Efficient Zero-shot and Label-free Log Anomaly Detection for Resource-constrained Systems

Zuohan Wu*, Jiachuan Wang†, Libin Zheng‡, Yongqi Zhang*, Shuangyin Li§, Lei Chen*¶
*The Hong Kong University of Science and Technology (Guangzhou), China
zh.wu@connect.hkust-gz.edu.cn, yongqizhang@hkust-gz.edu.cn
†University of Tsukuba, Japan wangjc@slis.tsukuba.ac.jp
‡Sun Yat-sen University, China zhenglb6@mail.sysu.edu.cn
§South China Normal University, China shuangyinli@scnu.edu.cn
¶The Hong Kong University of Science and Technology, Hong Kong SAR, China leichen@cse.ust.hk

***Abstract*—Logs, generated from modern computational systems such as cloud servers or DBMS, are the primary indicator of system states and thus have drawn significant attention from researchers. One of its key tasks is log anomaly detection, aiming to discover anomalous signals that can subsequently imply errors in systems. Conventionally, the major challenge of such a detection task is the insufficiency of well-labeled logs, which require unaffordable human resources. To tackle this issue, recent works have adopted the large language models (LLMs) as zero-shot label-free log anomaly detectors. However, these solutions necessitate direct deployment of LLM instances for downstream tasks, incurring substantial computational costs. Such costs limit their applications in resource-constrained scenarios, which motivates us to reposition the LLMs from direct detectors to training assistants and propose MaidLog. Specifically, MaidLog comprises an LLM-assisted pseudo-labels assignment workflow that automatically generates high-quality labels at training, enabling it to run entirely without manual labels. Based on them, we propose a lightweight detector designed for generalizability, where a well-trained detector is even applicable to downstream systems without any post-training. Once the training is finished, the downstream detection is efficient and only detector-based, without any involvement of LLM. Extensive experiments show that MaidLog effectively handles challenging real-world scenarios. With only a few unlabeled entries, MaidLog achieves comparable performance with a 25,000x speedup over SOTA LLM-centric solutions in out-of-domain systems, demonstrating its applicability on resource-constrained systems.**

***Index Terms*—Anomaly Detection, Log Management**

I. Introduction

Logs generated from computational systems lay the fundamentals for modern Development & Operations, which is now attracting significant interest from researchers. In the domain of log data management, typical tasks can be coarsely categorized into four pipeline levels: logging level [1]–[3], processing level [4]–[12], organization level [13], [14] and application level [15]–[20]. One of the important tasks at the processing level is log anomaly detection, which identifies anomalous log data as indicators of system malfunctions.

In real-world applications, since a delay in discovering the system's error would cause severe economic loss, an anomalous log requires timely detection. Several early research solutions have been proposed under relatively ideal settings [21]–[26]. Specifically, in these conventional frameworks, logs first undergo parsing, a preprocessing step to remove *variables* (e.g., hash strings), before being categorized into *event templates*. Sequentially, these templates are fed to a detector that generates its judgments. This pipeline relies on two key assumptions: (1) **strict assumptions of consistency** to ensure reliable parsing and downstream detection, and (2) **massive label requirements** for training the detector. However, these assumptions are often unrealistic in practice, limiting the broader adoption of such methods.

Improvements have been undertaken to address the aforementioned issues through two lines. The first line concerns accommodating unstable and evolving systems for (1). By approximating log parsing [27], [28] or eliminating the necessity of parsing [29], [30], these methods treat logs similarly to natural languages and achieve high flexibility for diverse and changing systems. In the second line, new solutions aim at performing unsupervised [27], [31], [32] or semi-supervised learning [33], decreasing budgets on anomalous labels for (2). However, these solutions still exhibit generalizability deficiencies, as they typically assume logs always resemble those in the training set. In scenarios such as newly deployed servers or updated running servers, they naturally lose validity. Although several studies employ transfer learning [34], [35] or meta-learning [36] to address this drawback, they still require substantial training samples from new environments or fail when facing significant inconsistency. As both label and consistency issues pertain to model training, we categorize these non-LLM solution limitations as ***training-phase limitations***.

Fortunately, advances in large language models (LLMs) from natural language processing demonstrate potential to overcome previous limitations. Pioneering studies have demonstrated LLMs' significant potential in log analysis [5], [30], [37]–[42]. The majority of them employ prompt engineering [41] or fine-tuning [30], [40], [42] alone with *centric* LLMs to comply with strong generalization and stability in log analysis. For example, LLMs can perform **zero-shot** (without in-domain samples) and **label-free** (without labeled data) anomaly detection for online systems using natural language. Despite some training-free scheme with LLMs that can skip the above training-phase limitations, directly applying LLMs

†Jiachuan Wang is the corresponding author.

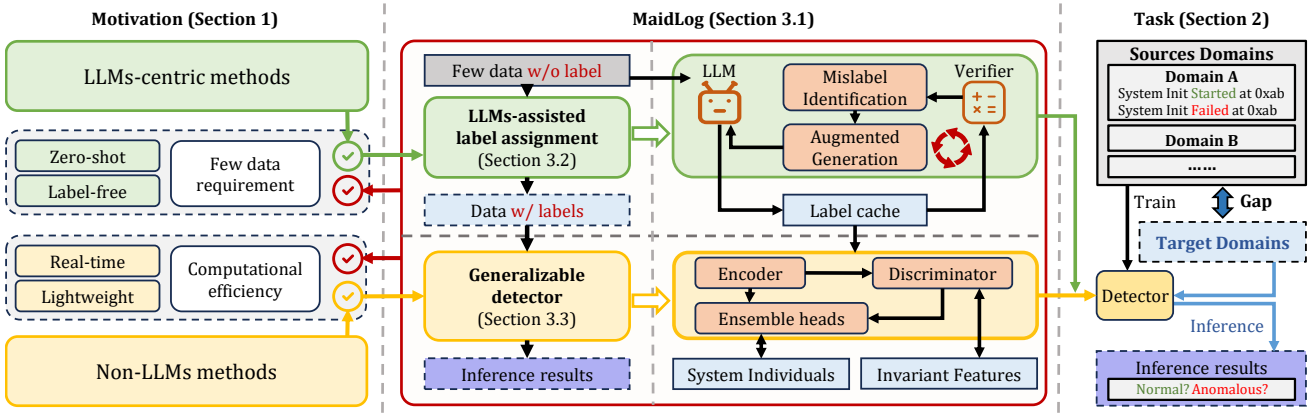

Fig. 1. Overview of MaidLog. The left side introduces our motivation in MaidLog to combine advantages from both LLM-centric and Non-LLMs methods; The middle part summarizes the architecture. MaidLog comprises an LLMs-assisted label assignment (Section III-B) to acquire anomalous labels on unlabeled training data, and a detector with generalizable designs (Section III-C) for zero-shot detection on target systems; The right side overviews the task, where unlabeled log entries in source systems (i.e, domains) are available for training a detector for anomalous identification on unseen target system.

