LogMoE: Lightweight Expert Mixture for Cross-System Log Anomaly Detection

Jiaxing Qi
Beihang University
Beijing, China
jiaxingqi@buaa.edu.cn

Zhongzhi Luan*
Beihang University
Beijing, China
luan.zhongzhi@buaa.edu.cn

Shaohan Huang
Beihang University
Beijing, China
huangshanhao@buaa.edu.cn

Carol Fung
Concordia University
Montreal, Canada
carol.fung@concordia.ca

Yuchen Wang
Beihang University
Beijing, China
wangyuchen21@buaa.edu.cn

Aibin Wang
Beihang University
Beijing, China
by2406132@buaa.edu.cn

Hongyu Zhang
Chongqing University
Chongqing, China
hyzhang@cqu.edu.cn

Hailong Yang
Beihang University
Beijing, China
hailongyang@buaa.edu.cn

Depei Qian
Beihang University
Beijing, China
depeiq@buaa.edu.cn

***Abstract*—Robust anomaly detection in system logs plays a crucial role in maintaining stable and reliable software operations. However, existing methods often struggle to accommodate evolving log formats and distributional shifts across systems, as they heavily rely on large volumes of labeled data, log parsing, and predefined event templates. To address these challenges, we propose LogMoE, a scalable and parsing-free log anomaly detection framework. LogMoE utilizes labeled logs from multiple mature systems to train a set of lightweight expert models, which are integrated via a gating mechanism within a Mixture-of-Experts (MoE) architecture. This design enables LogMoE to generalize effectively to previously unseen target systems. By eliminating the need for log parsing, our approach remains robust against the heterogeneity of log formats and syntactic structures. We conduct extensive evaluations on eight log datasets under varying generalization scenarios: single-system, homogeneous-system, and heterogeneous-system. Experimental results demonstrate that LogMoE consistently achieves robust generalization, particularly under conditions with scarce labeled data in the target system. As such, LogMoE provides a scalable, parsing-free, and generalization-capable solution tailored for complex and continuously evolving software system environments, positioning it as a future-ready approach to log anomaly detection.**

***Index Terms*—System Logs, Anomaly Detection, Software Reliability**

I. Introduction

Modern large-scale software systems (e.g., cloud services, HPC clusters) generate massive volumes of logs that record runtime states and events. These logs are critical for anomaly detection, helping identify early signs of faults and ensuring high availability and reliability of systems [1]–[4]. However, manual inspection of such logs is labor-intensive and error-prone, making automated log anomaly detection a vital topic in AIOps and system monitoring.

Despite rapid progress, log anomaly detection still faces several challenges. First, supervised approaches [5], [6] often rely on large amounts of labeled anomaly logs, which are costly and time-consuming to acquire, especially for newly deployed or proprietary systems. Second, most existing approaches [7]–[9] depend on log parsing to convert raw logs into structured formats, followed by frequency or sequence modeling. These approaches implicitly assume that log formats and event types remain stable over time [10], [11]. However, log evolution, such as software version upgrades or architectural changes, leads to new or altered log templates, which causes parsing errors, unrecognized events, and unstable sequences [12], [13].

To reduce data dependency, semi and unsupervised approaches detect anomalies by learning patterns from normal data alone. For instance, DeepLog and LogAnomaly [14], [15] use LSTM-based language models to predict the next event and flag low-probability ones as anomalies. However, these models typically rely on accurate log parsing and stable event sequences—assumptions that often break in evolving environments. Parsing errors can propagate through the pipeline and significantly degrade performance [16]–[19]. To mitigate this, recent efforts explore parser-free approaches [20], [21] that operate directly on raw log messages. Yet, many of these approaches still struggle to maintain robustness under log evolution and cross-system scenarios.

To improve the models' generalization and robustness, researchers have introduced transfer learning and domain adaptation techniques to apply models trained on mature systems to target systems. LogTransfer [22] uses GloVe embeddings and LSTM models to extract sequence patterns from source logs and transfer knowledge via shared layers. LogTAD [23] applies adversarial domain adaptation to align log distributions without requiring labeled anomalies. EvLog [12] introduces a multi-level log representation learning framework that avoids parsing and incorporates attention to handle evolving logs. More recently, LogDLR [24] leverages adversarial training to extract domain-invariant features. MetaLog [9] applies MAML-style meta-learning with globally consistent semantic embeddings to achieve strong few-shot adaptation. Nevertheless, most existing approaches rely on partial structural assumptions or require meticulously designed cross-system alignment strategies, which limit their scalability and hinder

*Corresponding author

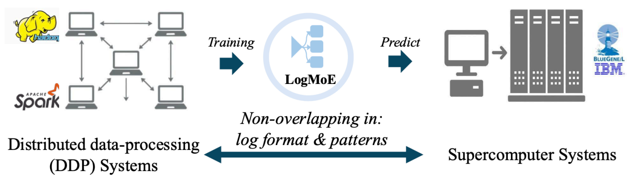

Training
LogMoE
Predict
Distributed data-processing (DDP) Systems
Non-overlapping in: log format & patterns
Supercomputer Systems

Fig. 1: A diagram of our LogMoE. It integrates a set of lightweight experts to enable zero-shot log anomaly detection on unseen systems.

seamless integration into plug-and-play systems.

In light of these limitations, we propose **LogMoE**, a scalable and parser-free framework for cross-system log anomaly detection (see Figure 1). Unlike prior approaches that rely on parsing or require costly labeled data from target systems, LogMoE utilizes a pool of lightweight domain experts, each trained on logs from a mature source system, and dynamically integrates them via a gating-based Mixture-of-Experts (MoE) architecture [25]–[27]. This design offers several advantages: it (1) eliminates dependence on log parsing, (2) supports generalization cross-systems with diverse log formats, and (3) enables zero-shot inference without retraining or fine-tuning on the target domain. The MoE architecture decouples the modeling of system-specific log semantics by assigning each expert to a dedicated source system. During the detection stage, the gating network dynamically selects experts based on input semantics, enabling LogMoE to adapt to post-deployment distribution shifts and log format changes. This mechanism absorbs evolving patterns by routing inputs to relevant experts, supporting cross-system generalization [28], [29]. Moreover, unlike parsing-based approaches [9], [22], LogMoE operates directly on raw-text embeddings, ensuring robustness to syntactic and semantic variations without structural normalization. Each expert is implemented using parameter-efficient Low-rank adaptation (LoRA) [30]–[32], allowing LogMoE to scale to more source systems without incurring significant computational costs. We evaluate LogMoE on eight public log datasets across three generalization scenarios: single-system, homogeneous-system, and heterogeneous-system. Experimental results show that LogMoE consistently achieves competitive or superior performance, even with no labeled logs from the target system. Our contributions are summarized as follows:

- We identify and formalize the problem of heterogeneous-system generalization in log anomaly detection, which remains underexplored in prior work.
- We propose LogMoE, a parser-free, modular framework that reuses labeled logs from mature systems through lightweight, expert-specific adaptation and learnable input-aware expert composition.
- We demonstrate the effectiveness and efficiency of LogMoE across multiple generalization scenarios. Notably, it achieves strong zero-shot performance without requiring any labeled logs or retraining on the target system.

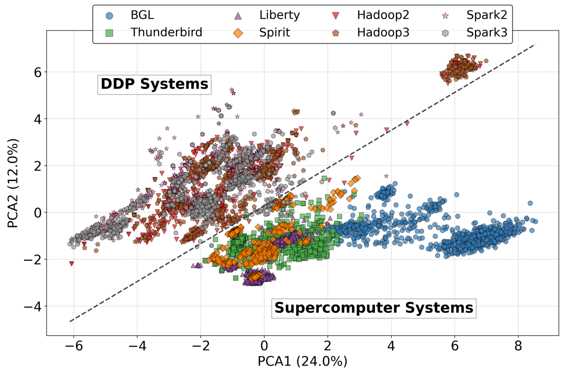


Fig. 2: PCA visualization of logs (BERT embedding) from eight systems. Logs from distributed data-processing systems (Hadoop2/3, Spark2/3) and supercomputing systems (BGL, Spirit, Thunderbird, Liberty) form distinct clusters.

