficult for existing PLMs to perceive different system components from the raw logs.

Summary. PLM-based log representation methods are only pre-trained on natural language and overlook the multi-level structure of logs, leading to: 1) an inability to identify unexpected execution flows, 2) insensitivity to anomalous system events, and 3) a failure to separate different system components. Thus resulting in suboptimal fault diagnosis performance.

4 Methodology

4.1 Overview

This paper introduces *Bifrost*, an innovative log representation learning method designed to enhance the capabilities of PLMs in fault diagnosis. While log-based fault diagnosis is crucial for maintaining the reliability of web systems, existing PLM-based methods primarily encode semantic information from logs. They often fail to perceive system failure patterns embedded within the multi-level structure of logs, leading to suboptimal diagnostic performance. To bridge this gap, Bifrost draws inspiration from the analytical strategies of SREs. *Bifrost* formulate three distinct learning objectives, each targeting a specific type of fallibility representation corresponding to a different level of the log structure. However,



```
17:37:03.550  INFO – Log Printer- begin method= com.A.arms.mock ...
17:37:03.550  INFO - TraceFilter- request http://mall-user-service ...
17:37:03.563  INFO – Log Printer - end cost time=84ms ...
17:37:03.634  INFO – Log Printer- begin method= com.A.arms.mock ...
......
```

Figure 4: The Bifrost's pipeline. It learns log's fallibility representations and generates log representations for fault diagnosis.

influenced by in-batch stochastic sampling, the optimization process of fallibility learning exhibits significant instability. We term this **Fallibility Jitter** and have analyzed the weight error upper bound. To address the fallibility jitter problem, *Bifrost* proposes the **Co-Anchored Fallibility Representation Learning** strategy. By robustly learning these representations, *Bifrost* produces log embeddings that are highly effective for fault diagnosis tasks. The overall pipeline of *Bifrost* is illustrated in Figure 4.

4.2 Fallibility Representation for Log-based Fault Diagnosis

Bifrost aims to learn **fallibility representations** from system logs, thereby producing embeddings tailored for fault diagnosis. Since such representations cannot be directly extracted, we operationalize this concept by designing three self-supervised contrastive learning tasks: Execution Flow Prediction, Abnormal Event Discrimination, and System Component Perception.

4.2.1 ***Execution Flow Prediction (EFP).***

In real-world practice, SREs pay close attention to execution flows that deviate from established historical patterns. Such unexpected execution paths are considered strong indicators of system anomalies. Consequently, the ideal log representation must be capable of identifying these deviations. To this end, we propose the EFP task, which learns the patterns of normal system execution flows in a predictive manner.

We define the entire corpus of logs from normal system operation as a single, continuous execution flow, denoted by $\mathbf{L} = [l_1, l_2, \ldots, l_T]$. EFP uses a sliding window over this sequence to generate a context $\mathbf{x}_t = [l_{t-w}, \ldots, l_{t-1}]$. This context is encoded by a PLM $f_\theta(\cdot)$ and fed into an auxiliary model $g_\phi(\cdot)$ to produce a predictive embedding $\hat{\mathbf{e}}_t = g_\phi(f_\theta(\mathbf{x}_t))$. The objective of EFP is to make the predictive embedding $\hat{\mathbf{e}}_t$ similar to representations of events that belong to a normal execution flow, while being dissimilar to those of unexpected events. This forces the model to learn the general patterns of a normal flow rather than merely memorizing specific sequences.

To achieve this, we construct a contrastive learning objective. If the standard contrastive learning loss can be formalized as:

$$\mathcal{L}_{\mathrm{CL}}(f_\theta(l), f_\theta(\mathcal{P}(l)), f_\theta(\mathcal{N}(l)))$$
$$= -\sum \log \frac{\exp(s(f_\theta(l), f_\theta(\mathcal{P}(l)))/\tau)}{\exp(s(f_\theta(l), f_\theta(\mathcal{P}(l)))/\tau) + \sum_{\tilde{l} \in \mathcal{N}(l)} \exp(s(f_\theta(l), f_\theta(\tilde{l}))/\tau)}, \tag{1}$$

where $\mathcal{P}(\cdot)$ is the positive sample generator, $\mathcal{N}(\cdot)$ is the negative sample generator.

The optimization is formulated by minimizing the following loss:

$$\mathcal{L}_{\mathrm{EFP}}(l_t, \mathcal{P}_{\mathrm{E}}(l_i), \mathcal{N}_{\mathrm{E}}(l_t)) = \mathcal{L}_{\mathrm{CL}}(\hat{\mathbf{e}}_t, f_\theta(\mathcal{P}_{\mathrm{E}}(l_i)), f_\theta(\mathcal{N}_{\mathrm{E}}(l_t)))$$
$$= -\sum_{t=w}^{T} \log \frac{\exp(s(\hat{\mathbf{e}}_t, f_\theta(\mathcal{P}_{\mathrm{E}}(l_i)))/\tau)}{\exp(s(\hat{\mathbf{e}}_t, f_\theta(\mathcal{P}_{\mathrm{E}}(l_i)))/\tau) + \sum_{\tilde{l} \in \mathcal{N}_{\mathrm{E}}(l_t)} \exp(s(\hat{\mathbf{e}}_t, f_\theta(\tilde{l}))/\tau)}, \tag{2}$$

where:

- $l_i \in \mathbf{x}_t$ is a log randomly sampled from the context window, encouraging the prediction to capture the general characteristics of the normal execution flow.
- $\mathcal{P}_{\mathrm{E}}(\cdot)$ creates a semantically similar variant.
- $\mathcal{N}_{\mathrm{E}}(\cdot)$ creates a variant that represents an unexpected event.
- $s(\cdot,\cdot)$ is the cosine similarity function, and τ is a temperature hyperparameter.

The positive generator $\mathcal{P}_{\mathrm{E}}(\cdot)$ applies minor, semantics-preserving string transformations (e.g., random word deletion, synonym replacement, and swapping adjacent words). This strategy is proven to enhance representation robustness without significantly altering the original semantics.

The negative generator $\mathcal{N}_{\mathrm{E}}(\cdot)$ is based on the insight that unexpected execution flows often manifest failure-related semantics. It injects anomaly-related semantics by inserting a phrase from a predefined fault lexicon $\mathcal{A} = \{a_1, a_2, \ldots, a_m\}$ into the log, where each a_k represents a known system failure phenomenon. That is, $\mathcal{N}_{\mathrm{E}}(l_t) = \{l_t \oplus a_k \mid a_k \in \mathcal{A}\}$, where $\oplus$ denotes string insertion.

4.2.2 ***Abnormal Event Discrimination (AED)***.

At the event level, SREs abstract individual log entries into discrete system events. The specific semantics of each event are critical for determining the nature and cause of a fault. Therefore, an effective log representation must clearly separate normal system events from abnormal ones. To achieve this, *Bifrost* includes the AED task, which learns to discriminate anomalous semantics at the event level.

For each log event l within the execution flow $\mathbf{L}$, AED aims to pull its representation $f_\theta(l)$ closer to its semantically-preserved augmentations, which represent normal system events, and push it away from its anomaly-injected variants. This process explicitly trains the model to distinguish between normal and abnormal semantics.

$$\mathcal{L}_{\mathrm{AED}}(l,\mathcal{P}_{\mathrm{A}}(l),\mathcal{N}_{\mathrm{A}}(l))=\mathcal{L}_{\mathrm{CL}}(f_\theta(l),f_\theta(\mathcal{P}_{\mathrm{A}}(l)),f_\theta(\mathcal{N}_{\mathrm{A}}(l)))$$
$$=-\sum_{l\in\mathbf{L}}\log\frac{\exp(\mathrm{s}(f_\theta(l),f_\theta(\mathcal{P}_{\mathrm{A}}(l)))/\tau)}{\exp(\mathrm{s}(f_\theta(l),f_\theta(\mathcal{P}_{\mathrm{A}}(l)))/\tau)+\sum_{\tilde{l}\in\mathcal{N}_{\mathrm{A}}(l)}\exp(\mathrm{s}(f_\theta(l),f_\theta(\tilde{l}))/\tau)}, \tag{3}$$

- l is the original log sample serving as the anchor.
- $\mathcal{P}_{\mathrm{A}}(\cdot)$ generates a positive sample from l.
- $\mathcal{N}_{\mathrm{A}}(\cdot)$ generates a set of negative samples from l.

As the consistent generation goals, $\mathcal{P}_{\mathrm{A}}(\cdot)\equiv\mathcal{P}_{\mathrm{E}}(\cdot)$, $\mathcal{N}_{\mathrm{A}}(\cdot)\equiv\mathcal{N}_{\mathrm{E}}(\cdot)$.

4.2.3 ***System Component Perception (SCP)***.

Log entries often contain fine-grained information about specific system components (e.g., GPU, disk). By analyzing this information, SREs can localize a fault to a particular software or hardware unit, enabling precise diagnosis. An ideal log representation should therefore be able to distinguish between different system components. The SCP task is designed to learn representations that are sensitive to component-specific semantics.

The system components belong to a predefined set $C=\{c_1,\ldots,c_N\}$. For each log $l\in\mathbf{L}$, we randomly sample a component c_j from C. We then generate an anchor log $l'_j=l\oplus c_j$ by injecting the keyword of the sampled component. The objective of the SCP task is to make the representation of l'_j similar to its own augmentation but dissimilar to logs injected with different component keywords.

$$\mathcal{L}_{\mathrm{SCP}}(l'_j,\mathcal{P}_{\mathrm{S}}(l'_j),\mathcal{N}_{\mathrm{S}}(l,c_j))=\mathcal{L}_{\mathrm{CL}}(f_\theta(l'_j),f_\theta(\mathcal{P}_{\mathrm{S}}(l'_j)),f_\theta(\mathcal{N}_{\mathrm{S}}(l,c_j)))$$
$$=-\sum_{l\in\mathbf{L}}\log\frac{\exp(\mathrm{s}(f_\theta(l'_j),f_\theta(\mathcal{P}_{\mathrm{S}}(l'_j)))/\tau)}{\exp(\mathrm{s}(f_\theta(l'_j),f_\theta(\mathcal{P}_{\mathrm{S}}(l'_j)))/\tau)+\sum_{\tilde{l}\in\mathcal{N}_{\mathrm{S}}(l,c_j)}\exp(\mathrm{s}(f_\theta(l'_j),f_\theta(\tilde{l}))/\tau)}, \tag{4}$$

- c_j is randomly sampled from C for each l.
- $\mathcal{P}_{\mathrm{S}}(\cdot)\equiv\mathcal{P}_{\mathrm{E}}(\cdot)$ generates a positive sample from l'_j.
- $\mathcal{N}_{\mathrm{S}}(l,c_j)=\{l\oplus c_i\mid c_i\in C\setminus\{c_j\}\}$ generates negative samples by injecting different component keywords into l.