still encounters the following ***inference-phase limitations***:

1) **Efficiency.** Existing *LLM-centric* solutions demand calling LLMs for each inference, causing an unfeasible computational budget and latency, especially for users on light systems/applications like edge devices.
2) **Effectiveness.** Naively adopting LLMs for anomaly identification can only yield suboptimal results [42], [43], since logs are different with the training corpora. Some extant LLM-centric solutions have to furnish LLMs with more in-domain examples, yet are inapplicable when encountered with data scarcity [30], [40], [44]. Moreover, the selection of these examples is non-trivial, and inappropriate demonstrations may degrade performance.
3) **Bias.** LLMs exhibit inherent biases due to varying architectures and training corpora, leading to divergent behaviors across model instances. Prompt sensitivity further worsens this vulnerability, as outputs may differ by prompts. Consequently, identifying both optimal prompts and the most appropriate LLM instance remains challenging.

Given both **training-phase** and **inference-phase** limitations, we are motivated to decouple training and inference processes while combining the strengths of LLM-centric and non-LLM approaches. In this paper, we propose a novel solution named LLMs-assisted Log (MaidLog) to perform anomaly identification with high efficiency and minimal data requirements. As shown in Figure 1 (left), MaidLog focuses on dual functionality: maintaining advantages of LLM-centric methods (zero-shot and label-free operation) while providing lightweight and real-time inference as non-LLM solutions. Specifically, (1) for training-phase limitations, we employ LLMs to automatically generate *massive labels*. Then, those labels will be augmented and utilized to train a lightweight detector without *consistency assumptions*. And, such detector makes the inference phase LLM-free, thus has high *efficiency*; (2) in terms of the inference-phase limitation, LLM in MaidLog would only be utilized for pseudo-label inference during the detector's training, enabling further chances to post-improve before the corresponding training is applied. We further introduce an iterative calibration workflow to prevent the above detector from inheriting the *effectiveness* and *bias* issues. In summary, our contributions are as follows:

- We propose a novel framework, named MaidLog, to perform efficient and effective log anomaly identification in a zero-shot label-free manner in Section III-A by decomposing training and inference runtime.
- We propose an LLM-assisted pseudo-labels generation framework for logs, tackling the limitation of *label requirements* for non-LLM solutions in Section III-B.
- We propose a generalizable and efficient detector for inference in Section III-C with the unique nature of logs.
- We conduct extensive experiments in real-world datasets to demonstrate our motivations and designs in Section IV.

II. Problem Analysis

In this section, we will introduce the existing problem settings and the unsolved challenges.

A. Preliminaries

Logs are generally organized in lines. We denote the i-th log line as x_i, treating it as a natural language sentence. In reality, logs are sequentially arriving, and the demand is to classify whether the system runs normally or not with the logs it generates. Well-designed logging tools enable logs to precisely reflect system states, thereby reducing the task to identifying anomalous logs [45], [46]. This problem can be viewed from two distinct perspectives depending on the granularity:

- **Log anomaly detection**: This task is to diagnose anomalies by full sessions/sequences. Given an ordered batch of logs, the objective is to provide a single answer to the total batch, indicating whether or not there is an anomaly.
- **Log anomaly identification**: A recent trend [47], [48] aims to acquire fine-grained hints from logs, leading to the formulation of this sub-task. Given an ordered batch of logs, this task requires a line-by-line anomaly identification. Our primary focus is on this task as it has finer granularity.

As outlined in Section I, we aim to tackle the most realistic settings, specifically the zero-shot label-free log anomaly identification. It further imposes that (1) no entry and label from target systems (**zero-shot**) and (2) only entries but no

label from source systems (**label-free**) are available during method designing and training. Here, target systems refer to those in downstream inferences, while logs from source systems are already established and documented as prior knowledge. Technically, each system with distinct data and label distributions is referred to as a *domain*. In particular, *systems* are interchangeable with *applications* since domains are logical models. In common practice, a system includes vast applications like databases, a Java backend, etc. Despite their various logging features, they collectively form a complete application, inspiring **a multi-domain modeling**.

Additionally, as previously mentioned in Section I and prior studies [27]–[29], [35], [40], [47], [49], logs may be streaming from multiple cooperative applications concurrently. Consequently, temporally proximate logs may originate from different application sources. This deviates from conventional settings, where researchers presume that the log sequences from the same source can be directly organized by time windows or sessions. Moreover, session organization varies significantly across instances, even within a single system. As MaidLog is desired to be a sufficiently generalizable solution, we avoid the above system-dependent factors and prefer the following standalone entry-level anomaly identification formulated in Section II-B. [48] also adopts such settings for similar reasons, as we will discuss and dive deeper in Section VI.