II. Preliminaries

A. Problem Definition

Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ denote a set of log messages, where each x_i is a raw or tokenized log sequence, and $y_i \in \{0, 1\}$ indicates whether the log is anomalous ($y_i = 1$) or normal ($y_i = 0$). In typical log anomaly detection tasks, the goal is to learn a function $f : x \mapsto y$ that can accurately detect anomalies based on previously observed log patterns.

Unlike conventional settings, we focus on a more practical and challenging scenario: heterogeneous-system log anomaly detection. In this scenario, the available labeled data $\mathcal{D}_{\text{source}}$ comes from one or more *source systems*, which have different logging formats, vocabularies, and event patterns from the *target system* to be monitored. The core challenge lies in learning an anomaly detection function that generalizes from $\mathcal{D}_{\text{source}}$ to a target system's log data $\mathcal{D}_{\text{target}}$, where labeled anomalies are extremely sparse or completely unavailable. More formally, let:

- $\mathcal{D}_{\text{source}} = \bigcup_{k=1}^K \{(x_i^{(k)}, y_i^{(k)})\}_{i=1}^{N_k}$ denote logs from K mature source systems, each with sufficient labels
- $\mathcal{D}_{\text{target}} = \{x_j^{(t)}\}_{j=1}^M$ denote the target system log data, assumed to be mostly unlabeled or weakly supervised.

Our objective is to learn a log anomaly detection model $f(\cdot)$ that achieves high performance on the target system $\mathcal{D}_{\text{target}}$, while remaining independent of log parsers or structural normalization. Moreover, the model should be robust to distributional shifts in log formats and capable of adapting to semantic variations across heterogeneous systems.

B. Challenges in Cross-System Generalization

While supervised anomaly detection methods achieve high accuracy within a specific system, their performance often degrades significantly when applied to unseen systems. This is due to the inherent *distribution shift* and *semantic heterogeneity* that exist across logs generated by different software

TABLE I: Examples of Structured Log Entries from Hadoop and BGL Systems

System	Timestamp	Level	Log Message (Component + Content)
Hadoop	2015-10-18 18:01:51,650	INFO	`org.apache.hadoop.mapreduce.v2.app.job.impl.JobImpl: Number of reduces for job job_1445144423722_0020 = 1`
BGL	2005-03-11-02.55.54.739972	FATAL	`RAS KERNEL: instruction cache parity error corrected`

infrastructures. We summarize the main challenges that hinder cross-system generalization for log anomaly detection:

1) Distribution Shift in Log Semantics: Logs differ not only in syntax but also in semantics across systems. For instance, the term *"replica"* refers to data redundancy in distributed data-processing (DDP) systems like Hadoop and Spark, but denotes message duplication in supercomputing systems such as BGL and Liberty. These domain-specific meanings hinder cross-system generalization.

As illustrated in Figure 2, we present the PCA projections [33], [34] of log embeddings sampled from eight systems—four supercomputing systems (BGL, Spirit, Thunderbird, Liberty) [5], [35] and four DDP systems (Hadoop2, Hadoop3, Spark2, Spark3) [12], with 1,000 log events from each. The resulting distributions form distinct clusters, where logs from DDP and supercomputing systems are largely separated into two disjoint regions. This clear boundary between domains highlights a pronounced semantic gap across system types, confirming that even similar log expressions may carry entirely different meanings.

2) Inconsistent Log Formats and Templates: System logs follow semi-structured formats but lack a universal schema. As shown in Table I, log syntax and semantics vary widely across systems due to differences in vendors, infrastructure, and software evolution. Table II further highlights this heterogeneity—e.g., Hadoop2 has over 350K unique tokens, while Spirit has fewer than 5K, and the same parser (e.g., Drain [36]) produces drastically different numbers of templates. Such diversity leads to frequent parsing errors and template explosion, making structured or parsing-based methods brittle under system changes.

3) Anomaly Patterns are System-Specific: Anomalies in different systems manifest in diverse ways. A "missing heartbeat" in a DDP system may be normal in supercomputer logs due to batch scheduling. Thus, anomaly labels do not directly transfer across systems, making supervised training on source domains insufficient for reliable target domain inference.

4) Evolution-Induced Inconsistencies: Even within the same system, software version updates, reconfigurations, or code refactorings may introduce new log templates and modify existing ones. Models relying on fixed template IDs or strict sequential patterns often fail due to these changes.

5) Data Scarcity in Target Systems: In practice, target systems frequently encounter the "cold start" problem, namely, the lack of sufficient labeled anomaly data. However, annotating logs is both time-consuming and costly, often requiring deep domain expertise. This limitation hinders the direct training of reliable models in the target environment.

III. METHOD

A. Overview

To address the challenges of cross-system generalization in log anomaly detection, we propose **LogMoE**, a lightweight and parser-free *Mixture-of-Experts (MoE)* framework. LogMoE leverages labeled log data from multiple mature source systems to train a collection of specialized expert models. These experts are then dynamically integrated via a gating mechanism to adaptively generalize to unseen target systems without requiring log parsing and additional fine-tuning.

As illustrated in Figure 3, LogMoE consists of three main components: 1) **Expert Pool Construction**: We train K lightweight expert models $\{f_1, f_2, \ldots, f_K\}$, each on a labeled source system's logs. These experts share a common backbone encoder (e.g., BERT) and differ only in a small set of trainable LoRAs, significantly reducing model redundancy and enabling efficient deployment. 2) **Gate Module**: Given a new target log message x, the gate module computes a weight distribution $\mathbf{w} \in \mathbb{R}^K$ over the expert pool based on its semantic representation. This enables the framework to softly select and aggregate relevant expert outputs without requiring prior knowledge of the system origin or structure. 3) **Prediction Aggregation**: The final anomaly score is computed as a weighted combination of expert outputs, i.e., $f(x) = \sum_{k=1}^{K} w_k f_k(x)$. This adaptive routing mechanism allows the model to flexibly respond to diverse and unseen log formats by exploiting the complementary knowledge of different experts.

Compared with conventional single-model transfer learning or meta-learning, LogMoE avoids rigid assumptions of domain similarity and reduces overfitting to a single source. Unlike parsing-based approaches, our method directly operates on raw or tokenized log messages, making it robust to system evolution and formatting variance. In the following subsections, we describe the detailed design of each component, including expert model architecture, gate mechanism formulation, and training objectives.

B. Log Message Preprocessing

To ensure robust and consistent input representations across heterogeneous log sources, we adopt a parsing-free preprocessing pipeline comprising four stages.

(1) *Regular Expression Filtering.* Each raw log message x_i is processed with regular expressions to remove system-specific tokens such as numeric values (e.g., IPs and Ports) and punctuations (e.g., colons, quotes). This results in a normalized log event e_i:

$$e_i = \texttt{RegexClean}(x_i). \tag{1}$$

This step eliminates noise and improves the generalizability of learned representations.