4.3 Learning from Fallibility Representation

The goal of *Bifrost* is to build an end-to-end framework for fallibility representation learning. To this end, we first introduce a vanilla learning strategy and analyze its optimization process. We find that inherent stochastic sampling leads to significant optimization instability—a problem we term **Fallibility Jitter**—which prevents the model from converging to an ideal representation space. To address this, we propose the **CARL** strategy. CARL enables robust and efficient representation learning by anchoring the entire optimization process on real system events.

4.3.1 ***Fallibility Jitter in Representation Learning***.

To learn the fallibility representation within a end-to-end framework, the vanilla strategy is to jointly optimize the three objectives. In each training step, given a central log l_t, this strategy randomly samples a context log l_i from its context window $[l_{t-w},\ldots,l_{t-1}]$ and an injected component c_j from the component set C. The $\mathcal{L}_{\mathrm{Vanilla}}$ is:

$$\mathcal{L}_{\mathrm{Vanilla}}=\lambda_1\mathcal{L}_{\mathrm{EFP}}(l_t,\mathcal{P}_{\mathrm{E}}(l_i),\mathcal{N}_{\mathrm{E}}(l_t))+\lambda_2\mathcal{L}_{\mathrm{AED}}(l_t,\mathcal{P}_{\mathrm{A}}(l_t),\mathcal{N}_{\mathrm{A}}(l_t))$$
$$+\lambda_3\mathcal{L}_{\mathrm{SCP}}(l'_j,\mathcal{P}_{\mathrm{S}}(l'_j),\mathcal{N}_{\mathrm{S}}(l_t,c_j)), \tag{5}$$

Where $l'_j=l_t\oplus c_j$. We conducted a detailed theoretical analysis of the optimization process for $\mathcal{L}_{\mathrm{Vanilla}}$; the details are provided in **Appendix B**. Our analysis indicates that the optimization of $\mathcal{L}_{\mathrm{Vanilla}}$ is affected by inherent stochastic sampling, causing the parameter trajectory $\mathbf{W}'_k$, driven by stochastic gradients, to deviate from the ideal trajectory $\mathbf{W}_k$, which is driven by the true gradient. This ultimately leads to suboptimal representation learning outcomes, a phenomenon we term as **fallibility jitter**. In **Appendix C**, we provide a detailed analysis of the upper bound of its weight error.

$$\mathbb{E}[\|\mathbf{W}_K-\mathbf{W}'_K\|]\le\eta\sum_{k=0}^{K-1}(1+\eta L)^{K-1-k}\mathbb{E}_{\mathbf{W}'_k}[\sigma_{\mathrm{Vanilla}}(\mathbf{W}'_k)], \tag{6}$$

Here, $\sigma_{\mathrm{Vanilla}}$ is the single-step gradient standard deviation, η is the learning rate, L is the Lipschitz constant of the gradient function and K is the training epochs. Equation (6) shows that the gradient variance $\sigma^2_{\mathrm{Vanilla}}$ at each step is amplified and accumulated into the final parameter deviation, causing the model parameters to oscillate drastically on the convergence path, which in turn degrades the quality of unreliability representation learning. Our analysis further reveals that this deviation is essentially caused by the stochastic sampling inherent in the optimization process.

4.3.2 ***Co-Anchored Fallibility Representation Learning***.

To solve the fallibility jitter problem, we propose the CARL strategy. Its core idea is to use the real, non-sampled system event l_t as an optimization anchor, providing a deterministic center for each stochastic batch to ensure the stability of the optimization process. Specifically, the CARL loss function is redesigned as:

$$\mathcal{L}_{\mathrm{CARL}}=\lambda_1\mathcal{L}_{\mathrm{EFP}}(l_t,\mathcal{P}_{\mathrm{E}}(l_t),\mathcal{N}_{\mathrm{SN}}(l_t,c_j))+\lambda_2\mathcal{L}_{\mathrm{CL}}(f_\theta(\mathcal{P}_{\mathrm{E}}(l_t)),f_\theta(\mathcal{P}_{\mathrm{E}}(l_i)),f_\theta(\mathcal{N}_{\mathrm{SN}}(l_t,c_j)))$$
$$+\lambda_3\mathcal{L}_{\mathrm{AED}}(l_t,\mathcal{P}_{\mathrm{A}}(l_t),\mathcal{N}_{\mathrm{SN}}(l_t,c_j))+\lambda_4\mathcal{L}_{\mathrm{SCP}}(l'_j,\mathcal{P}_{\mathrm{S}}(l'_j),\mathcal{N}_{\mathrm{S}}(l_t,c_j)), \tag{7}$$

where key designs include: (1) a new EFP task anchored at l_t (the λ_1 term); (2) an independent constraint $\mathcal{L}_{\mathrm{CL}}$ (the λ_2 term) to handle the stochasticity from sampling l_i; and (3) a shared negative set $\mathcal{N}_{\mathrm{SN}}=\mathcal{N}_{\mathrm{A}}(l_t)\cup\mathcal{N}_{\mathrm{S}}(l_t,c_j)$.

We have analyzed its optimization process in **Appendix D**. The analysis shows that CARL systematically reduces the weight errors through two structural mechanisms, thereby achieving more stable and efficient fallibility representation learning.

1. **Variance Isolation:** The variance introduced by sampling l_i is isolated into the independent $\mathcal{L}_{\mathrm{CL}}$ term, which typically has a smaller weight, thereby reducing its contribution to the total variance.
2. **Negative Covariance Constraint:** The shared negative set mechanism mathematically introduces a critical negative covariance term, which actively canceling out a portion of the variance generated by component sampling c_j.

Table 2: Comparison results with baseline methods for anomaly detection.

Embedding Model	Parameters	BGL			Hadoop			Thunderbird			Platform-X		
		Precision	Recall	F1-Score	Precision	Recall	F1-Score	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Pretrained Small Language Models													
Glove-300d	120M	88.41 ± 3.54	92.06 ± 0.33	90.16 ± 1.84	85.11 ± 4.31	94.82 ± 0.43	89.65 ± 2.51	84.67 ± 4.22	70.15 ± 13.98	76.13 ± 10.69	88.20 ± 2.73	86.73 ± 2.85	87.37 ± 0.41
BERT-Base	110M	93.90 ± 1.42	89.96 ± 1.33	91.89 ± 1.36	89.04 ± 9.11	94.92 ± 0.37	91.65 ± 5.38	91.50 ± 2.08	81.19 ± 0.37	86.03 ± 0.92	88.29 ± 2.07	92.78 ± 0.16	90.47 ± 1.07
ALBERT-Base	12M	94.60 ± 0.61	92.34 ± 0.08	93.46 ± 0.29	50.87 ± 3.33	94.65 ± 0.14	66.11 ± 2.74	97.25 ± 0.78	77.88 ± 2.38	86.46 ± 1.26	90.28 ± 2.20	87.26 ± 1.04	88.71 ± 0.73
BART-Base	140M	90.53 ± 4.11	91.96 ± 1.22	91.18 ± 1.88	70.18 ± 8.94	94.56 ± 0.39	80.27 ± 5.49	92.60 ± 3.82	79.97 ± 1.93	85.74 ± 1.26	81.37 ± 2.39	84.44 ± 2.35	82.83 ± 1.37
Pretrained Large Language Models													
GTE-Qwen-1.5B-Instruct	1500M	91.85 ± 3.52	93.47 ± 1.45	92.63 ± 2.36	60.40 ± 12.50	94.94 ± 0.30	73.13 ± 8.53	96.94 ± 1.03	80.79 ± 0.77	88.13 ± 0.67	98.72 ± 0.36	90.54 ± 0.22	94.45 ± 0.16
Stella_en_1.5B_v5	1500M	90.69 ± 2.64	90.73 ± 2.92	90.71 ± 2.68	63.99 ± 9.16	95.15 ± 0.29	76.14 ± 6.52	96.53 ± 1.97	83.20 ± 1.45	89.35 ± 0.80	98.92 ± 0.72	90.56 ± 0.17	94.55 ± 0.28
E5-mistral-7b-instruct	7200M	98.45 ± 0.94	94.16 ± 0.90	96.25 ± 0.53	78.84 ± 10.00	95.32 ± 0.52	85.93 ± 6.02	98.75 ± 0.42	84.18 ± 2.39	90.86 ± 1.35	99.13 ± 0.39	91.05 ± 0.20	94.92 ± 0.12
SFR-Embedding-Mistral	7200M	98.65 ± 0.34	95.72 ± 2.16	97.15 ± 0.99	83.12 ± 13.88	95.30 ± 0.66	88.13 ± 8.42	98.59 ± 0.37	81.27 ± 9.32	88.81 ± 5.65	99.08 ± 0.61	90.95 ± 0.25	94.84 ± 0.24
Learning from Log Models													
PreLog	140M	95.94 ± 2.31	89.38 ± 1.20	92.52 ± 0.91	72.72 ± 12.24	96.52 ± 0.13	82.38 ± 7.82	79.17 ± 11.60	75.98 ± 7.43	77.14 ± 8.11	88.34 ± 5.09	82.78 ± 5.90	85.12 ± 1.50
Bifrost (Ours)	140M	**98.81 ± 0.67**	**98.34 ± 0.53**	**98.57 ± 0.52**	**97.56 ± 0.81**	**96.14 ± 0.60**	**96.84 ± 0.22**	**97.50 ± 2.09**	**90.78 ± 2.13**	**94.01 ± 1.72**	**96.33 ± 0.76**	**97.25 ± 0.95**	**96.78 ± 0.21**

Table 3: Comparison results with baseline methods for root cause localization.