B. Problem formulation

Referring to log sources (e.g., some applications) as *domains* where each of them retains a distinct joint distribution of logs and anomalous labels, n_d source domains in $\mathcal{D}^s$ and any target domain(s) $\mathcal{D}^t$ are given. In this challenging setting, the training samples consist solely of n_j unlabeled log lines from each source domain D^j, formalized as $\mathcal{D}^s = \{\{x_i^j\}_{i=1}^{n_j} | j = 1, \ldots, n_d\}$. The objective is to identify anomalous log lines in $\mathcal{D}^t = \{(x_i^t, y_i^t)\}_{i=1}^{n_t}$ with a detector trained exclusively on $\mathcal{D}^s$. Formally, our task is to obtain the optimal parameters $\boldsymbol{W}^*$ in a parameterized detector/classifier $cls(\cdot)$ with and only with $\mathcal{D}^s$ to detect anomalies in $\mathcal{D}^t$ as

$$\boldsymbol{W}^* = \arg\max_{\boldsymbol{W}} \mathbb{E}_{(x_i^t, y_i^t) \sim \mathcal{D}^t} \left[\mathbb{P}(cls(x_i^t; \boldsymbol{W}, \mathcal{D}^s) = y_i^t) \right] , \quad (1)$$

where $y_i^t = \{0, 1\}$ denote x_i^t is normal (0) or anomalous (1), respectively, as the right side of Figure 1.

C. Resource constraints

To contextualize the design of MaidLog, we first outline key "resource constraints" encountered in real-world deployments:

- **Computational resource constraint.** Logs originate from diverse systems, ranging from high-performance computing (HPC) clusters to lightweight edge devices (e.g., smart watches, routers). A widely applicable log anomaly detection method should function as an auxiliary module that requires neither GPUs nor high-performance CPUs, and must maintain low computational overhead.
- **Data resource constraints.** As noted in Section I, log data is often sparse. An effective framework should operate in a zero-shot manner without large volumes of in-distribution data, and should be label-free to avoid manual annotation.

Given the strictness of these constraints, balancing detection effectiveness and computational efficiency is critical for practical log anomaly detection.

III. Methodology

In this section, we outline MaidLog. The target of our task and the key ideas of MaidLog are introduced in Section III-A. Then, the pseudo-label generation framework and the detector are proposed in Section III-B and Section III-C, respectively.

A. Principles in MaidLog

Before discussing the details of MaidLog, we will first identify some unique attributes of the task. Training a detector on $\mathcal{D}^s$ appears to be a straightforward unsupervised anomaly detection task, but it becomes a peculiar case in our situation.

What is an anomaly. From the aspect of general anomaly detection, anomalies are defined as those different from the training set. By this definition, representative unsupervised solutions [27], [31], [50] attempt to induce patterns in training sets and define out-of-domain testing samples as anomalous. These methods operate under the assumption that there is no significant distribution shift between training and testing data. However, this assumption is impractical in real applications, such as when the systems are intensively evolved or cold-started [28], [29], [47]. Furthermore, the out-of-domain anomaly detection setting requires that all training samples be strictly normal. This assumption does not hold in our task or in practice, and its violation leads to collapses, as demonstrated in Section IV. Even if one could remove **all** samples with anomalous labels, the sample efficiency would still significantly decrease as an entire category is excluded [49].

Thereby, supervised methods are more suitable for the task, although we do not initially have labels with $\mathcal{D}^s$. Fortunately, the emergence of LLMs provides a promising alternative solution. As demonstrated in recent LLM-centric solutions [40], [41], LLMs can serve as experts and generate pseudo-labels. Although this approach is efficient since it requires no human intervention, the results are not sufficiently accurate. To enhance the quality of pseudo-labels, we introduce an iterative calibration mechanism in Section III-B.

Using the high-quality pseudo-labels obtained, we propose a sophisticated detector tailored to log characteristics in Section III-C, designed to maximize generalization from $\mathcal{D}^s$ to downstream $\mathcal{D}^t$. The complete architecture of MaidLog is illustrated in Figure 1 (center).

B. Pseudo-labels assignment

Recall that LLMs cannot reliably provide high-quality labels through simple zero-shot approaches (e.g., direct question answering), as demonstrated in [40], [41]. Thus, a more sophisticated assignment framework is needed to provide precise training signals and prevent errors (e.g, biases) from accumulating to inherit in the detector. To meet the above requirement, we first consider the following two insights regarding the weakness of LLM:

- **LLMs as black boxes**. It is challenging to ascertain how exactly LLMs identify anomalies. Although sophisticated prompts can establish certain anomaly standards, the interaction mechanisms between LLMs and prompts remain opaque to developers.
- **The absence of domain knowledge**. In general, LLMs lack specialized training on log data and possess limited domain-specific knowledge about the relevant log domains.

The first limitation hinders the evaluation of generated labels. Although LLMs are too complex to be analyzed, conventional parametric classifiers (e.g., the model in Section III-C) can extract more informative measurements from losses and logits. This motivated the implementation of a neural "verifier" to evaluate the labels generated by LLMs. Meanwhile, retrieval-augmented generation has shown potential for addressing the second limitation [40], [44]. Specifically, providing LLMs with exemplars can guide them to generate better assignments than from question-only queries.

In light of the above observations, we propose two key mechanisms: a *mislabel identification mechanism* (Section III-B1) and an *augmented generation mechanism* (Section III-B2). In summary, the fundamental idea is to iteratively discover previously mislabeled samples with the verifier and subsequently calibrate them by providing more pertinent examples to LLMs. The detailed workflow will be introduced later in Section III-B3.

Preprocessing As a deep learning solution, the first step is to embed input logs, whereby variables (i.e., numbers, IP addresses and hash-like values) in logs are cleaned by regexes. Then, the pretrained FastText [51], an efficient and non-GPU embedding method, is utilized to acquire dense representation $e_i^j \in \mathbb{R}^{d_e}$ for each log line. Here, d_e is a fixed vector length depending on the settings of FastText. For the sake of brevity, we attach $\{e_i^j\}$ to corresponding $\{x_i^j\}$ to all domains. The performance of FastText has been validated in RobustLog [28].

Proposed framework The whole assignment process is illustrated in Figure 2. Mislabel identification starts after the initial pseudo-labels are generated for all entries by the LLM. Then, the mechanisms in Section III-B1 will mark which samples are probably mislabeled ("candidate" $\mathcal{D}^c$) and which are more reliably labeled ("support" $\mathcal{D}^p$), respectively. Subsequently, each entry in $\mathcal{D}^c$ will be incorporated with relevant entries in $\mathcal{D}^p$, seeking a better reassignment as detailed in Section III-B2. The above processes interleave for several rounds. Finally, all generated pseudo labels saved in the cache will be summarized into the final outcomes.