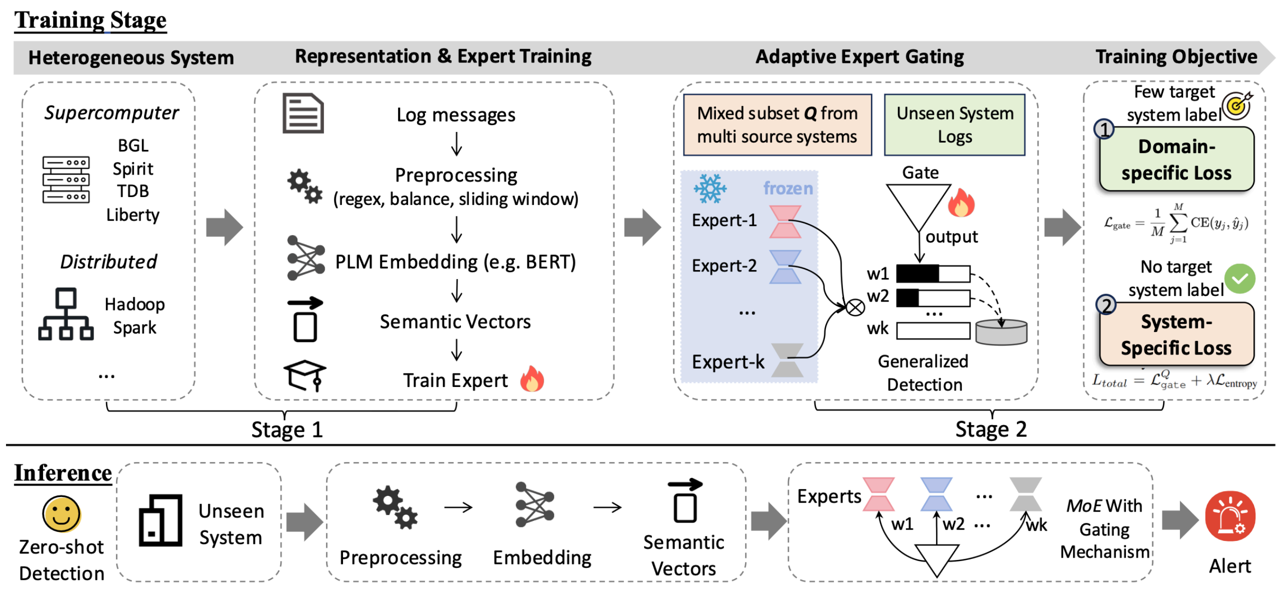


Fig. 3: Overview of the LogMoE pipeline. In Stage 1, expert models are independently trained on labeled logs from multiple source systems. In Stage 2, the gating network is trained using either a small number of labeled logs from the target system or mixed logs from source systems. During inference, the gate dynamically selects relevant experts based on the input representation of target system log sequences.

(2) *Log Balancing.* To address the severe class imbalance, we retain all anomalous samples $\mathcal{A}$ and uniformly downsample the normal logs $\mathcal{N}$ to maintain approximate parity:

$$|\mathcal{A}| \lesssim |\mathcal{N}'|, \quad \text{where } \mathcal{N}' \subseteq \mathcal{N}. \tag{2}$$

(3) *Log Sequence Construction.* Log sequences are then constructed using a fixed-size sliding window of size k, generating log subsequences:

$$S(i) = [e_i, e_{i+1}, \ldots, e_{i+k-1}]. \tag{3}$$

Each subsequence $S(i)$ retains local temporal context and serves as the model input unit.

(4) *Semantic Encoding via Pre-trained Language Model.* Each log event e_i is encoded using a Pre-trained Language Model (PLM). Let $T(e_i)$ be the tokenized form and $H(e_i) \in \mathbb{R}^{L\times d}$ denote its hidden state outputs. The event embedding v_i is computed via mean pooling:

$$v_i = \frac{1}{L}\sum_{j=1}^{L} H(e_i)[j]. \tag{4}$$

These dense semantic vectors are used as the input to subsequent expert modules and gating mechanisms.

C. Expert Model Construction

The expert pool in LogMoE consists of a set of lightweight yet expressive anomaly detection models, each trained independently on a specific source system with sufficient labeled logs. Instead of training separate full models per system, we adopt a parameter-efficient approach by sharing a frozen language backbone across experts while injecting a small number of trainable parameters into each expert through low-rank adaptation. This design reduces storage and computation cost, improves cross-system generalization, and mitigates overfitting under limited labels.

Each expert f_k is built on top of a shared pre-trained BERT encoder $\mathcal{B}(\cdot)$, which encodes the raw or tokenized log message x into a contextualized representation:

$$h = \mathcal{B}(x) \in \mathbb{R}^{L\times d}, \tag{5}$$

where L is the sequence length and d is the hidden size.

To specialize the representation for each system without fine-tuning all BERT parameters, we insert lightweight trainable LoRA into selected layers of the BERT attention blocks. Specifically, for a self-attention weight matrix $W \in \mathbb{R}^{d\times d}$, we replace it with:

$$\hat{W} = W + \Delta W = W + \alpha AB, \tag{6}$$

where $A \in \mathbb{R}^{d\times r}$ and $B \in \mathbb{R}^{r\times d}$ are trainable low-rank matrices, $r \ll d$, and α is a scaling factor. Only A and B are optimized per expert, while W remains frozen. This yields compact, system-specific parameter blocks without modifying the shared backbone.

The output representation h is aggregated through average pooling and fed into a binary classification head:

$$\hat{y}_k = \sigma\left(\mathbf{w}_k^\top h + b_k\right), \tag{7}$$

where $\mathbf{w}_k$ and b_k are the classification parameters unique to expert f_k, and σ is the sigmoid function.

By decoupling the shared backbone from the system-specific parameters, each expert remains compact and efficient, introducing only a small number of additional parameters that scale with the rank of the LoRA. This parameter efficiency allows the expert pool to grow linearly with the number of source systems, enabling scalable knowledge accumulation without overburdening memory or compute. Moreover, the modularity of the architecture ensures that new experts can be trained and integrated independently, without retraining existing components, thereby supporting continuous expansion as new systems emerge. Finally, the LoRA-based adaptation allows each expert to specialize in the semantics and patterns of its corresponding source system, preserving domain-specific knowledge while avoiding catastrophic forgetting—a common challenge in sequential fine-tuning. In the next subsection, we describe how the gate dynamically integrates expert outputs based on input semantics.

D. Mixture-of-Experts with Gating Mechanism

To enable cross-system generalization, LogMoE integrates the multiple expert models through a gating mechanism, which dynamically assigns weights to experts based on the semantic representation of the input log message. This allows LogMoE to adaptively select the most relevant system-specific knowledge for each input without prior knowledge of its origin.

Given a log sequence input x, we first encode it using the shared BERT encoder $\mathcal{B}(\cdot)$ to obtain the hidden representation $h = \mathcal{B}(x)$. The gating module takes h as input and computes a probability distribution $\mathbf{w} = \text{Gate}(h) \in \mathbb{R}^K$ over the K experts, where:

$$w_k = \frac{\exp(\phi_k^\top h)}{\sum_{j=1}^{K} \exp(\phi_j^\top h)} \quad \text{for } k = 1, \ldots, K. \tag{8}$$

Here, each $\phi_k \in \mathbb{R}^d$ is a trainable gating vector representing the affinity between the input representation and expert f_k.

Each expert f_k independently produces an anomaly score $\hat{y}_k = f_k(x) \in [0, 1]$, and the final prediction $\hat{y}$ is computed as a weighted sum:

$$\hat{y} = \sum_{k=1}^{K} w_k \cdot \hat{y}_k. \tag{9}$$

This soft selection mechanism provides LogMoE with adaptive capacity and strong generalization ability. By assigning dynamic weights to system-specific experts based on the semantic embedding of each log input, the gate allows the model to emphasize experts whose knowledge aligns most closely with the input's latent characteristics. This design not only improves flexibility across diverse log formats but also enhances robustness in unfamiliar scenarios, as the gate can distribute weights smoothly when the input falls outside any known source domain. Importantly, because this mechanism relies purely on semantic features and not explicit system identifiers, LogMoE can perform effective cross-system log anomaly detection without requiring domain labels or prior knowledge about the input system.

E. Training and Inference Strategy

The training process of LogMoE is divided into two stages: (1) independent expert training and (2) gate module learning. This modular design improves scalability and makes the framework extensible to new systems.

Stage 1: *Expert Training on Source Systems.* Each expert f_k is trained independently using labeled log data from a source system $\mathcal{D}_k = \{(x_i^{(k)}, y_i^{(k)})\}_{i=1}^{N_k}$. Given the shared BERT encoder $\mathcal{B}(\cdot)$ and LoRA for expert k, we optimize the binary classification loss:

$$\mathcal{L}_{\text{expert}}^{(k)} = -\frac{1}{N_k} \sum_{i=1}^{N_k} \left[y_i^{(k)} \log \hat{y}_i^{(k)} + (1 - y_i^{(k)}) \log(1 - \hat{y}_i^{(k)}) \right], \tag{10}$$

where $\hat{y}_i^{(k)} = f_k(x_i^{(k)})$ is the anomaly score predicted by expert k. During this stage, only the LoRA parameters and expert-specific classification heads are updated, while the BERT backbone remains frozen.