Dataset	Embedding Model	HR@1	HR@3	HR@5	PR@3	PR@5	MAP@3	MAP@5	MRR
	Pretrained Small Language Models								
	Glove-300d	40.00 ± 6.00	52.55 ± 5.39	56.29 ± 2.86	44.32 ± 5.02	45.81 ± 3.61	42.41 ± 5.46	43.65 ± 4.77	46.62 ± 5.13
	BERT-Base	58.64 ± 10.0	73.21 ± 12.3	79.83 ± 11.8	61.63 ± 5.87	65.62 ± 6.31	59.65 ± 6.95	61.70 ± 6.43	67.40 ± 8.81
	ALBERT-Base	53.74 ± 0.58	61.24 ± 2.68	68.07 ± 4.00	55.67 ± 1.11	58.13 ± 1.40	54.72 ± 0.76	55.78 ± 0.87	61.03 ± 1.07
	BART-Base	62.47 ± 5.26	78.10 ± 13.27	84.35 ± 12.71	68.04 ± 8.08	69.60 ± 8.63	65.38 ± 6.44	66.93 ± 7.29	72.11 ± 6.95
BGL	***Pretrained Large Language Models***								
	GTE-Qwen-1.5B-Instruct	81.24 ± 3.51	88.65 ± 3.86	90.05 ± 4.01	85.13 ± 3.92	85.60 ± 3.63	83.44 ± 3.78	84.22 ± 3.64	85.08 ± 3.73
	Stella_en_1.5B_v5	69.46 ± 7.23	89.01 ± 5.70	91.69 ± 3.49	75.78 ± 4.27	78.54 ± 4.10	73.47 ± 4.68	75.28 ± 4.33	79.55 ± 5.46
	E5-mistral-7b-instruct	78.85 ± 12.42	94.14 ± 2.61	95.02 ± 2.90	78.76 ± 4.52	79.29 ± 4.90	79.09 ± 6.39	79.25 ± 5.64	86.52 ± 7.13
	SFR-Embedding-Mistral	85.67 ± 4.96	92.35 ± 3.90	93.38 ± 3.61	88.57 ± 4.36	90.02 ± 4.00	87.16 ± 4.59	88.18 ± 4.36	89.06 ± 4.39
	Learning from Log Models								
	PreLog	54.16 ± 4.64	73.87 ± 9.27	80.08 ± 6.26	53.30 ± 4.53	55.26 ± 4.25	53.17 ± 3.82	53.83 ± 3.80	64.94 ± 4.84
	Bifrost (Ours)	**90.10 ± 1.25**	**94.67 ± 1.57**	**95.12 ± 1.45**	**92.40 ± 1.29**	**93.38 ± 1.22**	**91.22 ± 1.23**	**92.00 ± 1.25**	**92.43 ± 1.33**
	Pretrained Small Language Models								
	Glove-300d	8.07 ± 2.23	38.08 ± 5.15	54.78 ± 3.76	29.80 ± 3.41	42.67 ± 3.28	19.71 ± 2.73	27.68 ± 2.95	27.19 ± 2.50
	BERT-Base	25.77 ± 8.81	36.85 ± 1.49	42.44 ± 2.92	30.59 ± 3.86	37.16 ± 2.60	28.31 ± 5.84	31.22 ± 4.36	34.84 ± 4.61
	ALBERT-Base	38.43 ± 8.43	52.88 ± 4.37	61.96 ± 3.75	42.99 ± 5.48	50.85 ± 4.55	40.20 ± 6.73	43.67 ± 5.83	48.36 ± 5.96
	BART-Base	44.83 ± 5.08	58.94 ± 6.45	66.67 ± 6.63	46.54 ± 5.14	54.35 ± 5.13	45.61 ± 4.67	48.33 ± 4.62	55.37 ± 4.26
Thunderbird	***Pretrained Large Language Models***								
	GTE-Qwen-1.5B-Instruct	41.98 ± 5.96	67.47 ± 6.76	84.79 ± 7.48	56.96 ± 3.18	72.35 ± 5.38	49.36 ± 4.00	57.15 ± 4.42	58.29 ± 5.12
	Stella_en_1.5B_v5	52.65 ± 8.62	59.95 ± 9.53	69.08 ± 9.82	56.98 ± 9.45	63.63 ± 9.50	54.59 ± 8.95	57.58 ± 9.13	59.80 ± 8.77
	E5-mistral-7b-instruct	42.13 ± 10.22	70.16 ± 9.36	83.28 ± 5.43	57.29 ± 6.14	68.35 ± 4.96	49.80 ± 7.67	56.17 ± 6.69	59.08 ± 8.14
	SFR-Embedding-Mistral	47.38 ± 6.05	77.98 ± 7.70	91.33 ± 3.67	61.34 ± 5.63	72.84 ± 4.84	53.97 ± 5.81	60.45 ± 5.37	64.27 ± 5.54
	Learning from Log Models								
	PreLog	52.93 ± 12.22	74.30 ± 8.04	77.74 ± 7.71	61.66 ± 6.89	68.44 ± 7.26	57.47 ± 8.85	61.29 ± 7.68	63.40 ± 9.29
	Bifrost (Ours)	**75.77 ± 5.16**	**83.85 ± 5.51**	**86.07 ± 5.51**	**81.41 ± 5.31**	**83.18 ± 5.43**	**78.80 ± 5.29**	**80.37 ± 5.31**	**80.32 ± 5.31**
	Pretrained Small Language Models								
	Glove-300d	52.44 ± 3.17	70.24 ± 0.98	78.21 ± 1.83	56.29 ± 1.69	60.36 ± 1.45	54.22 ± 2.22	56.29 ± 1.55	62.64 ± 1.86
	BERT-Base	72.75 ± 0.77	79.05 ± 0.80	79.68 ± 0.70	71.43 ± 0.91	70.69 ± 0.78	72.05 ± 0.81	71.56 ± 0.76	75.91 ± 0.78
	ALBERT-Base	72.41 ± 1.84	81.42 ± 0.45	82.69 ± 0.21	72.70 ± 1.29	73.49 ± 1.39	72.64 ± 1.54	72.90 ± 1.44	76.98 ± 1.13
	BART-Base	74.80 ± 1.32	79.17 ± 1.16	80.91 ± 0.93	73.52 ± 0.95	74.62 ± 0.92	74.00 ± 1.06	74.11 ± 0.97	77.27 ± 1.04
Platform-X	***Pretrained Large Language Models***								
	GTE-Qwen-1.5B-Instruct	83.32 ± 0.37	84.91 ± 0.18	85.24 ± 0.34	80.37 ± 0.35	80.97 ± 0.73	81.78 ± 0.33	81.37 ± 0.36	84.19 ± 0.14
	Stella_en_1.5B_v5	83.15 ± 0.20	84.68 ± 0.59	85.67 ± 0.49	80.01 ± 0.80	80.35 ± 0.98	81.36 ± 0.66	80.91 ± 0.73	84.10 ± 0.33
	E5-mistral-7b-instruct	84.89 ± 0.76	86.29 ± 0.11	86.33 ± 0.08	79.74 ± 0.99	78.63 ± 1.05	82.13 ± 0.93	80.78 ± 0.64	85.57 ± 0.45
	SFR-Embedding-Mistral	84.69 ± 0.98	86.17 ± 0.36	86.26 ± 0.28	80.61 ± 1.05	79.95 ± 1.28	82.50 ± 0.64	81.49 ± 0.58	85.43 ± 0.65
	Learning from Log Models								
	PreLog	69.91 ± 1.61	77.20 ± 2.08	78.02 ± 2.06	71.95 ± 1.83	72.41 ± 2.04	71.11 ± 1.67	71.56 ± 1.78	73.62 ± 1.86
	Bifrost (Ours)	**87.98 ± 0.96**	**92.35 ± 0.86**	**93.08 ± 0.94**	**88.36 ± 0.95**	**88.40 ± 1.42**	**88.33 ± 0.85**	**88.35 ± 1.00**	**90.14 ± 0.93**

5 Experimental Evaluation

5.1 Experimental Setup

Details of the experimental setup are provided in **Appendix A**.

Datasets. We conducted extensive experiments on three public log datasets collected from BGL, Hadoop, and Thunderbird systems, as well as a log dataset from an industrial MLaaS platform, Platform-X, to demonstrate the effectiveness of Bifrost [24, 42].

Baselines. We use state-of-the-art PLM-based log representation methods [2, 11, 14, 16, 25] and method pre-trained on logs as the primary baselines. We also include LLMs with tens of times more parameters than Bifrost as strong baselines [17, 21, 29, 36].

Evaluation Task. We conducted a comprehensive fault diagnosis evaluation, covering three sub-tasks: AD, RCA, and FI.

Implementations. We present the detailed implementation. The code is available at https://anonymous.4open.science/r/Bifrost.

Table 4: Performance comparison on the fault identification.

Embedding Model	Macro-Precision	Macro-Recall	Macro-F1-Score
Pretrained Small Language Models			
Glove-300d	80.95 ± 0.50	77.29 ± 1.95	77.26 ± 1.13
BERT-Base	77.68 ± 2.65	72.11 ± 3.28	72.92 ± 3.15
ALBERT-Base	73.62 ± 8.14	71.44 ± 4.59	70.76 ± 6.15
BART-Base	47.47 ± 5.94	66.15 ± 1.92	54.86 ± 4.27
Pretrained Large Language Models			
GTE-Qwen-1.5B-Instruct	97.66 ± 0.65	88.02 ± 0.63	92.45 ± 0.56
Stella_en_1.5B_v5	96.77 ± 1.69	86.40 ± 3.11	90.76 ± 2.94
E5-mistral-7b-instruct	97.80 ± 1.27	88.25 ± 1.63	92.43 ± 1.76
SFR-Embedding-Mistral	97.78 ± 0.56	88.55 ± 0.61	92.73 ± 0.64
Learning from Log Models			
PreLog	51.00 ± 8.46	65.02 ± 1.01	56.38 ± 4.74
Bifrost (Ours)	**94.84 ± 0.44**	**92.81 ± 0.53**	**93.35 ± 0.43**

Table 5: Ablation results on the anomaly detection task.