*1) **Mislabel identification**:* Deriving ideas from learning with noise [52], researchers have found that noisy labels (i.e., mislabels) typically obtain larger confusions (e.g., losses) in the neural classifier (e.g., multi-layer perceptrons). This phenomenon can be interpreted as the classifier first learning common features before memorizing individual instances [52]. When incorrect labels constitute a minority in the dataset, they can be identified by analyzing classifier behavior patterns based on this phenomenon [53].

Specifically for our task, when using a proper classifier to train a mapping from entries $\{x_i^j\}$ to current pseudo labels, mislabeled samples will not be well-fitted in the early training stage. They typically exhibit either (a) inconsistency between the classifier's outputs and the pseudo-labels, or (b) larger losses compared to other samples in the same class. Therefore, we propose to utilize a neural classifier as a "verifier" with two mislabel identification criteria as follows:

- **Disagreement**. Based on criterion (a), when the pseudo-label and the classification result of an entry are inconsistent, it's potentially mislabeled. A predetermined portion of entries will be sampled from these entries and marked.
- **Confusion**. Based on criterion (b), we sort classification losses within both normal and anomalous categories, respectively, when the pseudo-label and the classification result of an entry are consistent. A predetermined portion of entries with top losses are subsequently marked.

Using these criteria, we identify samples meeting at least one criterion in each domain and class. Due to the log anomaly's categorical imbalance, we standardize the marked sample size across domains and classes to the minimum observed. The union of selected samples, called "candidates" $\mathcal{D}^c = \{\{(e_i^j, \bar{y}_i^j)\}_i\}_j^{n_d}$, contains labels identified as incorrect and requiring calibration.

Recall that we need entries with reliable pseudo-labels for subsequent augmented reassignment. To obtain this reliably-labeled "support set", we extract the complement of the "candidates", denoted as $\mathcal{D}^p$. Specifically, we set a larger predetermined portion when selecting $\mathcal{D}^p$. This ensures $\mathcal{D}^p$ contains relatively more reliable labels than $\mathcal{D}^c$, minimizing the risk that augmented reassignment is misled by noisy examples incorrectly included in the support set.

*2) **Augmented generation**:* To calibrate the candidates $\mathcal{D}^c$, we employ in-context augmented generation using LLMs with three distinct prompt types. All three prompts leverage multiple support samples from $\mathcal{D}^p$ to calibrate the labels in $\mathcal{D}^c$. Details of the three prompts (augmentations) are as follows:

- **Induce**. This prompt guides the LLM to induce and learn patterns from labeled support samples before evaluating the candidate entry. Particularly, *Induce* requires both normal **and** anomalous examples.
- **Filter**. This prompt provides only normal support samples to the LLM, which then filters and determines if the candidate belongs to the normal class.
- **Refine**. This prompt augments the LLM by providing temporally adjacent entries as the context for calibration.

Please refer to Section IV-C for concrete augmentation prompt formats. Hereafter, the remaining challenge is selecting appropriate examples for "induce" and "filter". To ensure robust performance, selected examples must be both **reliable** and **relevant** to each query. The reliability, as described at the end in Section III-B1, has been satisfied as the examples would only be selected in the support set $\mathcal{D}^p$. Then, to ensure relevance, we perform KNN retrieval using entry embeddings e to identify the most semantically similar support samples.

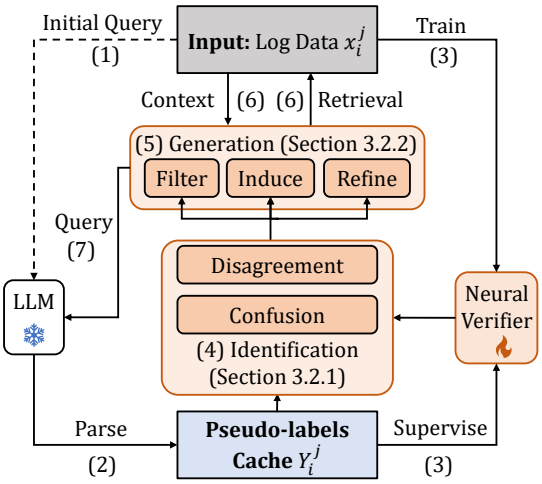

Fig. 2. LLM-assisted pseudo-label assignment. The assignment process iteratively interleaves (5) *mislabel identification* and (4) *augmented generation*. In each round, the neural verifier will identify samples probably with incorrect pseudo-labels in previous rounds (Section III-B1); selected samples will be re-queried to LLMs, seeking example-augmented reassignments (Section III-B2).

For each candidate entry (embedding e) in $\mathcal{D}^c$, we retrieve the top-s_t^j support samples from $\mathcal{D}^p$ using:

$$\text{top-}s_t^j \ \langle e, e_i^j \rangle \, , \bar{y}_i^j = t \, , \tag{2}$$

and extract the corresponding original log entries $\{x_i^j\}$. The sample count s_t^j varies according to the prompting strategy employed. Finally, we format both the query and retrieved examples using natural language templates corresponding to the selected augmentation strategy, then batch them for label reassignment. Here, a query prompt comprises instructions for the anomaly identification task, an augmentation prompt (e.g., "refine") with several example entries, and the target entry.

*3) **Detailed Workflow**:* Figure 2 depicts the whole process of the LLM-assisted pseudo-label assignment. After (1) the initial round of generation, the pseudo-labels are (2) parsed from the LLM's responses, and the neural verifier undergoes (3) supervised training over several startup epochs. Subsequently, the proposed (4) mislabel identification and (5) augmented generation processes are interleaved to ensure continuous calibration. Specifically, (6) depicts the retrieval and organization introduced in Section III-B2, and a round ends after (7) the reassignment queries and (2) the results are parsed. Thereafter, steps (2)-(7) will be repeated until the budget is reached.