Stage2: *Gate Training.* **Few-shot Training Strategy.** Once the expert pool is trained, we freeze their parameters and train the gating module on a small set of labeled or pseudo-labeled target data $\mathcal{D}_{\text{target}} = \{(x_j, y_j)\}_{j=1}^{M}$. The gating module computes soft attention weights $\mathbf{w}_j$ for each input x_j, and aggregates expert predictions as:

$$\hat{y}_j = \sum_{k=1}^{K} w_{j,k} \cdot f_k(x_j). \tag{11}$$

We optimize the same binary cross-entropy loss:

$$\mathcal{L}_{\text{gate}} = -\frac{1}{M} \sum_{j=1}^{M} \left[y_j \log \hat{y}_j + (1 - y_j) \log(1 - \hat{y}_j) \right]. \tag{12}$$

Zero-shot Training Strategy. In scenarios where no labeled data is available from the target system, we adopt an unsupervised training strategy to optimize the gating mechanism for expert selection.

Given a set of source systems $\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_K\}$, we independently sample subsets of unlabeled log sequences from each source, denoted as $Q_i \subset \mathcal{S}_i$. These subsets are merged into a mixed training batch:

$$Q = \bigcup_{i=1}^{n} Q_i, \tag{13}$$

Each log sequence $x \in Q$ is embedded into a semantic vector v using the shared PLM encoder. The gating network then computes a soft routing distribution $\mathbf{p} = \texttt{Gate}(v) \in \mathbb{R}^K$, where K is the number of experts.

To encourage confident and interpretable expert selection, we adopt an entropy regularization objective:

$$\mathcal{L}_{\text{entropy}} = -\frac{1}{|Q|} \sum_{v \in Q} \sum_{k=1}^{K} p_k(v) \log p_k(v). \tag{14}$$

Minimizing $\mathcal{L}_{\text{entropy}}$ encourages sparse gate outputs, allowing each input to be routed predominantly to a small subset of experts. The final objective is $L_{total} = \mathcal{L}^{Q}_{\texttt{gate}} + \lambda \mathcal{L}_{\text{entropy}}$, where λ is the hyperparameter.

This training strategy is rooted in multi-source domain generalization and MoE learning. Since each expert is trained on logs from a specific source domain, the gate can learn to discover latent clusters in the joint embedding space and assign appropriate experts based on domain-invariant representations. Drawing unlabeled sequences from multiple $\mathcal{S}_i$ further promotes robustness and prevents overfitting to any single system. By learning to assign experts based solely on semantic similarity, without relying on labeled data from the target system, this strategy enables *zero-shot generalization* to unseen systems.

1) Detection stage: Given an input log sequence x, we compute its semantic representation $h = \mathcal{B}(x)$, use the gate to compute weights $\mathbf{w}$, and aggregate expert predictions as described. Since all experts share the same encoder and only a subset of experts is activated per input, the inference remains efficient. This decoupled training strategy allows LogMoE to adapt quickly to new domains with minimal supervision, and to incorporate new source systems incrementally without retraining existing experts. It also makes the framework highly suitable for production environments with evolving and diverse logging infrastructures.

IV. Evaluation

A. Experimental Setup

Datasets. Our evaluation is conducted on eight public log datasets. Four datasets, Liberty, Thunderbird (TDB), Spirit, and BGL, originate from large-scale supercomputing systems and are widely used in prior studies [7], [9], [37]. The other four datasets, Hadoop2, Hadoop3, Spark2, and Spark3, are provided by EvLog [12] and correspond to different software versions of DDP systems. These versioned datasets are specifically used to evaluate the cross-version generalization capabilities of models under software evolution scenarios. Table II summarizes the statistics of these datasets.

Baselines. We compare LogMoE with six representative baselines covering supervised, semi-supervised, transfer learning, and meta-learning paradigms. **LogRobust** [16] is an attention-enhanced supervised model designed to improve robustness against unstable logs. **PLELog** [38] employs semi-supervised learning by iteratively refining pseudo-labels. **EvLog** [12] introduces multi-level representations and attention mechanisms to handle log evolution across software versions. **LogTransfer** [22] and **LogTAD** [23] are transfer-based approaches; the former is supervised and assumes structural similarity between domains, while the latter adopts unsupervised domain adaptation. **MetaLog** [9] utilizes meta-learning to enhance generalization across systems with limited target supervision. These baselines form a comprehensive foundation for evaluating the generalization ability of LogMoE.

Metrics and Implementation Details. Following prior works [12], [22], we adopt standard classification metrics: Precision, Recall, and F1-score, which evaluate overall performance. All methods are implemented in PyTorch. We adopt a frozen BERT-base model as the shared encoder across all experts. For the LoRA modules, we set the rank to $r = 4$. Each expert is trained independently for 100 epochs using the Adam optimizer with a learning rate of 1×10^{-4}. The gating module is trained using 1% anomaly logs from the target system when available; otherwise, it is optimized with an entropy regularization objective with $\lambda = 0.5$. The log sequence window size is set to 100 throughout all experiments.

Experimental Settings. To evaluate generalization capability, we consider three progressive levels of difficulty: i) **Single-system generalization**: Training and testing on different partitions of the same system. ii) **Homogeneous-system generalization**: Transfer across different systems with similar infrastructures or vendors. iii) **Heterogeneous-system generalization**: Transfer across systems with different architectures and vendors. These settings reflect real-world operational environments, where labeled logs are available only for legacy systems, but anomaly detection must be applied to new or evolving systems without or fewer manual annotations.

B. Generalization Scenarios and Results

1) Single-system generalization: As shown in Table III, LogMoE achieves the best performance across all systems, demonstrating strong effectiveness even without distribution shifts. In this setting, LogMoE is reduced to a single-expert model, yet still significantly outperforms both traditional and recent methods. This confirms that the modular MoE design does not compromise performance in single-system scenarios. Although only one expert is used, the gating mechanism is retained for consistency and future extensibility. These results establish a strong lower bound, validating that LogMoE is both accurate and efficient in standard settings.

2) Homogeneous-system generalization: Table IV presents the results when each target system is tested using models trained on other systems of the same category. Compared to the single-system scenario, this scenario introduces moderate domain shifts caused by differences in component configurations and log formats within the same type.

We observe that LogMoE consistently outperforms all baselines across both supercomputer and DDP systems. This demonstrates that its expert modularity and gating mechanism enable effective knowledge transfer between related systems. For instance, when testing on a Spark system, LogMoE can dynamically route inputs to the most relevant expert trained on other Spark variants, yielding better generalization. These results confirm that LogMoE is not only competitive in single-system scenarios, but also adaptable to homogeneous-system variability. However, traditional supervised methods such as LogRobust and PLELog suffer a notable performance drop, indicating limited generalization capacity beyond the training system. Transfer learning methods like LogTransfer and LogTAD improve over them by aligning latent representations across domains, but still face difficulties when handling diverse log semantics. MetaLog and EvLog perform more robustly, benefiting from meta-learning and multi-view modeling.

TABLE II: Statistic of Log Datasets

Dataset	**Logs**	**Anomalous logs**	**Anomaly Ratio**	**Avg. Words/Line**	**Std. Words/Line**	**Vocabulary Size**	**Templates (Drain)**
Supercomputer Systems							
Liberty	10,000,000	3,256,970	32.57%	10.31	2.89	5,839	2,960
Thunderbird	10,000,000	353,794	3.54%	12.39	7.28	8,781	4,992
Spirit	5,000,000	764,890	15.30%	10.12	3.03	4,576	2,880
BGL	4,747,963	348,698	7.34%	10.38	6.56	5,417	1,847
Distributed data-processing (DDP) Systems							
Hadoop2	2,120,739	224,271	10.6%	7.2	3.1	351,529	319
Hadoop3	2,050,488	189,871	9.3%	7.2	3.0	371,440	313
Spark2	931,960	11,055	1.2%	8.3	4.2	43,133	130
Spark3	1,600,273	11,369	0.7%	9.0	5.2	77,527	134

TABLE III: Single-system generalization results. Each model is trained and tested on the same system.