Learning Objective				BGL			Hadoop			Thunderbird			Platform-X		
CARL	EFP	AED	SCP	Precision	Recall	F1-Score	Precision	Recall	F1-Score	Precision	Recall	F1-Score	Precision	Recall	F1-Score
✓	✓	✗	✗	98.45 ± 0.52	92.52 ± 1.86	95.39 ± 1.12	95.73 ± 1.43	95.55 ± 0.55	95.63 ± 0.78	96.36 ± 1.09	86.31 ± 4.06	91.03 ± 2.64	92.35 ± 2.09	94.46 ± 0.75	93.38 ± 1.19
✓	✗	✓	✗	98.11 ± 0.83	92.95 ± 1.59	95.45 ± 0.93	96.38 ± 1.69	95.49 ± 0.44	95.93 ± 0.83	97.89 ± 1.34	82.83 ± 4.77	89.63 ± 2.30	94.33 ± 1.77	89.99 ± 0.68	92.10 ± 0.58
✓	✗	✗	✓	98.10 ± 1.70	88.49 ± 1.18	93.03 ± 0.36	82.29 ± 5.84	97.34 ± 0.24	89.07 ± 3.44	96.15 ± 0.97	80.95 ± 0.50	87.89 ± 0.63	96.15 ± 1.24	89.54 ± 0.51	92.72 ± 0.48
✓	✗	✓	✓	98.66 ± 0.44	93.53 ± 2.27	96.01 ± 1.06	96.44 ± 0.54	96.00 ± 0.77	96.22 ± 0.40	96.45 ± 1.01	85.77 ± 2.45	90.78 ± 1.57	94.20 ± 2.50	97.23 ± 0.16	95.68 ± 1.27
✓	✓	✗	✓	98.70 ± 0.73	96.06 ± 2.08	97.35 ± 0.99	97.66 ± 0.70	95.87 ± 0.65	96.75 ± 0.30	94.77 ± 3.62	89.11 ± 2.80	91.78 ± 1.90	94.54 ± 1.14	95.92 ± 0.62	95.22 ± 0.35
✓	✓	✓	✗	99.15 ± 0.43	97.39 ± 1.53	98.25 ± 0.64	97.51 ± 0.98	95.72 ± 0.76	96.60 ± 0.49	96.84 ± 1.43	89.01 ± 2.23	92.74 ± 1.43	92.67 ± 2.06	94.95 ± 0.65	93.78 ± 0.91
✗	✓	✓	✓	98.84 ± 0.71	95.37 ± 3.25	97.04 ± 1.76	96.33 ± 0.26	94.34 ± 2.58	95.14 ± 1.58	95.18 ± 1.69	84.60 ± 2.18	89.55 ± 1.27	93.23 ± 1.55	96.01 ± 0.73	94.59 ± 0.69
✓	✓	✓	✓	**98.81 ± 0.67**	**98.34 ± 0.53**	**98.57 ± 0.52**	**97.56 ± 0.81**	**96.14 ± 0.60**	**96.84 ± 0.22**	**97.50 ± 2.09**	**90.78 ± 2.13**	**94.01 ± 1.72**	**96.33 ± 0.76**	**97.25 ± 0.95**	**96.78 ± 0.21**

Table 6: Ablation results on the root cause localization task.

Dataset	Learning Objective				Metrics							
	CARL	EFP	AED	SCP	HR@1	HR@3	HR@5	PR@3	PR@5	MAP@3	MAP@5	MRR
BGL	✓	✓	✗	✗	82.85 ± 3.59	88.86 ± 3.74	89.69 ± 4.00	85.55 ± 3.47	86.23 ± 3.16	84.23 ± 3.44	85.02 ± 3.38	85.81 ± 3.66
	✓	✗	✓	✗	81.24 ± 3.51	88.65 ± 3.86	90.05 ± 4.01	85.13 ± 3.92	85.60 ± 3.63	83.44 ± 3.78	84.22 ± 3.64	85.08 ± 3.73
	✓	✗	✗	✓	83.05 ± 3.99	89.05 ± 3.41	90.54 ± 3.25	81.21 ± 9.94	82.48 ± 10.9	81.70 ± 7.39	81.89 ± 8.74	86.20 ± 3.54
	✓	✗	✓	✓	85.98 ± 3.40	91.29 ± 2.99	92.08 ± 2.74	88.39 ± 3.14	89.34 ± 3.02	87.24 ± 3.22	88.01 ± 3.15	88.80 ± 3.09
	✓	✓	✗	✓	86.52 ± 1.98	91.86 ± 1.13	92.42 ± 1.05	89.34 ± 1.15	90.22 ± 1.06	88.05 ± 1.46	88.85 ± 1.30	89.35 ± 1.43
	✓	✓	✓	✗	87.86 ± 1.11	92.57 ± 1.33	93.31 ± 1.52	90.01 ± 1.11	91.03 ± 1.26	88.96 ± 1.12	89.70 ± 1.14	90.44 ± 1.18
	✗	✓	✓	✓	83.92 ± 1.79	90.27 ± 1.90	91.39 ± 1.76	87.55 ± 1.77	88.91 ± 1.72	85.80 ± 1.71	86.95 ± 1.71	87.32 ± 1.56
	✓	✓	✓	✓	**90.10 ± 1.25**	**94.67 ± 1.57**	**95.12 ± 1.45**	**92.40 ± 1.29**	**93.38 ± 1.22**	**91.22 ± 1.23**	**92.00 ± 1.25**	**92.43 ± 1.33**
Thunderbird	✓	✓	✗	✗	66.79 ± 7.61	76.08 ± 6.58	77.70 ± 6.71	73.02 ± 7.01	75.14 ± 6.75	70.09 ± 7.24	71.94 ± 7.08	71.80 ± 7.14
	✓	✗	✓	✗	67.11 ± 3.77	74.66 ± 4.02	76.75 ± 3.70	72.21 ± 4.08	73.80 ± 3.95	69.88 ± 4.03	71.30 ± 4.01	71.21 ± 3.80
	✓	✗	✗	✓	65.39 ± 5.92	75.45 ± 7.08	78.28 ± 7.38	72.13 ± 6.72	74.74 ± 7.09	68.86 ± 6.34	70.98 ± 6.61	70.87 ± 6.34
	✓	✗	✓	✓	70.31 ± 4.06	79.23 ± 3.62	81.69 ± 3.94	76.00 ± 3.86	78.29 ± 3.83	73.26 ± 3.93	75.06 ± 3.82	75.23 ± 3.72
	✓	✓	✗	✓	68.99 ± 9.46	78.49 ± 8.54	81.13 ± 8.58	74.94 ± 8.45	77.49 ± 8.46	72.01 ± 8.83	73.96 ± 8.68	74.16 ± 8.89
	✓	✓	✓	✗	71.70 ± 4.35	81.49 ± 6.27	84.08 ± 6.60	78.49 ± 5.44	80.83 ± 6.04	75.37 ± 4.90	77.34 ± 5.32	76.98 ± 5.08
	✗	✓	✓	✓	70.67 ± 2.84	83.65 ± 3.86	85.72 ± 3.58	78.81 ± 2.91	81.67 ± 3.25	74.90 ± 2.82	77.36 ± 2.88	77.34 ± 2.98
	✓	✓	✓	✓	**75.77 ± 5.16**	**83.85 ± 5.51**	**86.07 ± 5.51**	**81.41 ± 5.31**	**83.18 ± 5.43**	**78.80 ± 5.29**	**80.37 ± 5.31**	**80.32 ± 5.31**
Platform-X	✓	✓	✗	✗	82.43 ± 1.58	88.55 ± 1.86	90.49 ± 1.93	81.57 ± 1.90	80.45 ± 2.17	82.18 ± 1.73	81.60 ± 1.85	85.62 ± 1.69
	✓	✗	✓	✗	82.52 ± 0.71	89.05 ± 0.97	91.45 ± 1.13	79.43 ± 2.04	77.73 ± 2.31	81.19 ± 1.38	79.94 ± 1.75	86.12 ± 0.76
	✓	✗	✗	✓	81.98 ± 1.15	87.79 ± 2.05	89.66 ± 2.57	80.38 ± 1.99	79.09 ± 2.50	81.30 ± 1.22	80.51 ± 1.55	85.09 ± 1.65
	✓	✗	✓	✓	85.61 ± 1.91	90.89 ± 1.18	91.95 ± 0.93	85.57 ± 2.55	84.57 ± 3.74	85.81 ± 2.18	85.39 ± 2.68	88.27 ± 1.45
	✓	✓	✗	✓	84.07 ± 1.60	89.22 ± 1.47	90.36 ± 1.33	83.94 ± 2.77	83.04 ± 3.87	84.16 ± 2.30	83.78 ± 2.82	86.68 ± 1.41
	✓	✓	✓	✗	86.64 ± 2.52	92.32 ± 1.49	93.78 ± 1.04	84.95 ± 2.38	84.16 ± 2.70	85.93 ± 2.36	85.27 ± 2.45	89.60 ± 1.90
	✗	✓	✓	✓	86.55 ± 0.24	91.98 ± 0.65	92.86 ± 0.64	85.60 ± 0.68	84.64 ± 1.81	86.29 ± 0.44	85.74 ± 0.86	89.27 ± 0.31
	✓	✓	✓	✓	**87.98 ± 0.96**	**92.35 ± 0.86**	**93.08 ± 0.94**	**88.36 ± 0.95**	**88.40 ± 1.42**	**88.33 ± 0.85**	**88.35 ± 1.00**	**90.14 ± 0.93**

Table 7: Ablation results on the fault identification task.

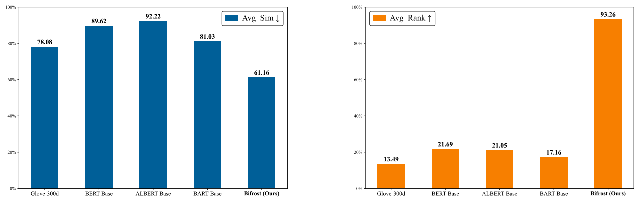
Learning Objective				Platform-X		
CARL	EFP	AED	SCP	Macro-Precision	Macro-Recall	Macro-F1
✓	✓	✗	✗	81.63 ± 1.25	78.62 ± 1.19	77.89 ± 1.09
✓	✗	✓	✗	82.57 ± 1.66	78.93 ± 1.91	77.82 ± 1.79
✓	✗	✗	✓	98.17 ± 0.46	91.80 ± 1.89	94.48 ± 0.89
✓	✗	✓	✓	93.00 ± 0.82	90.67 ± 1.42	90.80 ± 1.65
✓	✓	✗	✓	93.06 ± 0.53	91.20 ± 0.37	91.52 ± 0.49
✓	✓	✓	✗	83.18 ± 1.34	81.93 ± 1.73	80.90 ± 1.79
✗	✓	✓	✓	86.95 ± 2.38	85.68 ± 1.97	85.54 ± 2.17
✓	✓	✓	✓	**94.84 ± 0.44**	**92.81 ± 0.53**	**93.35 ± 0.43**

5.2 RQ1: How effective is fallibility representation in the log-based fault diagnosis?

This section evaluates the effectiveness of the fallibility representation in log-based fault diagnosis tasks; Tables 2, 3, and 4 present the comparison results with baseline models on three tasks. Overall, Bifrost achieves state-of-the-art log representations. For the AD task, the results obtained by Bifrost are, on average, 10.42%, 6.80%, and 12.26% higher in F1-score than the three baseline methods. For the RCA task, Bifrost outperforms the three baselines by an average of 27.61%, 9.35%, and 17.87% on HR@k. For the FI task, Bifrost achieves an even more significant advantage, surpassing the three baselines by an average of 24.40%, 1.26%, and 36.97% in Macro-F1. This substantial leap indicates that the fallibility representation can indeed capture the rich system failure information embedded in logs. This evidence demonstrates the potent effectiveness of the fallibility representation in fault diagnosis tasks.