Specifically, the verifier adopts the same architecture and objectives as described in Section III-C. In addition, we implement pseudo-label caches to aggregate the results from multiple rounds. For log embedding e_i^j, the pseudo-labels cache will save all the generated labels as $Y_i^j = \{(y_i^j, w_i^j)\}$. While training the verifier, we obtain supervision from:

$$\bar{y}_i^j = \text{round}\left(\frac{\sum_{Y_i^j} w_i^j y_i^j}{\sum_{Y_i^j} w_i^j}\right) = \left\lfloor \frac{\sum_{Y_i^j} w_i^j y_i^j}{\sum_{Y_i^j} w_i^j} - 0.5 \right\rfloor + 1 \, , \tag{3}$$

where $w_\cdot$ will *linearly* grow as the number of iterations. The consideration for such designs is to mitigate premature saturation or later over-calibration, as:

- The pseudo labels are averaged from all rounds of generations, ensuring later opposing reassignment (potentially over-calibration, or noise) will only be in effect after multiple distinct checks. Through that, we expect over-calibration will be rarer.
- Weights will linearly grow by the number of rounds, enabling quick correction in the early and middle calibration rounds. We believe such a design can prevent the calibration from being prematurely stuck.

In expectation, these would bring a "convergence" in assignment result when iteration increases, which is also observed in our experiments. Notably, the LLMs consistently generate *binary* labels (i.e., 0 or 1) in natural language, and thus we align the supervision for the verifier to be rounded as in Equation (3). Eventually, the pseudo-label caches, which accumulate labels $\{Y_i^j\}$ from multiple iterations, provide supervision for the subsequent detector. As a brief preview, the verifier will share the same architecture and training process as the detector, which will be introduced in the next section.

C. Generalizable detector

Once the $\{Y_i^j\}$ from the above process are available, we can subsequently train a supervised detector that provides a categorical answer of normal or anomalous for each log. A typical objective from Equation (1) is:

$$\arg\max_{\boldsymbol{W}} \mathop{\mathbb{E}}_{(e_i^j, \tilde{y}_i^j) \in \mathcal{D}^s} \left(\sum_{t=0,1} y_{ti}^j \log \mathbb{P}\left(cls_{\boldsymbol{W}}(e_i^j) = t \right) \right) \, , \tag{4}$$

where $y_{0i}^j = 1 - \tilde{y}_i^j$ and $y_{1i}^j = \tilde{y}_i^j$ abbreviate the probability of being normal and anomalous respectively. Here, for each log embedding e_i^j, *soft* labels are instead utilized to obtain smooth scores (labels) as $\tilde{y}_i^j = \sum_{Y_i^j} w_i^j y_i^j / \sum_{Y_i^j} w_i^j$.

Recall Section III-A, we desire a better generalizable detector to enable zero-shot downstream transferring. Thereby, as shown in Figure 3, two techniques named *domain generalization* (DG) and *ensemble classifiers* (EL) induced by the nature of logs are subsequently utilized hereafter.

*1) **Domain generalization**:* For a zero-shot classifier, naively adapting Equation (4) is insufficient for generalization towards target domains [54]. This further requires the detector to mine common features across various domains rather than only domain individuals for accessing unseen target domains.

Technically, it proposes a domain generalization task. By constructing the detector $cls(\cdot)$ as $f \cdot g$ as an encoder-predictor structure, a general DG framework tries to align the joint distribution of $P(g(e), y) = P(y|g(e)) \cdot P(g(e))$ across domains [54]. The productions are a domain-invariant encoder $P(g(e))$ and a general head $P(y|g(e))$, where common knowledge is decoupled from domain-specific features. When inferring unseen target domains, the encoder can retain a common feature space and thus validate the downstream head. However, such alignment is far from suitable for logs.

Particularly for logs, the conditional distribution $P(y|g(e))$ is strongly defined by the domain individual and more importantly, by the users' interpretation of anomaly. For instance,

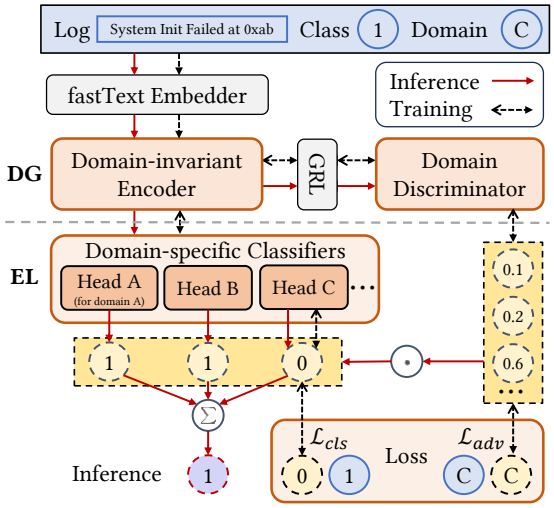

Fig. 3. The framework of the proposed detector. The detector contains three major components, where a domain-invariant encoder and a domain discriminator compose the DG. Multiple domain-specific classifiers (EG) are built on the encoder and output an anomalous score for each entry. Each domain is mapped to a head during training, while all heads are activated and the results are weighted by the discriminator during testing.

a *warning* level log is customary in a developing system, while it may be deemed unacceptable in a highly available system. Given that the same log in different systems may have different semantics, $P(y|g(e))$ can hardly and should not be aligned, while enforcing alignments will hinder the formation of identical and practical $P(g(e))$. Alternatively, exclusive alignment on $P(g(e))$ is sufficient for logs.

Formally, such alignment is to minimize the Jensen-Shannon divergence (JSD) among source domains as

$$\min_{\omega} \sum_{D^i, D^j \in \mathcal{D}^s} \mathrm{JSD}\Big(\mathbb{P}_{e\sim D^i}\big(g_\omega(e)\big) \Big\| \mathbb{P}_{e\sim D^j}\big(g_\omega(e)\big)\Big) . \quad (5)$$

where g is parameterized by ω. This equals optimal in an adversarial minimax game [55] as

$$\arg\min_{\omega}\max_{\phi} \mathop{\mathbb{E}}_{D^j\in\mathcal{D}^s, e\in D^j} \left[\log \mathbb{P}(d_\phi \cdot g_\omega(e) = j)\right] , \quad (6)$$

where $d_\phi(\cdot) \mapsto \mathbb{R}^{n_d}$ is an auxiliary discriminator with parameters ϕ that tries to figure out the belonging domain of the embedding e_i^j. Optimal in Equation (6) yields a "**domain-invariant encoder**" $g_\omega(\cdot)$ and "**domain discriminator**" $d_\phi(\cdot)$, which are the orange modules depicted in Figure 3.