	Supercomputer Systems				*DDP Systems*			
System	Liberty	TDB	Spirit	BGL	Hadoop2	Hadoop3	Spark2	Spark3
Metric	F1/P/R	F1/P/R	F1/P/R	F1/P/R	F1/P/R	F1/P/R	F1/P/R	F1/P/R
LogRobust	0.84/0.85/0.83	0.85/0.86/0.84	0.83/0.84/0.82	0.84/0.85/0.83	0.76/0.77/0.75	0.74/0.75/0.73	0.75/0.76/0.74	0.73/0.74/0.72
PLELog	0.86/0.87/0.85	0.87/0.88/0.86	0.86/0.87/0.85	0.87/0.88/0.86	0.80/0.81/0.79	0.79/0.80/0.78	0.80/0.81/0.79	0.78/0.79/0.77
EvLog	0.91/0.92/0.90	0.91/0.92/0.90	0.90/0.91/0.89	0.91/0.92/0.90	0.86/0.87/0.85	0.84/0.85/0.83	0.85/0.86/0.84	0.83/0.84/0.82
LogTransfer	0.89/0.90/0.88	0.90/0.91/0.89	0.89/0.90/0.88	0.90/0.91/0.89	0.84/0.85/0.83	0.83/0.84/0.82	0.84/0.85/0.83	0.82/0.83/0.81
LogTAD	0.90/0.91/0.89	0.91/0.92/0.90	0.90/0.91/0.89	0.91/0.92/0.90	0.85/0.86/0.84	0.84/0.85/0.83	0.85/0.86/0.84	0.83/0.84/0.82
MetaLog	0.88/0.89/0.87	0.89/0.90/0.88	0.88/0.89/0.87	0.89/0.90/0.88	0.82/0.83/0.81	0.81/0.82/0.80	0.82/0.83/0.81	0.80/0.81/0.79
Ours	**0.99/1.00/0.98**	**0.99/0.99/0.99**	**0.94/0.91/0.89**	**0.93/0.91/0.95**	**0.99/0.99/0.99**	**0.99/0.99/0.99**	**0.96/0.97/0.97**	**0.94/0.91/0.92**

TABLE IV: Homogeneous-system generalization results. For DDP systems, the model is trained on any three of {Hadoop2, Hadoop3, Spark2, Spark3} and tested on the remaining one. Likewise, within supercomputer systems, the model is trained on any three of {Liberty, TDB, Spirit, BGL} and tested on the remaining one.

	Supercomputer Systems				*DDP Systems*			
System	Liberty	TDB	Spirit	BGL	Hadoop2	Hadoop3	Spark2	Spark3
Metric	F1/P/R	F1/P/R	F1/P/R	F1/P/R	F1/P/R	F1/P/R	F1/P/R	F1/P/R
LogRobust	0.72/0.74/0.70	0.73/0.75/0.71	0.71/0.73/0.69	0.70/0.72/0.69	0.66/0.67/0.65	0.67/0.68/0.66	0.69/0.70/0.68	0.68/0.69/0.67
PLELog	0.76/0.78/0.75	0.77/0.78/0.76	0.75/0.76/0.74	0.74/0.75/0.73	0.70/0.71/0.69	0.71/0.72/0.70	0.73/0.74/0.72	0.73/0.74/0.72
LogTransfer	0.81/0.83/0.80	0.82/0.83/0.81	0.80/0.81/0.79	0.79/0.80/0.78	0.75/0.76/0.74	0.76/0.77/0.75	0.78/0.79/0.77	0.77/0.78/0.76
LogTAD	0.83/0.84/0.82	0.83/0.85/0.82	0.81/0.83/0.80	0.81/0.83/0.80	0.76/0.77/0.75	0.77/0.78/0.76	0.79/0.80/0.78	0.78/0.79/0.77
EvLog	0.85/0.86/0.84	0.84/0.85/0.83	0.82/0.83/0.81	0.81/0.82/0.80	0.77/0.78/0.76	0.78/0.79/0.77	0.80/0.81/0.79	0.80/0.81/0.79
MetaLog	0.79/0.81/0.78	0.80/0.82/0.79	0.78/0.79/0.77	0.77/0.78/0.76	0.73/0.74/0.72	0.74/0.75/0.73	0.76/0.77/0.75	0.76/0.77/0.75
Ours (1%-gate tuning)	**0.87/0.79/0.93**	**0.93/0.91/0.94**	**0.91/0.86/0.99**	**0.89/0.87/0.93**	**0.93/0.87/0.99**	**0.94/0.93/0.94**	**0.90/0.88/0.93**	**0.91/0.82/0.97**
Ours (zero-shot)	0.64/0.62/0.69	0.59/0.50/0.69	0.68/0.65/0.69	0.65/0.56/0.77	0.89/0.79/0.99	0.90/0.84/0.97	0.87/0.79/0.96	0.88/0.76/0.98

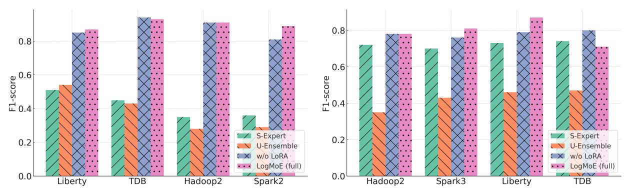
3) Heterogeneous-system generalization: Table V shows the results when models are trained on systems of a different type from the test system, e.g., training on supercomputers and testing on DDP systems, or vice versa. This is the most challenging setting, involving both structural and semantic shifts across system domains.

We observe that traditional supervised and semi-supervised methods like LogRobust and PLELog struggle significantly in this setting due to their reliance on domain-specific patterns and lack of adaptation mechanisms. Even advanced methods such as LogTransfer and LogTAD, though equipped with transfer modules, often assume certain shared distributions or similar feature spaces between source and target, which do not hold in this scenario. Their generalization degrades notably when system behaviors diverge significantly.

LogMoE, in contrast, consistently achieves the best performance across all cross-domain pairs. This validates its design philosophy: by maintaining a pool of system-specific experts and using a learned gating mechanism, LogMoE can select and interpolate relevant expertise even for unseen domains. For instance, when tested on a Spark system after training only on supercomputers, LogMoE dynamically routes inputs through the most semantically aligned expert, rather than relying on rigid shared representations. These results demonstrate that LogMoE uniquely addresses the core difficulty of cross-domain generalization—distribution mismatch—through modular specialization and adaptive routing, setting it apart from all prior methods. *We also compared the performance of LogMoE under different window sizes. The results and discussion are provided in the README at the code link.*

TABLE V: Heterogeneous-system generalization results. The model is trained on four DDP systems (Hadoop2, Hadoop3, Spark2, Spark3) and tested on four supercomputer systems (Liberty, TDB, Spirit, BGL), and vice versa.

	Supercomputer Systems→ DDP Systems				*DDP Systems → Supercomputer Systems*			
System	Hadoop2	Hadoop3	Spark2	Spark3	Liberty	TDB	Spirit	BGL
Metric	F1/P/R	F1/P/R	F1/P/R	F1/P/R	F1/P/R	F1/P/R	F1/P/R	F1/P/R
LogRobust	0.36/0.35/0.37	0.24/0.25/0.23	0.22/0.16/0.34	0.20/0.14/0.35	0.32/0.33/0.31	0.20/0.14/0.35	0.29/0.23/0.39	0.26/0.24/0.29
PLELog	0.11/0.05/0.19	0.28/0.19/0.53	0.19/0.11/0.70	0.15/0.09/0.43	0.34/0.27/0.45	0.18/0.11/0.46	0.33/0.26/0.44	0.21/0.15/0.33
EvLog	0.34/0.27/0.45	0.22/0.16/0.34	0.22/0.15/0.39	0.23/0.21/0.24	0.43/0.35/0.53	0.46/0.34/0.67	0.31/0.24/0.42	0.41/0.33/0.51
LogTransfer	0.71/0.67/0.75	0.73/0.71/0.75	0.76/0.77/0.75	0.75/0.76/0.74	0.76/0.77/0.75	0.49/0.34/0.89	0.75/0.76/0.74	0.74/0.75/0.73
LogTAD	0.63/0.60/0.66	0.68/0.59/0.80	0.71/0.69/0.73	0.66/0.63/0.69	0.71/0.68/0.74	0.47/0.41/0.55	0.68/0.67/0.69	0.65/0.64/0.66
MetaLog	0.73/0.74/0.72	0.72/0.73/0.71	0.72/0.73/0.71	0.70/0.71/0.69	0.70/0.71/0.69	0.71/0.72/0.70	0.69/0.70/0.68	0.68/0.69/0.67
Ours (1%-gate tuning)	**0.78/0.71/0.89**	**0.77/0.73/0.81**	**0.78/0.75/0.83**	**0.71/0.62/0.80**	**0.87/0.78/1.00**	**0.71/0.58/0.92**	**0.78/0.73/0.86**	**0.82/0.83/0.81**
Ours (zero-shot)	0.52/0.43/0.67	0.50/0.39/0.72	0.48/0.39/0.65	0.44/0.43/0.46	0.66/0.52/0.91	0.62/0.48/0.88	0.67/0.63/0.72	0.69/0.67/0.71



(a) Homogeneous-system (b) Heterogeneous-system

Fig. 4: Ablation study under homogeneous-system scenario and heterogeneous-system scenario.