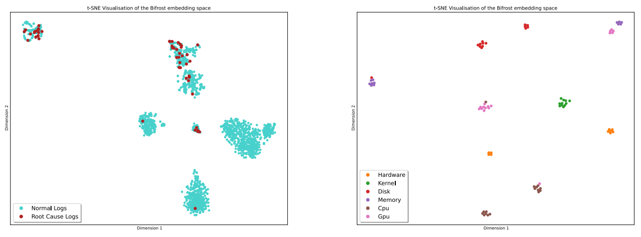
5.3 RQ2: What are the contributions of different training strategies to Bifrost?

This section conducts an ablation study to evaluate the effectiveness of the three learning objectives and one learning strategy in Bifrost. The results of the ablation study are shown in Table 5, 6 and 7. Overall, the ablation of all training strategies suffered varying degrees of performance degradation, which demonstrates the effectiveness



(a) Avg_Sim as Metric **(b) Avg_Rank as Metric**

Figure 5: Representational similarity in Bifrost.



(a) Event Representation **(b) Component Representation**

Figure 6: The log representation generated by Bifrost.

of the proposed strategies. In the ablation of the three learning objectives, ablating only a single objective caused an average performance drop of 1.45%, 2.66%, and 5.61% on the three tasks. Simultaneously ablating multiple learning objectives showed a more significant performance drop, with an average of 3.95%, 6.03%, and 9.95%. This indicates that the three well-designed learning objectives indeed help in learning the fallibility representations of logs. Furthermore, the ablation of the CARL strategy also showed a significant performance degradation, causing an average performance loss of 2.47%, 2.44%, and 7.81% on the three tasks. This evidence proves that the CARL strategy can indeed alleviate the fallibility jitter problem, thereby enabling the learning of higher-quality log representations.

5.4 RQ3: How capable is Bifrost of encoding the fallibility representation?

This section evaluates whether Bifrost effectively encodes the fallibility representations of logs. To this end, we examined the log representations of Bifrost, following the experimental setup in Section 3. Specifically, at the execution flow level, as shown in Figure 5, Bifrost achieved better performance on both the *Avg_Sim* and *Avg_Rank* metrics, indicating its ability to identify unexpected execution flows. In Figure 6a, at the event level, compared to baseline PLMs, the anomalous event representations encoded by Bifrost aggregate into distinct small clusters, demonstrating significant separability. In Figure 6b, at the component level, the individual components in the log representations encoded by Bifrost cluster together respectively, which showcases its awareness of system components. This evidence demonstrates that Bifrost can indeed effectively learn the fallibility representations of logs through its well-designed training strategies, thereby achieving effective fault diagnosis.

6 Related Work

6.1 Log-based Web System Fault Diagnosis

Discovering and identifying problems by analyzing logs has always been an active area of research [9, 26, 33, 37, 41]. Recent fault diagnosis work has been studied from the three tasks of AD, RCA, and FI. Some state-of-the-art AD methods [3, 34, 39] first extract execution flow sequences, then learn the patterns of historical execution flow sequences and compare them with log sequences in the production environment to detect conflicts. Other works focus on localizing the root causes of anomalies. LogRCA [30] ranks all logs within the investigation time window based on PU-Learning. Onion [40] proposed a form of log aggregation called Log Clique to identify event-indicating logs. Other works [4, 27, 41] have studied the classification of faults, mainly by constructing deep learning-based models to capture the sequential features of faults. However, all these works use PLMs as the log representation method, which cannot bridge the structural gap between logs and natural language, thus failing to meet the requirements of fault diagnosis tasks. Bifrost addresses this by compelling PLMs to learn fallibility representations of logs through meticulously designed tasks.

6.2 System Log Representation

Learning log-oriented representations from system logs has been extensively studied in recent years [5, 7, 14, 20, 32]. LLMeLog [7] conducted an empirical study whose results show that log representations encoded by existing PLMs suffer from issues such as content missing, which it improves by rewriting system logs using LLMs. KnowLog [20] proposed a training framework enhanced by system knowledge, which learns the abbreviation, context, and style knowledge of logs from system logs through well-designed training tasks to improve the log representation. PreLog [14] is a PLM pre-trained on system logs, designed to uniformly perform various log analysis tasks through the paradigm of prompt-tuning. For AD tasks, LogFormer [5] proposed a cross-system anomaly detection paradigm, which aims to learn general system representations and adapt to different systems through fine-tuning. Although these works all learn from system logs, they only focus on the semantic information of logs, ignoring their multi-level structural information, and are not designed for fault diagnosis tasks. In contrast, Bifrost draws inspiration from the log analysis experience of SREs, compelling PLMs to learn fallibility representations of logs to obtain high-quality log representations oriented toward fault diagnosis tasks.

7 Conclusion

In this paper, we propose Bifrost, a log representation learning method for log-based fault diagnosis. Inspired by the log analysis experience of SREs, Bifrost utilizes multi-level information from logs—such as execution flows, events, and components—as fallibility representations and learns them through meticulously designed strategies based on self-supervised contrastive learning to enhance the capabilities of PLMs in fault diagnosis. Comprehensive evaluations on three public systems and one industrial MLaaS system show that the log representations generated by Bifrost achieve an average advantage of 9.83%, 18.28% and 20.88% over existing PLMs. This evidence highlights the powerful role of Bifrost and its fallibility representations in fault diagnosis. In the future, we plan to explore larger-scale log representation models based on fallibility representations to achieve more precise, effective, and general-purpose industrial fault diagnosis.

References

[1] 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.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 NAACL*. 4171–4186.

[3] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 1285–1298.

[4] Chiming Duan, Yong Yang, Tong Jia, Guiyang Liu, Jinbu Liu, Huxing Zhang, Qi Zhou, Ying Li, and Gang Huang. 2025. FAMOS: Fault diagnosis for Microservice Systems through Effective Multi-modal Data Fusion. In *ICSE 2025*. IEEE Computer Society, 610–610.

[5] Hongcheng Guo, Jian Yang, Jiaheng Liu, Jiaqi Bai, Boyang Wang, Zhoujun Li, Tieqiao Zheng, Bo Zhang, Junran Peng, and Qi Tian. 2024. Logformer: A pre-train and tuning pipeline for log anomaly detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 135–143.

[6] Minghua He, Chiming Duan, Pei Xiao, Tong Jia, Siyu Yu, Lingzhe Zhang, Weijie Hong, Jin Han, Yifan Wu, Ying Li, et al. 2025. United We Stand: Towards End-to-End Log-based Fault Diagnosis via Interactive Multi-Task Learning. *arXiv preprint arXiv:2509.24364* (2025).

[7] Minghua He, Tong Jia, Chiming Duan, Huaqian Cai, Ying Li, and Gang Huang. 2024. LLMeLog: An Approach for Anomaly Detection based on LLM-enriched Log Events. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 132–143.

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

[9] Junjie Huang, Zhihan Jiang, Zhuangbin Chen, and Michael Lyu. 2025. No More Labelled Examples? An Unsupervised Log Parser with LLMs. *Proceedings of the ACM on Software Engineering* 2, FSE (2025), 2406–2429.

[10] Miao Jiang, Mohammad A Munawar, Thomas Reidemeister, and Paul AS Ward. 2011. Efficient fault detection and diagnosis in complex software systems with information-theoretic monitoring. *IEEE Transactions on Dependable and Secure Computing* 8, 4 (2011), 510–522.

[11] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. [n. d.]. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. In *International Conference on Learning Representations*.

[12] Van-Hoang Le and Hongyu Zhang. 2021. Log-based anomaly detection without log parsing. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 492–504.

[13] 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.

[14] Van-Hoang Le and Hongyu Zhang. 2024. Prelog: A pre-trained model for log analytics. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–28.

[15] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, and Michael R Lyu. 2023. Eadro: An end-to-end troubleshooting framework for microservices on multi-source data. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1750–1762.

[16] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 7871–7880.

[17] Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. 2023. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281* (2023).

[18] Qingwei Lin, Tianci Li, Pu Zhao, Yudong Liu, Minghua Ma, Lingling Zheng, Murali Chintalapati, Bo Liu, Paul Wang, Hongyu Zhang, et al. 2023. Edits: An easy-to-difficult training strategy for cloud failure prediction. In *Companion Proceedings of the ACM Web Conference 2023*. 371–375.

[19] Chuan Luo, Pu Zhao, Bo Qiao, Youjiang Wu, Hongyu Zhang, Wei Wu, Weihai Lu, Yingnong Dang, Saravanakumar Rajmohan, Qingwei Lin, et al. 2021. NTAM: neighborhood-temporal attention model for disk failure prediction in cloud platforms. In *Proceedings of the Web Conference 2021*. 1181–1191.

[20] Lipeng Ma, Weidong Yang, Bo Xu, Sihang Jiang, Ben Fei, Jiaqing Liang, Mingjie Zhou, and Yanghua Xiao. 2024. Knowlog: Knowledge enhanced pre-trained language model for log understanding. In *Proceedings of the 46th ieee/acm international conference on software engineering*. 1–13.

[21] Rui Meng, Ye Liu, Shafiq Rayhan Joty, Caiming Xiong, Yingbo Zhou, and Semih Yavuz. 2024. SFR-Embedding-Mistral:Enhance Text Retrieval with Transfer Learning. Salesforce AI Research Blog. https://www.salesforce.com/blog/sfr-embedding/

[22] 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.