*2) **Ensemble Learning**:* Recall that $P(y|g(e))$ depends on domains. Thus, an identical classification head that projects $g_\omega(e)$ to an anomalous logit is insufficient to reserve domain-specific individuals. Thereby, we adopt ideas from ensemble learning to initialize multiple **domain-specific classifiers** $\{f_{\theta_j}(\cdot)\}_{j=1}^{n_d}$. As shown in Figure 3 (e.g. $j=3$), $f_{\theta_j}\cdot g_\omega(\cdot) \mapsto \mathbb{R}^2$ is designated during forward steps for e_i^j from D_j.

For a quick review, DG and EL aim to capture the homogeneity and heterogeneity among log domains, respectively. Specifically, the homogeneity refers to general knowledge common to logs, such as standard logging levels and the general structure of log entries. Through DG-based alignment across domains, we explicitly reinforce the extraction of this general knowledge by training the encoder to project domains into a shared distribution. Conversely, the heterogeneity defines domain-specific characteristics, such as unique logging components and formats. The data distribution that characterizes this heterogeneity cannot be aligned by DG and would be captured individually per domain by EL.

*3) **Optimization**:* There are two classification-based optimization targets respectively from Equation (4) and Equation (6). Specifically, for gradient-based optimization, we adopt Focal loss [56] as a refined loss function to address imbalances in the training data. Then, Equation (4) is transformed to *categorical classification* objective as

$$\mathcal{L}_{cls} = \mathbb{E}_{(e_i^j,\tilde{y}_i^j)\in\mathcal{D}^s}\left[-\sum_{t=0,1} y_{ti}^j \alpha_t^j (1-p_{ti}^j)^\gamma \log p_{ti}^j\right] , \quad (7)$$

where $p_{ti}^j = \mathbb{P}(f_{\theta_j}\cdot g_\omega(e_i^j) = t)$ and $\{\alpha_0^j, \alpha_1^j\}_{j=1}^{n_d}, \gamma$ denote data-related hyperparameters. Deriving from Equation (6), an additional domain classification objective is:

$$\mathcal{L}_{adv} = \mathbb{E}_{e_i^j\in\mathcal{D}^s}\left[-\mathbb{I}_j(k)(1-p_i^k)^\gamma \log p_i^k\right] , \quad (8)$$

where $p_i^k = \mathbb{P}(d_\phi \cdot \mathrm{GRL}\cdot g_\omega(e_i) = k)$. Here, GRL refers to the gradient reversal layer [57] as

$$(\mathrm{GRL}\cdot f(\cdot))' = -\lambda f'(\cdot)\,, \mathrm{GRL}\cdot f(\cdot) = f(\cdot)\,, \quad (9)$$

which enables direct one-pass gradient descent optimizations in the whole minimax game (Equation (6)). Ultimately, we can learn the detector from source domains by

$$\mathcal{L}(\{\theta_j\},\phi,\omega) = \mathcal{L}_{cls}(\{\theta_j\},\omega) + \beta\mathcal{L}_{adv}(\phi,\omega)\,, \quad (10)$$

where β represents a trade-off hyperparameters. Gradient-based optimization is applied to Equation (10) as shown by black arrows in Figure 3.

*4) **Inference**:* During inferences on target domains (illustrated by red arrows in Figure 3), the discriminator first figures out the relevant scores of each source domain. Subsequently, the corresponding domain-specific heads are activated and the output is the weighted summarization of their opinions as

$$\mathbb{P}(\bar{y}=t) = \sum_{j\in n_d} \mathbb{P}(f_{\theta_j}\cdot g_\omega(e)=t)\mathbb{P}(d_\phi\cdot g_\omega(e)=j)\,, \quad (11)$$

where $t=0,1$. Eventually, we obtain a lightweight detector for downstream tasks that is deployable on edge devices and capable of real-time inference. The computationally intensive component, specifically the LLM-related module, is invoked only once during centralized training. By decoupling training from inference, MaidLog combines the advantages of both LLM-centric and non-LLM approaches as a more practical and efficient solution for resource-constrained systems.

IV. Experiments

In this experimental section, the first three sections detail the implementations, while the following eight sections provide a comprehensive analysis. **Our repository at https://github.com/hehepig4/maidlog includes additional experiments that are omitted in the paper due to page limitations.**

A. Datasets

Our datasets are sourced from two widely used sources: Loghub [58] and Evlog [47]. From Loghub, we select twelve datasets [31], [31], [59], [60], each comprising 2,000 log lines randomly sampled from the examples provided in the repository. Additionally, we include two entry-level labeled datasets from Loghub: *BGL* (BG) and *Thunderbird* (TH), and two from Evlog: *Hadoop2* (H2) and *Hadoop3* (H3) that originate from the same software but different versions.

The experimental setup includes: (1) twelve source domains (2,000 log entries each) for training, and (2) four target domains (BG, TH, H2, H3) for testing, each containing over 2 million entries. Our evaluation is designed under stringent **data resource constraints**: it uses minimal unlabeled training data (significantly less than existing training-based solutions [34], [36]) and ensures no overlap between training and testing domains. Due to the computational infeasibility of processing full datasets for LLM-centric baselines (more than 10 days), we use sampled subsets of BG and TH for testing. Detailed statistics are available in our repository.

B. Baselines

To validate our design motivations, we conduct comprehensive comparisons between MaidLog with the SOTA LLM-centric baselines as follows:

1) **LogPrompt** [41]: Liu et al. proposed a Chain-of-Thought based to execute online log anomaly identification task.
2) **LogGPT** [40]: Hadadi et al. also focuses on prompt engineering. While the original work did not formally name the method, we designate it as LogGPT for clarity. For task adaptation, we modify the input format to process individual log entries and adopt LogGPT's zero-shot template.