C. Ablation Study

We conduct ablation studies to examine the individual contributions of key components in LogMoE. Specifically, we compare four model variants: (1) **Single-Expert (S-Expert)**, which randomly activates a single expert at inference time without any ensemble or gating mechanism; (2) **Uniform Ensemble (U-Ensemble)**, which averages the outputs of all experts with equal weights, removing the gating mechanism; (3) **LogMoE w/o LoRA**, where each expert is implemented as a fully fine-tuned BERT model instead of using parameter-efficient adaptation; and (4) **LogMoE (full)**, our complete framework that integrates LoRA-based experts with a learned gating mechanism. These variants allow us to isolate the effects of expert diversity, gating strategy, and parameter efficiency on the model's generalization performance.

As shown in Figure 4, the results reveal that incorporating multiple experts leads to substantial gains over using a single expert, highlighting the importance of expert diversity. Moreover, the learned gating mechanism consistently outperforms uniform averaging, indicating the effectiveness of adaptive expert selection in handling system heterogeneity. Notably, LoRA-based adaptation achieves competitive performance compared to full fine-tuning, while offering significant parameter efficiency. These findings collectively validate that both expert diversity and input-dependent composition are critical for achieving robust cross-system generalization.

D. Efficiency and Scalability Analysis

LogMoE is designed to deliver strong generalization capabilities while maintaining lightweight computational overhead. This section evaluates its parameter efficiency, inference latency, and scalability, highlighting the advantages of modularity and low-rank adaptation.

TABLE VI: Parameter comparison (per expert).

Method	Trainable Params	Relative Size
Full BERT (per system)	110M	100%
LoRA Expert	6K	0.006%

TABLE VII: Average inference time (ms / 1000 log sequences).

Method	Time (ms)	Speedup
Full-BERT	978	1×
LogMoE (top-1)	412	2.37×

Parameter Efficiency. In LogMoE, all experts share a frozen BERT-base encoder with approximately 110 million parameters. Each expert introduces a lightweight set of LoRAs (rank $r = 4$), inserted into the final attention layer and the pooling projection layer. This design adds only ∼6K trainable parameters per expert. As shown in Table VI, this enables the deployment of hundreds of domain-specific experts while keeping the total number of trainable parameters still far below that of a fully fine-tuned BERT model. This parameter-efficient setup supports scalable expert expansion without incurring significant memory or computational overhead.

Inference Time. We measure inference latency using batches of 1000 log sequences. The comparison is conducted under two configurations: (i) *Full-BERT baseline*, where each system requires a separate fine-tuned model; and (ii) *LogMoE with top-1 routing*, where only one expert is activated per input through the gate mechanism. As shown in Table VII, LogMoE reduces average inference latency from 978 ms to 412 ms, achieving a 2.37× speedup without compromising accuracy.

Scalability. A key strength of LogMoE is its support for *incremental expert addition* without the need to retrain the entire model. When integrating a new system, only a LoRA and a lightweight classification head are trained. The gate module, composed of a few linear layers over pooled semantic embeddings, can be efficiently fine-tuned on a small set of labeled or pseudo-labeled samples. This modular training paradigm

TABLE VIII: Qualitative comparison of baseline approaches across key design dimensions.

Method	Generalization	Domain Knowledge	Embedding	Log Parsing	Target Labels Usage	Efficiency
LogRobust	Single	No	GloVe	Required	High (100%)	Medium
PLELog	Single	No	GloVe	Required	Medium (1–100%)	High
LogTransfer	Homogeneous	Shared encoder	GloVe	Required	High (target)	Medium
LogTAD	Homogeneous	Adversarial	GloVe	Required	None / Low	Low
MetaLog	Homogeneous	Meta-learning	GloVe	Required	Low ($\sim$1%)	Moderate
EvLog	Homogeneous	No	BERT	No	None	High
LogMoE	Heterogeneous	MoE	BERT	No	None	High

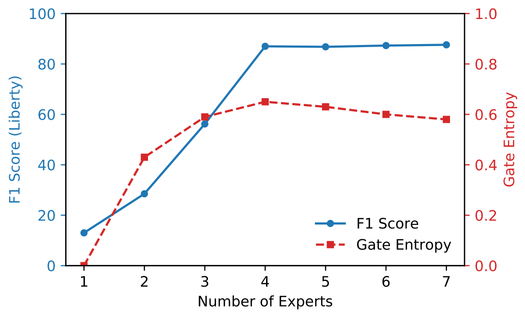


Fig. 5: F1 score and gate entropy on the Liberty as the number of experts increases. The gate entropy decreases after expert expansion, suggesting an increasingly sparse expert selection.

ensures that previously learned experts remain unaffected, facilitating continual learning in dynamic environments.

To evaluate the effectiveness and scalability of LogMoE under adaptive routing, we conduct an expert scaling experiment with soft routing enabled. Specifically, we fix the target system (**Liberty**) and gradually expand the number of source experts from 1 to 7 in the following order: Hadoop2, Hadoop3, Spark2, Spark3, TDB, Spirit, and BGL. After each addition, all existing experts are frozen, and only the gating mechanism is fine-tuned.

As shown in Figure 5, LogMoE exhibits substantial performance improvements as more relevant experts are introduced. The F1 score increases steadily, with Spark3 providing the most significant gain, and further improvements are observed after incorporating Spirit and BGL. Interestingly, although soft routing is used, the gate learns to allocate increasingly selective weights as the expert pool grows. This is reflected by a decrease in gate entropy beyond the fourth expert, indicating a trend toward dynamic sparsity. Such behavior suggests that the gate automatically focuses on a few high-utility experts even without explicit sparsity regularization. Meanwhile, inference latency increases moderately with expert count, confirming that the computational overhead of soft routing remains acceptable. These results validate that LogMoE supports scalable expert composition and efficient adaptation without retraining prior components.

E. Visualization of Gate Weight

To gain a deeper understanding of how LogMoE dynamically integrates knowledge from multiple system-specific experts, we visualize the gate weights assigned to each expert during inference. Figures 6 and 7 present heatmaps of the gate weight distributions under two cross-domain generalization scenarios: *DDP* $\rightarrow$ *Supercomputer* and *Supercomputer* $\rightarrow$ *DDP*, respectively.

For each scenario, we randomly sample 200 log sequences from the test set of each target system. The gate module outputs a probability distribution over experts, which we visualize across these test samples. Each row in the heatmap corresponds to a log sequence, and each column corresponds to an expert trained on a specific source system. The visualizations reveal several key insights:

- **Adaptive Expert Selection.** The gate assigns diverse weights depending on the semantic content of the input log, confirming its ability to perform fine-grained, input-dependent expert routing. For instance, in the *DDP* $\rightarrow$ *Supercomputer* transfer (Figure 6), Spark3 often receives higher attention when the target is Liberty or TDB, indicating semantic similarity among these systems.
- **Cross-System Knowledge Utilization.** Even when the target system differs significantly from any single source domain, the gate assigns weights across multiple experts rather than relying on one, thereby leveraging complementary knowledge from the expert pool. This behavior becomes more evident in cross-domain scenarios, reflecting both the larger distributional shift and the necessity for aggregation across heterogeneous domains.
- **Sparse Activation for Efficiency.** The gate typically produces sharp and sparse distributions, with high probability mass concentrated on a small subset of experts. This sparsity implies that only a few experts are actively involved in each prediction, reducing computational overhead and avoiding unnecessary interference among unrelated domains. Such sparse activation not only improves runtime efficiency but also preserves the independence and specialization of each expert module.