[23] Shuyi Niu, Jiawei Jin, Xiutian Huang, Yonggeng Wang, Wenhao Xu, and Youyong Kong. 2023. Locating Faulty Applications via Semantic and Topology Estimation. In *Companion Proceedings of the ACM Web Conference 2023*. 528–532.

[24] Adam Oliner and Jon Stearley. 2007. What supercomputers say: A study of five system logs. In *37th annual IEEE/IFIP international conference on dependable systems and networks (DSN'07)*. IEEE, 575–584.

[25] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[26] Thomas Reidemeister, Mohammad A Munawar, and Paul AS Ward. 2010. Identifying symptoms of recurrent faults in log files of distributed information systems. In *2010 IEEE Network Operations and Management Symposium-NOMS 2010*. IEEE, 187–194.

[27] Yicheng Sui, Yuzhe Zhang, Jianjun Sun, Ting Xu, Shenglin Zhang, Zhengdan Li, Yongqian Sun, Fangrui Guo, Junyu Shen, Yuzhi Zhang, et al. 2023. Logkg: Log failure diagnosis through knowledge graph. *IEEE Transactions on Services Computing* 16, 5 (2023), 3493–3507.

[28] Jinrui Sun, Tong Jia, Minghua He, Yihan Wu, Ying Li, and Gang Huang. 2025. Exploring Variable Potential for LLM-based Log Parsing Efficiency and Reduced Costs. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*. 596–600.

[29] Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2023. Improving Text Embeddings with Large Language Models. *arXiv preprint arXiv:2401.00368* (2023).

[30] Thorsten Wittkopp, Philipp Wiesner, and Odej Kao. 2024. LogRCA: Log-Based Root Cause Analysis for Distributed Services. In *European Conference on Parallel Processing*. Springer, 362–376.

[31] W Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. 2016. A survey on software fault localization. *IEEE Transactions on Software Engineering* 42, 8 (2016), 707–740.

[32] Xingfang Wu, Heng Li, and Foutse Khomh. 2023. On the effectiveness of log representation for log-based anomaly detection. *Empirical Software Engineering* 28, 6 (2023), 137.

[33] Wensheng Xia, Ying Li, Tong Jia, and Zhonghai Wu. 2019. Bugidentifier: An approach to identifying bugs via log mining for accelerating bug reporting stage. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 167–175.

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

[35] Qingyang Yu, Changhua Pei, Bowen Hao, Mingjie Li, Zeyan Li, Shenglin Zhang, Xianglin Lu, Rui Wang, Jiaqi Li, Zhenyu Wu, et al. 2023. Cmdiagnostor: An ambiguity-aware root cause localization approach based on call metric data. In *Proceedings of the ACM web conference 2023*. 2937–2947.

[36] Dun Zhang, Jiacheng Li, Ziyang Zeng, and Fulong Wang. 2025. Jasper and Stella: distillation of SOTA embedding models. arXiv:2412.19048 [cs.IR] https://arxiv.org/abs/2412.19048

[37] Lingzhe Zhang, Liancheng Fang, Chiming Duan, Minghua He, Leyi Pan, Pei Xiao, Shiyu Huang, Yunpeng Zhai, Xuming Hu, Philip S Yu, et al. 2025. A Survey on Parallel Text Generation: From Parallel Decoding to Diffusion Language Models. *arXiv preprint arXiv:2508.08712* (2025).

[38] Shenglin Zhang, Sibo Xia, Wenzhao Fan, Binpeng Shi, Xiao Xiong, Zhenyu Zhong, Minghua Ma, Yongqian Sun, and Dan Pei. 2024. Failure Diagnosis in Microservice Systems: A Comprehensive Survey and Analysis. *arXiv preprint arXiv:2407.01710* (2024).

[39] 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.

[40] 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.

[41] Yingying Zhang, Zhengxiong Guan, Huajie Qian, Leili Xu, Hengbo Liu, Qingsong Wen, Liang Sun, Junwei Jiang, Lunting Fan, and Min Ke. 2021. CloudRCA: A root cause analysis framework for cloud computing platforms. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 4373–4382.

[42] Jieming Zhu, Shilin He, Pinjia He, Jinyang Liu, and Michael R Lyu. 2023. 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, 355–366.

A Evaluation Setup Details

A.1 Datasets

The BGL dataset [24] is derived from the operational logs of the Blue Gene/L supercomputing system, which contains 128K processors. The Thunderbird dataset [24] contains over 200 million log messages, collected a supercomputer with 9,024 processors and 27,072 GB of memory. The Hadoop dataset [42] was gathered from a 46-core Hadoop distributed file system spanning five machines. Platform-X is an integrated MLaaS platform deployed by a major cloud service provider, featuring over 6,000 heterogeneous GPUs responsible for executing ML jobs submitted by users. We collected system logs from Platform-X under a 7-day workload for evaluation, and the failures were manually classified into 6 different software and hardware types by SREs. Specifically, these include failures related to hardware, kernel, disk, memory, CPU, and GPU. Considering the huge scale of the Thunderbird dataset, we followed the settings of the previous study [12, 13] and selected the earliest 10 million log messages for experimentation.

A.2 Baselines

To better evaluate the effectiveness of Bifrost, we compare it with state-of-the-art PLM-based log representation methods, as well as methods pre-trained on system logs. We also compare Bifrost with LLMs whose parameter counts are tens of times larger. Specifically, for PLMs in the 100M parameter class, we selected the most widely used BERT-Base [2], BART-Base [16], Glove-300d [25], and ALBERT-Base [11] as the main baseline models to evaluate Bifrost's effectiveness in enhancing the log representation capabilities of PLMs. For LLMs with over 1B parameters, we selected the state-of-the-art text representation language models GTE-Qwen-1.5B-Instruct [17], Stella_en_1.5B_v5 [36], E5-mistral-7b-instruct [29], and SFR-Embedding-Mistral [21] as strong comparative baselines to comprehensively explore the potential of Bifrost. Furthermore, to more comprehensively evaluate the effectiveness of Bifrost, we also include the language model PreLog [14], which is pre-trained on logs, as a baseline model.

A.3 Evaluation Task and Metrics

To comprehensively evaluate the effectiveness of Bifrost in the fault diagnosis of log-based web systems, we conducted a fault diagnosis evaluation comprising three sub-tasks: AD, RCA, and FI.

AD: For the anomaly detection task, we conducted evaluations on four datasets: BGL, Hadoop, Thunderbird, and Platform-X. The evaluation metrics used were Precision, Recall, and F1 Score [3, 13].

RCA: For the root cause analysis task, we conducted evaluations on three datasets: BGL, Thunderbird, and Platform-X. This is because Hadoop is a distributed system dataset and does not provide corresponding root cause logs, making it unsuitable for RCA evaluation. The evaluation metrics used are Hit Rate (HR@k), Normalized Discounted Cumulative Gain (NDCG@k), Mean Average Precision (MAP@k), and Mean Reciprocal Rank (MRR). We set k=1, 3, and 5 for the evaluation. When k=1, the meaning of the metrics, except for HR, is equivalent to HR@1; therefore, we only report the cases for k = 3 and 5 [6, 30].

FI: For the fault identification task, we only conducted the evaluation on the Platform-X dataset. This is because existing public log systems do not provide corresponding fault types, which are required for the evaluation. We used Macro-Precision, Macro-Recall, and Macro-F1 as the metrics, which are designed to measure the identification capability across multiple fault types [4, 27, 41].

A.4 Implementation Details

To ensure the reproducibility of the research, we introduce the detailed implementation details.

Bifrost Training. Initially, Bifrost is developed based on **BART-Base** [16] as the base model, employing the proposed CARL strategy to learn fallibility representations. The AdamW optimizer is used for training optimization. The hyperparameters $\lambda 1, \lambda 2, \lambda 3, \lambda 4$ used to balance the joint training loss function are set to 1, 0.5, 1, and 1, respectively. Table 8 and Table 9 show the system failure corpus and the system component corpus designed by Bifrost based on the log analysis experience of SREs. Bifrost sets the size of the system failure corpus to 10 (i.e., selecting the top 10 for sample training) and the system component corpus to 8 to perform fallibility learning within limited resources. The code is available at https://anonymous.4open.science/r/Bifrost.

Fault Diagnosis Evaluation. For the evaluation of fault diagnosis, we use state-of-the-art baselines for the corresponding sub-tasks. Specifically, for the AD task, we use LogRobust [39] as the backbone; for the RCA task, we use LogRCA [30] as the backbone; and for the FI task, we use CloudLog [41] as the backbone. Following the settings of existing work [13, 22], we group the dataset using a sliding window of size 20, parse the log messages using Drain [8], and divide the dataset into training, testing, and validation sets at a ratio of 6:3:1. For all experiments, we ran them five times and reported the mean and standard deviation. Bifrost was experimented on a Linux server equipped with an NVIDIA H20 96GB GPU Memory. For the baseline methods used for comparison in the experiments, we adopted their publicly available implementations and settings [13].

Table 8: The system failure corpus in Bifrost.

Index	Failure	Index	Failure
1	abort	11	terminate
2	refuse	12	invalid
3	except	13	assertion
4	interrupt	14	unexpect
5	error	15	bad
6	failure	16	close
7	critical	17	incorrect
8	anomaly	18	unsuccess
9	fail	19	delay
10	timeout	20	warn

Table 9: The system component corpus in Bifrost.

Index	Component	Index	Component
1	cpu fault	5	memory error
2	gpu failure	6	kernel panic
3	disk error	7	network disconnected
4	hardware timeout	8	file corrupt

B Gradient Analysis for Fallibility Jitter

In this section, we analyze the optimization process of fallibility representation learning from the perspective of gradients in the optimization space. We find that due to the random sampling of system logs and system components during optimization, the gradient trajectory of the optimization process deviates from the ideal gradient trajectory. This is the root cause of the instability in fallibility representation learning, which we term the fallibility jitter problem.

Let $\mathbf{W}$ be the set of all trainable parameters of the model. In a single training iteration, given a center log l_t, the fallibility learning process involves random sampling from two independent distributions: 1) sampling a context log $l_i \sim \text{Uniform}(\mathbf{x}_t)$ from the context window $\mathbf{x}_t$, and 2) sampling an injected component $c_j \sim \text{Uniform}(C)$ from the component set C. We denote the source of randomness in this iteration as a composite random variable $\xi = (l_i, c_j)$.