Both LLM-centric baselines adopt a relatively similar workflow. Specifically, a log entry will be placed into specialized prompts and then queried to the LLM, which will reason a binary label for each entry. While LogPrompt pays more attention to the feed additional human experiences (e.g., "Mark it abnormal when and only when the alert is explicitly expressed in textual content"), LogGPT defines anomalies more coarsely by "associated with unlikely entries or entries indicating errors, problems, or faults." One may notice that the initial pseudo-label generation round also falls in their procedure, MaidLog obtains a calibration process and removes the generation of LLM from inference time. For fair comparison, we evaluate all methods using identical LLM configurations. Meanwhile, non-LLM solutions, including semi-supervised [33] or supervised [9], [27]–[29], are infeasible as they require substantial labeled data from target domains. For unsupervised approaches, we implement three representative baselines:

1) **DeepSVDD** [50]: This method projects inputs into a latent hypersphere. With assumptions introduced in Section III-A, results far away from the centroid are considered anomalous during inference. This approach has demonstrated effectiveness in session-based log analysis, as evidenced by its adoption in LogBERT [42], etc.
2) **Unsupervised**: We adapt our proposed detector by replacing its objective with DeepSVDD's hypersphere projection.
3) **PCA** [26]: This traditional method detects anomalies through the L2-norm of feature vectors in the principal component space. In aligning our tasks, we modify the log feature vectors to $\{e_i^j\}$.

The unsupervised competitors share a similar preprocessing-embedding-classifying process as MaidLog, despite a different training object. We exclude other unsupervised methods that either rely on session-based assumptions or are incompatible with zero-shot scenarios [27], [31], [32], [35]. There are several intriguing methods yet excluded due to dependence on expert-annotated anchor data [61], [62]. Although transfer learning approaches [34], [36] have been extensively studied, they still require target domain data and are therefore unsuitable for our zero-shot scenario. To further disclose the internal mechanisms of MaidLog, we set up the following ablations:

1) **w/o Iter.**: We substitute the iterative generation process with the initial generation and train the classifier.
2) **w/o Adver.**: Here, we remove the adversarial part of the classifier $\mathcal{L}_{adv}$ to check the improvement from domain generalization by settings $\lambda \equiv 0$ in GRL layer. Particularly, the discriminator is still trained for inferences.
3) **w/o Ensem.**: By altering $\{f_j\}$ with a single classification head, we remove designs for domain individuals.

Particularly, we complement the **Unsupervised** with **Unsur. w/o Adver.** and **Unsur. w/o Ensem.** that shares similar ablative settings above for further validation. All the above baselines obtain the same log embeddings, training data, and neural network structure if they have. We measure performance by Area Under Receiver Operating Characteristics (AUROC) [63] since it serves as a threshold-free metric. AUROC is a larger-better metric, where a score of 1 indicates a perfect classifier (detector) that entirely separates the samples. All the experiments are run with one NVIDIA-A800 GPU and 8 CPU cores.

C. Implementations

We employ the FastText library [64] in Python to encode each preprocessed log line into 300-dimensional vectors. As we obtain 12 source domains in the training set, a total of 12 heads are equipped in the classifier. AdamW [65] implemented by PyTorch [66] is utilized with a learning rate of 0.0004. The optimizer runs for 600 epochs with a batch size of 2000, employing SWA [67] with the same learning rate starting at epoch 360. In the loss objective function, we define

$$\alpha_0^j = n^j/(2\sum_i \tilde{y}_{0i}^j)\,, \alpha_1^j = 1-\alpha_0^j\,, \tag{12}$$

to balance the categories for each training domain and $\gamma = 2$ in Equation (7). For the adversarial component, we progressively increase λ in the GRL layer following [57] as $\lambda = 1 - 2/(1+e^{30k})$,where $k \in (0,1)$ refers the training progress. we set $\beta = 1$ to equally weight $\mathcal{L}_{cls}$ and $\mathcal{L}_{adv}$.

For pseudo-label generation, we evaluate three different LLMs deployed using TGI [68]: Mistral-7B-Instruct-V0.3 [69], Qwen-2.5-7B-Instruct [70], and Llama-3.1-8B-

TABLE I
OVERALL PERFORMANCES MEASURED BY AUROC(↑) WHERE THE TOP-3 ARE BOLD

-	Method	Llama					Qwen					Mistral					-
		H2	H3	BG	TH	Avg.	H2	H3	BG	TH	Avg.	H2	H3	BG	TH	Avg.	AR(↓)
1	LogPrompt	**0.80**	**0.80**	0.68	0.79	**0.77**	0.59	0.58	0.86	0.86	0.72	**0.96**	**0.96**	0.86	0.83	**0.90**	3.958
2	LogGPT	**0.94**	**0.94**	**0.74**	0.78	**0.85**	**0.96**	**0.95**	**0.87**	0.84	**0.90**	**0.88**	**0.89**	**0.87**	0.83	0.87	**2.917**
3	PCA	0.11	0.11	0.24	0.01	0.12	0.11	0.11	0.24	0.01	0.12	0.11	0.11	0.24	0.01	0.12	11.00⋆
4	DeepSVDD	0.50	0.50	0.50	0.49	0.50	0.50	0.50	0.50	0.49	0.50	0.50	0.50	0.50	0.49	0.50	7.583⋆
5	Unsupervised	0.52	0.53	**0.93**	0.03	0.50	0.52	0.53	**0.93**	0.03	0.50	0.52	0.53	**0.93**	0.03	0.50	6.250⋆
6	Un. w/o Adver.	0.32	0.33	0.30	0.33	0.32	0.32	0.33	0.30	0.33	0.32	0.32	0.33	0.30	0.33	0.32	9.500⋆
7	Un. w/o Ensem.	0.47	0.47	0.33	0.30	0.39	0.47	0.47	0.33	0.30	0.39	0.47	0.47	0.33	0.30	0.39	8.917⋆
8	w/o Adver.	**0.83**	**0.82**	0.31	**0.96**	0.73	**0.91**	**0.90**	0.73	**0.96**	**0.88**	0.85	0.85	0.85	**0.96**	**0.88**	**3.625**
9	w/o Ensem.	0.77	0.77	0.45	**0.99**	0.74	0.82	0.82	0.67	**0.95**	0.82	0.79	0.78	0.84	**0.96**	0.84	4.750⋆
10	w/o Iter.	0.76	0.75	0.65	0.87	0.76	**0.89**	0.88	**0.88**	0.73	0.85	0.80	0.79	**0.87**	**0.99**	0.86	4.208⋆
11	MaidLog	0.76	0.76	**0.73**	**0.99**	**0.81**	**0.89**	**0.89**	0.76	**0.92**	**0.87**	**0.94**	**0.93**	**0.88**	0.95	**0.93**	**3.292**