These results demonstrate that the learned gate is both functional and interpretable. It enables LogMoE to generalize across diverse systems by selectively reusing relevant expertise while maintaining modular efficiency. Through compact parameterization, sparse expert activation, and modular expansion capabilities, LogMoE achieves an effective trade-off between generalization power and deployment efficiency.

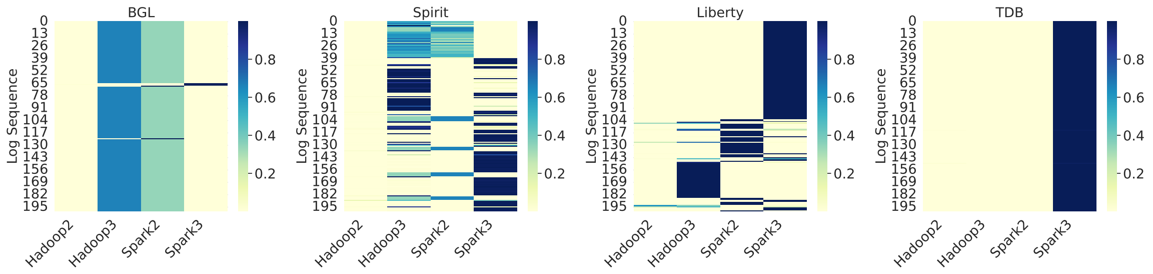


Fig. 6: Heatmap visualization of gate weights in cross-system settings (*DDP* → *Supercomputer*).

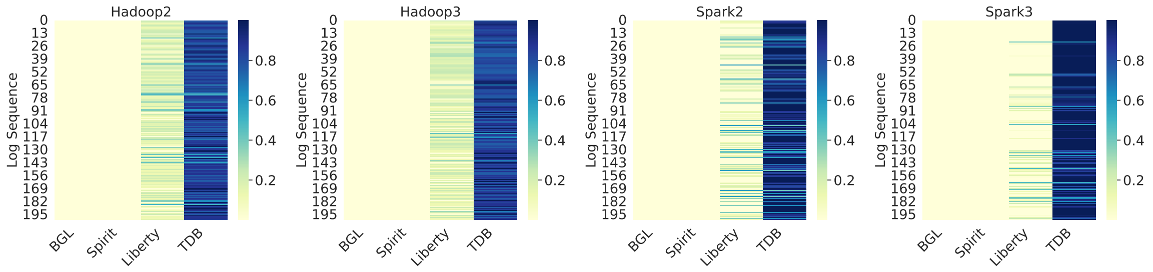


Fig. 7: Heatmap visualization of gate weights in cross-system settings (*Supercomputer* → *DDP*).

F. Qualitative Comparison of Approaches

As summarized in Table VIII, LogMoE presents several distinctive advantages over prior approaches [39]–[41]. First, unlike existing supervised or transfer-based approaches such as LogRobust and LogTransfer, which require extensive labeled data from the target system, LogMoE supports zero-shot anomaly detection. This enables it to generalize to previously unseen systems without requiring any target-side labels, addressing a major limitation in practical deployment scenarios. Second, LogMoE introduces a *Mixture-of-Experts* architecture, where each expert is trained independently on a specific mature system. By leveraging a gating mechanism to assign expert weights based on inputs dynamically, LogMoE effectively captures system-specific patterns while maintaining adaptability across diverse systems. This expert-based design offers greater generalization capability than single-model solutions such as MetaLog or LogTAD, which rely on shared representations and are often sensitive to source-target domain shifts. Third, LogMoE avoids dependency on log parsers and instead operates directly on log sequences using a shared PLM backbone. It injects system-specific knowledge via lightweight parameter-efficient fine-tuning (LoRA), achieving high inference efficiency with minimal additional overhead.

V. Related Works

Log-based anomaly detection approaches can be categorized based on their generalization capabilities into three primary classes: single-system generalization, homogeneous-system generalization, and heterogeneous-system generalization.

1) *Single-system Generalization.* These approaches focus on detecting anomalies within a single system, relying on the system's stable log structures and behaviors. Early approaches, such as DeepLog [14] and LogAnomaly [15], utilize sequential models like LSTMs to capture normal log patterns, flagging deviations as anomalies. Recent advancements aim to enhance the adaptability and efficiency of single-system anomaly detection. For instance, FastLogAD [42] introduces a generator-discriminator framework that generates pseudo-anomalies through mask-guided techniques, enabling the model to learn discriminative features without requiring labeled anomalies. This approach not only improves detection accuracy but also accelerates the training process. While effective within their respective systems, these models struggle to generalize across different systems due to variations in log formats and behaviors [19], [43]–[46].

2) *Homogeneous-system generalization.* These approaches aim to generalize anomaly detection across multiple systems with similar structures or functionalities. LogTransfer [22] applies transfer learning by aligning latent features between source and target systems, enabling knowledge transfer for anomaly detection. MetaLog [9] incorporates a globally consistent semantic embedding module and employs meta-learning to adapt to new systems with minimal labeled data. CroSysLog [37] leverages neural representations and meta-learning to perform log-event level anomaly detection across supercomputing systems, facilitating adaptation with limited labeled data. LogFormer [7] introduces a Transformer-based framework with pre-training and adapter-based tuning stages, improving generalization across multiple domains. MLAD [47] employs Sentence-BERT to capture semantic similarities across log sequences from different systems, using a Gaussian Mixture Model to handle the ”identical shortcut”

problem and enhance cross-system detection. While these approaches have made significant strides in extending anomaly detection capabilities across systems with similar structures or functionalities, they often assume consistent log formats, semantics, and operational behaviors across systems, which is rarely the case in real-world scenarios.

3) *Heterogeneous-system generalization.* These methods address the challenges of generalizing anomaly detection across systems with diverse structures and log formats. To the best of our knowledge, LogMoE is the first work that explicitly targets heterogeneous-system generalization in log anomaly detection. We hope this direction encourages future research toward more general, modular, and robust solutions for log anomaly detection in complex software infrastructures.

VI. Conclusion

We present LogMoE, a parameter-efficient Mixture-of-Experts framework for cross-system log anomaly detection. By combining domain-specialized experts with a learnable gating mechanism, LogMoE enables scalable generalization across heterogeneous systems without requiring full model fine-tuning. Extensive experiments show that LogMoE achieves superior performance and efficiency in both zero-shot and few-shot settings. In future work, we plan to explore automated expert pruning, integrate continual learning to handle log evolution over time, and investigate the combination of LogMoE with LLM-based log understanding to further enhance robustness and interpretability.

ACKNOWLEDGMENTS

This work is supported by Project of National Key R&D Program under Grant 2023YFB3001900 and National Natural Science Foundation of China (No. 62322201, 62072018, U23B2020, and U22A2028)

References

[1] G. Chu, J. Wang, Q. Qi, H. Sun, Z. Zhuang, B. He, Y. Jing, L. Zhang, and J. Liao, "Anomaly detection on interleaved log data with semantic association mining on log-entity graph," *IEEE Transactions on Software Engineering*, 2025.

[2] S. Zhang, Y. Ji, J. Luan, X. Nie, Z. Chen, M. Ma, Y. Sun, and D. Pei, "End-to-end automl for unsupervised log anomaly detection," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pp. 1680–1692, 2024.

[3] Y. Shavit, K. Razmadze, G. Mataev, H. Shteingart, E. Zahavi, and Z. Binshtock, "Semantilog: Log-based anomaly detection with semantic similarity," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pp. 2438–2439, 2024.

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

[5] B. Yu, J. Yao, Q. Fu, Z. Zhong, H. Xie, Y. Wu, Y. Ma, and P. He, "Deep learning or classical machine learning? an empirical study on log-based anomaly detection," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, pp. 1–13, 2024.

[6] M. Landauer, F. Skopik, and M. Wurzenberger, "A critical review of common log data sets used for evaluation of sequence-based anomaly detection techniques," *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 1354–1375, 2024.