In equation 5, the vanilla strategy for fallibility learning:

$$\mathcal{L}_{\text{Vanilla}}(\mathbf{W}; l_t, l_i, c_j) = \lambda_1 \mathcal{L}_{\text{EFP}}(\mathbf{W}; l_t, l_i) + \lambda_2 \mathcal{L}_{\text{AED}}(\mathbf{W}; l_t) + \lambda_3 \mathcal{L}_{\text{SCP}}(\mathbf{W}; l_t, c_j), \tag{8}$$

The **Stochastic Gradient** of $\mathcal{L}_{\text{Vanilla}}$ is:

$$\mathbf{g}_{\text{Vanilla}}(\mathbf{W}; l_t, \xi) = \nabla_{\mathbf{W}} \mathcal{L}_{\text{Vanilla}}(\mathbf{W}; l_t, \xi), \tag{9}$$

Here, the stochastic gradient is typically considered an unbiased estimator of the **True Gradient**. For $\mathcal{L}_{\text{Vanilla}}$, the true gradient is the gradient obtained by taking the expectation over all sources of randomness, conditioned on l_t:

$$\mathbf{G}_{\text{Vanilla}}(\mathbf{W}; l_t) = \mathbb{E}_{\xi}[\mathbf{g}_{\text{Vanilla}}(\mathbf{W}; l_t, \xi)], \tag{10}$$

During the fallibility representation learning process, the deviation between the stochastic gradient and the true gradient is a source of error. We use the **Gradient Variance** to quantify the expected squared norm of this deviation:

$$\sigma^2_{\text{Vanilla}}(\mathbf{W}; l_t) = \mathbb{E}_{\xi}\left[\|\mathbf{g}_{\text{Vanilla}}(\mathbf{W}; l_t, \xi) - \mathbf{G}_{\text{Vanilla}}(\mathbf{W}; l_t)\|^2\right], \tag{11}$$

Since the sampling processes of l_i and c_j are independent and the stochastic terms are separable, we can decompose the total variance. Let $\mathbf{g}_{\text{EFP}}(\mathbf{W}; l_t, l_i) = \nabla_{\mathbf{W}}(\lambda_1 \mathcal{L}_{\text{EFP}})$ and $\mathbf{g}_{\text{SCP}}(\mathbf{W}; l_t, c_j) = \nabla_{\mathbf{W}}(\lambda_3 \mathcal{L}_{\text{SCP}})$. The total variance is the sum of the variances of the gradient terms corresponding to each independent random variable:

$$\begin{aligned}\sigma^2_{\text{Vanilla}}(\mathbf{W}; l_t) = &\text{Var}_{l_i \sim \text{Uniform}(\mathbf{x}_t)}(\mathbf{g}_{\text{EFP}}(\mathbf{W}; l_t, l_i)) \\ &+ \text{Var}_{c_j \sim \text{Uniform}(C)}(\mathbf{g}_{\text{SCP}}(\mathbf{W}; l_t, c_j)),\end{aligned} \tag{12}$$

where for a random variable $\mathbf{X}$, $\text{Var}(\mathbf{X}) = \mathbb{E}[\|\mathbf{X} - \mathbb{E}[\mathbf{X}]\|^2]$. These two variance terms originate from the two random sampling strategies in the fallibility learning process. If k is the training step, η_k is the learning rate, and $\Delta(\cdot,\cdot)$ represents the update direction vector defined by the optimizer, we can define two optimization:

(1) **Ideal Trajectory**: Driven by the $\mathbf{G}_k(\mathbf{W}) \triangleq \mathbf{G}(\mathbf{W}; l_k)$.

$$\mathbf{W}_{k+1} = \mathbf{W}_k - \eta_k \Delta(\mathbf{W}_k, \mathbf{G}_k), \tag{13}$$

(2) **Stochastic Trajectory**: Driven by the $\mathbf{g}_k(\mathbf{W}) \triangleq \mathbf{g}(\mathbf{W}; l_k, \xi_k)$.

$$\mathbf{W}'_{k+1} = \mathbf{W}'_k - \eta_k \Delta(\mathbf{W}'_k, \mathbf{g}_k), \tag{14}$$

The error in each optimization step can be defined as the weight error between these two trajectories: $\mathbf{E}_k = \mathbf{W}_k - \mathbf{W}'_k$. This weight error exists in every training step and is the source of instability in fallibility representation learning, which we call fallibility jitter.

C Weight Error Upper Bound of Fallibility Jitter

To further investigate the fallibility jitter problem, we derive the upper bound on the cumulative weight error during the optimization process for two mainstream optimizers, Stochastic Gradient Descent (SGD) and Adam, thereby quantifying the impact of fallibility jitter on the representation space.

C.1 SGD as the Optimizer

For standard SGD, the update vector is $\Delta(\mathbf{W}, \mathbf{g}) \triangleq \mathbf{g}$. The single-step evolution of the weight error is:

$$\begin{aligned}\mathbf{E}_{k+1} &= \mathbf{W}_{k+1} - \mathbf{W}'_{k+1} \\ &= (\mathbf{W}_k - \eta_k \mathbf{G}_k(\mathbf{W}_k)) - (\mathbf{W}'_k - \eta_k \mathbf{g}_k(\mathbf{W}'_k)) \\ &= \mathbf{E}_k - \eta_k (\mathbf{G}_k(\mathbf{W}_k) - \mathbf{g}_k(\mathbf{W}'_k)),\end{aligned} \tag{15}$$

Applying the norm and using the triangle inequality, we get:

$$\begin{aligned}\|\mathbf{E}_{k+1}\| &= \left\|\mathbf{E}_k - \eta_k (\mathbf{G}_k(\mathbf{W}_k) - \mathbf{G}_k(\mathbf{W}'_k) + \mathbf{G}_k(\mathbf{W}'_k) - \mathbf{g}_k(\mathbf{W}'_k))\right\| \\ &\le \|\mathbf{E}_k\| + \eta_k \left\|\mathbf{G}_k(\mathbf{W}_k) - \mathbf{G}_k(\mathbf{W}'_k)\right\| + \eta_k \left\|\mathbf{g}_k(\mathbf{W}'_k) - \mathbf{G}_k(\mathbf{W}'_k)\right\|,\end{aligned} \tag{16}$$

We assume the true gradient function $\mathbf{G}_k(\cdot)$ is L-Lipschitz continuous for all k and define the gradient noise as $\boldsymbol{\delta}_k(\mathbf{W}) = \mathbf{g}_k(\mathbf{W}) - \mathbf{G}_k(\mathbf{W})$. Then the above inequality simplifies to:

$$\begin{aligned}\|\mathbf{G}_k(\mathbf{W}_a) - \mathbf{G}_k(\mathbf{W}_b)\| &\le L \|\mathbf{W}_a - \mathbf{W}_b\|, \\ \|\mathbf{E}_{k+1}\| &\le (1 + \eta_k L) \|\mathbf{E}_k\| + \eta_k \left\|\boldsymbol{\delta}_k(\mathbf{W}'_k)\right\|,\end{aligned} \tag{17}$$

where L represents the Lipschitz constant. Assuming a constant learning rate $\eta_k = \eta$ and recursively unrolling this inequality from $k = 0$ to $K - 1$:

$$\begin{aligned}\|\mathbf{E}_K\| &\le (1 + \eta L) \|\mathbf{E}_{K-1}\| + \eta \left\|\boldsymbol{\delta}_{K-1}(\mathbf{W}'_{K-1})\right\| \\ &\le (1 + \eta L) \left[(1 + \eta L) \|\mathbf{E}_{K-2}\| + \eta \left\|\boldsymbol{\delta}_{K-2}(\mathbf{W}'_{K-2})\right\|\right] + \eta \left\|\boldsymbol{\delta}_{K-1}(\mathbf{W}'_{K-1})\right\| \\ &\le \dots \\ &\le \sum_{k=0}^{K-1} \eta (1 + \eta L)^{K-1-k} \left\|\boldsymbol{\delta}_k(\mathbf{W}'_k)\right\|,\end{aligned} \tag{18}$$

Taking the expectation of both sides with respect to all sources of randomness ($\{\xi_k\}_{k=0}^{K-1}$) and applying Jensen's inequality:

$$\begin{aligned}\mathbb{E}[\|\mathbf{E}_K\|] &\le \eta \sum_{k=0}^{K-1} (1 + \eta L)^{K-1-k} \mathbb{E}[\left\|\boldsymbol{\delta}_k(\mathbf{W}'_k)\right\|] \\ &\le \eta \sum_{k=0}^{K-1} (1 + \eta L)^{K-1-k} \sqrt{\mathbb{E}[\left\|\boldsymbol{\delta}_k(\mathbf{W}'_k)\right\|^2]},\end{aligned} \tag{19}$$

Since $\mathbb{E}_{\xi_k}[\left\|\boldsymbol{\delta}_k(\mathbf{W}'_k)\right\|^2 | \mathbf{W}'_k] = \sigma^2(\mathbf{W}'_k; l_k)$, we obtain the final upper bound on the weight error:

$$\mathbb{E}[\|\mathbf{E}_K\|] \le \eta \sum_{k=0}^{K-1} (1 + \eta L)^{K-1-k} \mathbb{E}_{\mathbf{W}'_k}[\sigma(\mathbf{W}'_k; l_k)], \tag{20}$$

This equation shows that the gradient standard deviation $\sigma(\mathbf{W}'_k; l_k)$ at each step is amplified and accumulated into the final weight error, thereby affecting the fallibility representation learning.

C.2 Adam as the Optimizer

The Adam optimizer's update depends on the exponential moving average estimates of the first moment $\mathbf{m}_k$ and the second moment $\mathbf{v}_k$ of the gradients. The gradient noise contaminates both of them.

C.2.1 ***The First Moment Estimate.*** The update rules for the ideal and stochastic momentum are, respectively:

$$\begin{aligned}\mathbf{m}_{k+1} &= \beta_1 \mathbf{m}_k + (1-\beta_1)\mathbf{G}_k(\mathbf{W}_k)\\ \mathbf{m}'_{k+1} &= \beta_1 \mathbf{m}'_k + (1-\beta_1)\mathbf{g}_k(\mathbf{W}'_k),\end{aligned} \tag{21}$$

Define the momentum deviation as $\Delta\mathbf{m}_k \triangleq \mathbf{m}_k - \mathbf{m}'_k$.

$$\begin{aligned}\|\Delta\mathbf{m}_{k+1}\| &= \left\|\beta_1\Delta\mathbf{m}_k + (1-\beta_1)(\mathbf{G}_k(\mathbf{W}_k) - \mathbf{g}_k(\mathbf{W}'_k))\right\|\\ &\le \beta_1\|\Delta\mathbf{m}_k\| + (1-\beta_1)\left(\left\|\mathbf{G}_k(\mathbf{W}_k) - \mathbf{G}_k(\mathbf{W}'_k)\right\| + \left\|\boldsymbol{\delta}_k(\mathbf{W}'_k)\right\|\right)\\ &\le \beta_1\|\Delta\mathbf{m}_k\| + (1-\beta_1)(L\|\mathbf{E}_k\| + \left\|\boldsymbol{\delta}_k(\mathbf{W}'_k)\right\|),\end{aligned} \tag{22}$$

This recursive relationship shows that the gradient noise $\|\boldsymbol{\delta}_k\|$ directly drives the accumulation of momentum deviation, and its influence decays over time with a factor of β_1.

C.2.2 ***The Second Moment Estimate.*** The update of the stochastic second moment $\mathbf{v}'_k$ depends on $\mathbf{g}_k(\mathbf{W}'_k)^2$. Its expectation is:

$$\begin{aligned}\mathbb{E}_{\xi_k}[\mathbf{g}_k(\mathbf{W}'_k)^2] &= \mathbb{E}[(\mathbf{G}_k(\mathbf{W}'_k) + \boldsymbol{\delta}_k(\mathbf{W}'_k))^2]\\ &= \mathbf{G}_k(\mathbf{W}'_k)^2 + 2\mathbf{G}_k(\mathbf{W}'_k)\odot\mathbb{E}[\boldsymbol{\delta}_k(\mathbf{W}'_k)] + \mathbb{E}[\boldsymbol{\delta}_k(\mathbf{W}'_k)^2]\\ &= \mathbf{G}_k(\mathbf{W}'_k)^2 + \mathrm{diag}(\mathrm{Var}(\mathbf{g}_k)),\end{aligned} \tag{23}$$

where $\mathbb{E}[\boldsymbol{\delta}_k] = \mathbf{0}$, and $\mathbb{E}[\delta_{k,i}^2]$ is the variance of the i-th component of the gradient. This means that the update input for the stochastic second moment, $\mathbf{g}_k^2$, is a biased estimator of the ideal update input $\mathbf{G}_k^2$, with the magnitude of the bias being directly related to the gradient variance. T his bias accumulates into $\mathbf{v}'_k$ through the exponential moving average, leading to inaccurate calculations of the adaptive learning rate $\eta/(\sqrt{\hat{\mathbf{v}}'_k} + \epsilon)$ for each parameter dimension.

D Weight Error Upper Bound Analysis for CARL

In this section, we demonstrate that the stochastic gradient $\mathbf{g}_{\text{CARL}}$ of the CARL strategy has a lower variance, thereby mitigating fallibility jitter and enabling more robust fallibility representation learning. This is achieved through two specific mechanisms: variance isolation and negative covariance constraint.

D.1 Variance Isolation

In CARL, the randomness introduced by context sampling l_i is isolated into an independent loss term, $\mathcal{L}_{\text{CL}}$, with a weight of λ_2. In contrast, in the vanilla strategy, the entire EFP task, with a weight of $\lambda_1^{\text{Vanilla}}$, depends on l_i. Assuming that the gradient norms are of a comparable magnitude in both strategies, as long as λ_2 in CARL is smaller than $\lambda_1^{\text{Vanilla}}$ in the vanilla strategy, the contribution to the gradient variance from the sampling of l_i will be reduced:

$$\begin{aligned}\mathrm{Var}_{l_i}(\nabla_{\mathbf{W}}(\lambda_2\mathcal{L}_{\text{CL}})) &\propto \lambda_2^2\cdot\mathrm{Var}_{l_i}(\nabla_{\mathbf{W}}\mathcal{L}_{\text{CL}})\\ &< (\lambda_1^{\text{Vanilla}})^2\cdot\mathrm{Var}_{l_i}(\nabla_{\mathbf{W}}\mathcal{L}_{\text{EFP, Vanilla}}),\end{aligned} \tag{24}$$

D.2 Negative Covariance Constraint

In the CARL strategy, when an anchor $l'_j = l_t \oplus c_j$ is sampled for the SCP task, all other component-injected logs $\{l'_k\}_{k\neq j}$ are used as shared negative samples for other loss terms. When a component c_j is sampled, the stochastic part of the total gradient related to c_j can be decomposed into the sum of two vectors, $\mathbf{Z}(c_j) = \mathbf{A}(c_j) + \mathbf{B}(c_j)$:

- **Primary Term** $\mathbf{A}(c_j)$: Originates from using l'_j as the anchor for the SCP task.

$$\mathbf{A}(c_j) \triangleq \nabla_{\mathbf{W}}(\lambda_4\mathcal{L}_{\text{SCP}}(\text{anchor} = l'_j, \ldots)), \tag{25}$$

- **Covariate Term** $\mathbf{B}(c_j)$: Originates from using the non-anchor injected logs $\{l'_k\}_{k\neq j}$ as shared negative samples for other tasks.

$$\mathbf{B}(c_j) \triangleq \nabla_{\mathbf{W}} \sum_{\text{task}\in\{\text{EFP},\ldots\}} \lambda_{\text{task}}\mathcal{L}_{\text{task}}(\ldots, \text{Negatives} = \{l'_k\}_{k\neq j}), \tag{26}$$

$$\mathrm{Var}(\mathbf{Z}) = \mathrm{Var}(\mathbf{A}) + \mathrm{Var}(\mathbf{B}) + 2\mathrm{Cov}(\mathbf{A},\mathbf{B}), \tag{27}$$

If $\mathrm{Cov}(\mathbf{A},\mathbf{B}) < 0$, this strategy reduces the gradient variance. We define the gradient vectors for each component $c_k \in C$:

- $\mathbf{g}_{A,k}$: The gradient generated when c_k is selected as the SCP anchor. $\mathbf{A}(c_k) \triangleq \mathbf{g}_{A,k}$.
- $\mathbf{g}_{B,k}$: The gradient generated when l'_k is used as a shared negative sample. $\mathbf{B}(c_j) \triangleq \sum_{k\neq j}\mathbf{g}_{B,k}$.

Since the objectives of both $\mathbf{g}_{A,k}$ and $\mathbf{g}_{B,k}$ are to push away l'_k, their gradient directions are highly aligned. Thus, $\{\mathbf{g}_{A,k}\}_{k=1}^N$ and $\{\mathbf{g}_{B,k}\}_{k=1}^N$ are strongly positively correlated.

$$\mathrm{Cov}_k(\mathbf{g}_{A,k}, \mathbf{g}_{B,k}) = \mathbb{E}_k[\mathbf{g}_{A,k}^T\mathbf{g}_{B,k}] - \mathbb{E}_k[\mathbf{g}_{A,k}]^T\mathbb{E}_k[\mathbf{g}_{B,k}] > 0, \tag{28}$$

Let $|C| = N$ and assume the sampling is uniform.

$$\mathbb{E}[\mathbf{A}] = \mathbb{E}_j[\mathbf{g}_{A,j}] = \frac{1}{N}\sum_{k=1}^N \mathbf{g}_{A,k} \triangleq \bar{\mathbf{g}}_A, \tag{29}$$

$$\mathbb{E}[\mathbf{B}] = \mathbb{E}_j\left[\sum_{k\neq j}\mathbf{g}_{B,k}\right] = \frac{1}{N}\sum_{j=1}^N\sum_{k\neq j}\mathbf{g}_{B,k} = \frac{1}{N}\sum_{k=1}^N (N-1)\mathbf{g}_{B,k} = (N-1)\bar{\mathbf{g}}_B, \tag{30}$$

The expectation of the cross-term is:

$$\begin{aligned}\mathbb{E}[\mathbf{A}^T\mathbf{B}] &= \mathbb{E}_j\left[\mathbf{g}_{A,j}^T\left(\sum_{k\neq j}\mathbf{g}_{B,k}\right)\right] = \mathbb{E}_j\left[\mathbf{g}_{A,j}^T\left(\sum_{k=1}^N \mathbf{g}_{B,k} - \mathbf{g}_{B,j}\right)\right]\\ &= \mathbb{E}_j\left[\mathbf{g}_{A,j}^T(N\bar{\mathbf{g}}_B) - \mathbf{g}_{A,j}^T\mathbf{g}_{B,j}\right]\\ &= \mathbb{E}_j[\mathbf{g}_{A,j}]^T(N\bar{\mathbf{g}}_B) - \mathbb{E}_j[\mathbf{g}_{A,j}^T\mathbf{g}_{B,j}]\\ &= N\bar{\mathbf{g}}_A^T\bar{\mathbf{g}}_B - \mathbb{E}_j[\mathbf{g}_{A,j}^T\mathbf{g}_{B,j}],\end{aligned} \tag{31}$$

$$\begin{aligned}\mathrm{Cov}(\mathbf{A},\mathbf{B}) &= \left(N\bar{\mathbf{g}}_A^T\bar{\mathbf{g}}_B - \mathbb{E}_j[\mathbf{g}_{A,j}^T\mathbf{g}_{B,j}]\right) - (\bar{\mathbf{g}}_A^T)((N-1)\bar{\mathbf{g}}_B)\\ &= \bar{\mathbf{g}}_A^T\bar{\mathbf{g}}_B - \mathbb{E}_j[\mathbf{g}_{A,j}^T\mathbf{g}_{B,j}]\\ &= -\left(\mathbb{E}_j[\mathbf{g}_{A,j}^T\mathbf{g}_{B,j}] - \bar{\mathbf{g}}_A^T\bar{\mathbf{g}}_B\right)\\ &= -\mathrm{Cov}_j(\mathbf{g}_{A,j}, \mathbf{g}_{B,j}),\end{aligned} \tag{32}$$

Since $\mathbf{g}_{A,k}$ and $\mathbf{g}_{B,k}$ are positively correlated, i.e., $\mathrm{Cov}_j(\mathbf{g}_{A,j}, \mathbf{g}_{B,j}) > 0$, i.e., $\mathrm{Cov}(\mathbf{A},\mathbf{B}) < 0$ .This proves that the CARL indeed reduces the gradient variance introduced by the component sampling of c_j.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009