[7] H. Guo, J. Yang, J. Liu, J. Bai, B. Wang, Z. Li, T. Zheng, B. Zhang, J. Peng, and Q. Tian, "Logformer: A pre-train and tuning pipeline for log anomaly detection," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, pp. 135–143, 2024.

[8] Z. Li, J. Shi, and M. Van Leeuwen, "Graph neural networks based log anomaly detection and explanation," in *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, pp. 306–307, 2024.

[9] C. Zhang, T. Jia, G. Shen, P. Zhu, and Y. Li, "Metalog: Generalizable cross-system anomaly detection from logs with meta-learning," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pp. 1–12, 2024.

[10] Z. Ma, A. R. Chen, D. J. Kim, T.-H. Chen, and S. Wang, "Llmparser: An exploratory study on using large language models for log parsing," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pp. 1–13, 2024.

[11] S. Abedu, A. Abdellatif, and E. Shihab, "Llm-based chatbots for mining software repositories: Challenges and opportunities," in *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*, pp. 201–210, 2024.

[12] Y. Huo, C. Lee, Y. Su, S. Shan, J. Liu, and M. R. Lyu, "Evlog: Identifying anomalous logs over software evolution," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 391–402, IEEE, 2023.

[13] V.-H. Le and H. Zhang, "Prelog: A pre-trained model for log analytics," *Proceedings of the ACM on Management of Data*, vol. 2, no. 3, pp. 1–28, 2024.

[14] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pp. 1285–1298, 2017.

[15] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun, *et al.*, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs.," in *IJCAI*, vol. 19, pp. 4739–4745, 2019.

[16] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, *et al.*, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the 2019 27th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, pp. 807–817, 2019.

[17] V.-H. Le and H. Zhang, "Log-based anomaly detection with deep learning: How far are we?," in *Proceedings of the 44th international conference on software engineering*, pp. 1356–1367, 2022.

[18] X. Li, P. Chen, L. Jing, Z. He, and G. Yu, "Swisslog: Robust anomaly detection and localization for interleaved unstructured logs," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 4, pp. 2762–2780, 2022.

[19] Z. A. Khan, D. Shin, D. Bianculli, and L. C. Briand, "Impact of log parsing on deep learning-based anomaly detection," *Empirical Software Engineering*, vol. 29, no. 6, p. 139, 2024.

[20] V.-H. Le and H. Zhang, "Log-based anomaly detection without log parsing," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 492–504, IEEE, 2021.

[21] L. Ma, W. Yang, B. Xu, S. Jiang, B. Fei, J. Liang, M. Zhou, and Y. Xiao, "Knowlog: Knowledge enhanced pre-trained language model for log understanding," in *Proceedings of the 46th ieee/acm international conference on software engineering*, pp. 1–13, 2024.

[22] R. Chen, S. Zhang, D. Li, Y. Zhang, F. Guo, W. Meng, D. Pei, Y. Zhang, X. Chen, and Y. Liu, "Logtransfer: Cross-system log anomaly detection for software systems with transfer learning," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, pp. 37–47, IEEE, 2020.

[23] X. Han and S. Yuan, "Unsupervised cross-system log anomaly detection via domain adaptation," in *Proceedings of the 30th ACM international conference on information & knowledge management*, pp. 3068–3072, 2021.

[24] J. Zhou, S. Ying, S. Wang, D. Zhao, J. Xiang, K. Liang, and P. Liu, "Logdlr: Unsupervised cross-system log anomaly detection through domain-invariant latent representation," *IEEE Transactions on Dependable and Secure Computing*, 2025.

[25] M. Le, H. Nguyen, T. Nguyen, T. Pham, L. Ngo, N. Ho, *et al.*, "Mixture of experts meets prompt-based continual learning," *Advances in Neural Information Processing Systems*, vol. 37, pp. 119025–119062, 2024.

[26] W. Cai, J. Jiang, F. Wang, J. Tang, S. Kim, and J. Huang, "A survey on mixture of experts in large language models," *IEEE Transactions on Knowledge and Data Engineering*, 2025.

[27] W. Li, D. Wang, Z. Ding, A. Sohrabizadeh, Z. Qin, J. Cong, and Y. Sun, "Hierarchical mixture of experts: Generalizable learning for high-level synthesis," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, pp. 18476–18484, 2025.

[28] S. Sharma, J. Henderson, and J. Ghosh, "Feamoe: fair, explainable and adaptive mixture of experts," in *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pp. 492–500, 2023.

[29] S. Wu, K. Cao, B. Ribeiro, J. Y. Zou, and J. Leskovec, "Graphmetro: Mitigating complex graph distribution shifts via mixture of aligned experts," *Advances in Neural Information Processing Systems*, vol. 37, pp. 9358–9387, 2024.

[30] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, *et al.*, "Lora: Low-rank adaptation of large language models.," *ICLR*, vol. 1, no. 2, p. 3, 2022.

[31] C. Huang, Q. Liu, B. Y. Lin, T. Pang, C. Du, and M. Lin, "Lorahub: Efficient cross-task generalization via dynamic lora composition," in *First Conference on Language Modeling*.

[32] Y. Mao, Y. Ge, Y. Fan, W. Xu, Y. Mi, Z. Hu, and Y. Gao, "A survey on lora of large language models," *Frontiers of Computer Science*, vol. 19, no. 7, p. 197605, 2025.

[33] A. M. Rekavandi, A.-K. Seghouane, and R. J. Evans, "Learning robust and sparse principal components with the α-divergence," *IEEE Transactions on Image Processing*, 2024.

[34] A. Maćkiewicz and W. Ratajczak, "Principal components analysis (pca)," *Computers & Geosciences*, vol. 19, no. 3, pp. 303–342, 1993.

[35] L. Zhang, T. Jia, M. Jia, Y. Li, Y. Yang, and Z. Wu, "Multivariate log-based anomaly detection for distributed database," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 4256–4267, 2024.

[36] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE international conference on web services (ICWS)*, pp. 33–40, IEEE, 2017.

[37] Y. Wang, M. V. Mäntylä, J. Nyyssölä, K. Ping, and L. Wang, "Cross-system software log-based anomaly detection using meta-learning," *arXiv preprint arXiv:2412.15445*, 2024.

[38] L. Yang, J. Chen, Z. Wang, W. Wang, J. Jiang, X. Dong, and W. Zhang, "Plelog: Semi-supervised log-based anomaly detection via probabilistic label estimation," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pp. 230–231, IEEE, 2021.

[39] C. Almodovar, F. Sabrina, S. Karimi, and S. Azad, "Logfit: Log anomaly detection using fine-tuned language models," *IEEE Transactions on Network and Service Management*, vol. 21, no. 2, pp. 1715–1723, 2024.

[40] Y. Xie, H. Zhang, and M. A. Babar, "Logsd: Detecting anomalies from system logs through self-supervised learning and frequency-based masking," *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 2098–2120, 2024.

[41] C. Zhang, X. Peng, C. Sha, K. Zhang, Z. Fu, X. Wu, Q. Lin, and D. Zhang, "Deeptralog: Trace-log combined microservice anomaly detection through graph-based deep learning," in *Proceedings of the 44th International Conference on Software Engineering*, pp. 623–634, 2022.

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

[43] A. Q. Khan, S. El Jaouhari, N. Tamani, and L. Mroueh, "Knowledge-based anomaly detection: Survey, challenges, and future directions," *Engineering Applications of Artificial Intelligence*, vol. 136, p. 108996, 2024.

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

[45] J. Zhou, Y. Gao, Y. Zhu, X. Yu, Y. Yang, C. Tan, and J. Xiang, "Drllog: Deep reinforcement learning for online log anomaly detection," *IEEE Transactions on Network and Service Management*, 2025.

[46] W. Yuan, S. Ying, X. Duan, H. Cheng, Y. Zhao, and J. Shang, "Midlog: An automated log anomaly detection method based on multi-head gru," *Journal of Systems and Software*, vol. 226, p. 112431, 2025.

[47] R. Zang, H. Guo, J. Yang, J. Liu, Z. Li, T. Zheng, X. Shi, L. Zheng, and B. Zhang, "Mlad: A unified model for multi-system log anomaly detection," *arXiv preprint arXiv:2401.07655*, 2024.