Instruct [71]. The verifier shares the same architectures and hyperparameters as the downstream classifier described above, but without SWA, and the batch size is 200. For the portion mentioned in Section III-B1 and label weights (used in Equation (3)), please refer to our extended hyperparameter analysis in the repository. Finally, for each query, we select: 6 examples for both "refine" and "filter", while 4 normal plus 2 anomalous examples for "induce". All three augmentations are randomly selected and applied to each regeneration query. The total calibration is executed for 42 rounds. The prompt describing the classification task is derived from LogPrompt and omitted due to limited pages (see our repository). Instead, the three instructions used for calibration are as follows:

> ***Induce***: Below are some similar examples with likely pre-assigned categories, which are not always the ground truth and just describe how the system likes. Use them critically to help you make your decision.
> (1) log entry: *log line 1*, category: *label 1* (2) ...

> ***Filter***: Additionally, given below are logs that are likely to be normal. Please use these additions to help you make a more reasonable judgment.
> (1) log entry: *example log line 1* (2) ...

> ***Refine***: Additionally, given below is a consecutive log sequence where the log entry to be judged is in the sequence. Please use this sequence to help you make a more reasonable judgment.
> (1) log entry: *example log line 1* (2) ...

D. Overall performance

Table I summarizes the final results of all implemented methods. The top three results are highlighted in bold font for emphasis. The average AUROC across the four testing datasets is indicated by Avg., while AR refers to the average rank in all twelve settings and ⋆ indicates a significant ($\alpha = 0.1$) performance difference between MaidLog and the baseline in the Wilcoxon signed-rank test.

MaidLog demonstrates improvements in addressing bias and effectiveness issues. Regarding bias mitigation, MaidLog prevents certain failure scenarios (e.g., 0.58 and 0.59 of H2 and H3 when using Qwen and LogPrompt) and exhibits more stable performance across datasets. These results indicate that MaidLog not only improves generalizability but also effectively mitigates the bias issue as introduced in Section I. Furthermore, the comparison between LogPrompt and LogGPT reveals that LLMs exhibit distinct preferences for prompt formats. Notably, Mistral achieves better performance with LogPrompt, whereas Llama and Qwen show the opposite trend. This observation further highlights the bias issue, which will be detailed in Section IV-F. Notice that the results of MaidLog in Table I are with prompt polished from LogPrompt, our method consistently surpasses LogPrompt in terms of average performance (up to $+21\%$ with Qwen).

TABLE II
APPROXIMATE INFERENCE TIME (IT) ON ALL TESTING DATA

Method	LogPrompt	LogGPT	MaidLog	MaidLog CPU
IT	100 h	90 h	**14 s**	**12s**

Unsupervised methods cannot function effectively in label-free settings. As introduced in Section III-A, while unsupervised methods are theoretically designed for label-free settings, they prove impractical in reality due to unrealistic assumptions. Table I (rows 3-7) demonstrates that unsupervised methods consistently underperform. This observation again underscores the rationale behind the utilization of the assignment process in Section III-B. An intriguing result is that the Unsupervised achieves 0.93 at BG. In our discussion of unsupervised methods, it is emphasized that the training dataset must strictly contain only normal samples. Recall that it only classifies by the distance to the latent center, where all training samples are trained to project to the center as "normal". When the normal entries are more similar to the training set than the anomalies, as in BG, it would show such unexpected good results. On the contrary, when the abnormal samples are more familiar to the detector, such as in TH, which is thus projected to the center, it leads to a flawed result.

The proposed method effectively promotes generalizability in zero-shot settings. Rows 8-10 in Table I present the ablation results. First, MaidLog achieves improved performance ($+2.4\% \sim 8.1\%$) using the calibration process described in Section III-B, demonstrating the efficacy of multi-round calibration. Then, when removing the adversarial objective

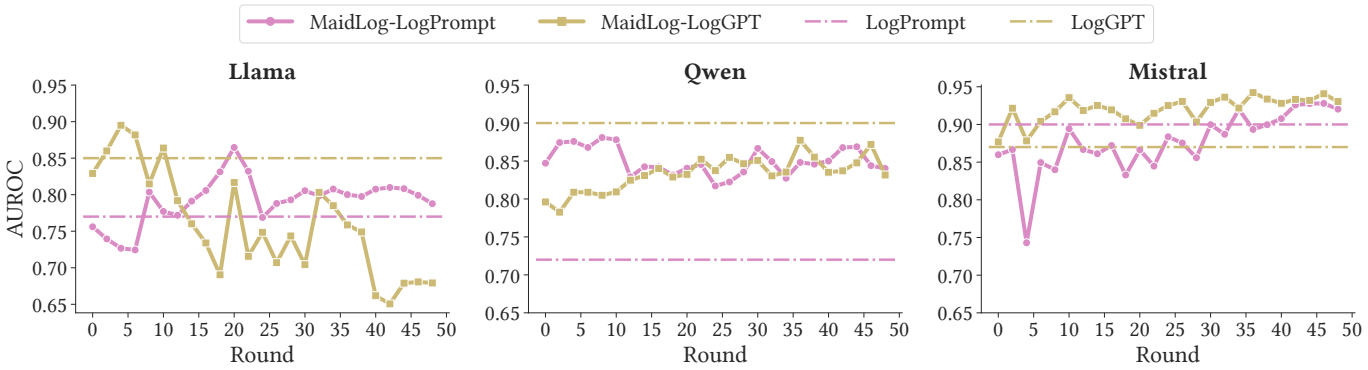

Fig. 4. Performances with different prompts and calibration rounds. The x-axis and the y-axis represent the iterations and the AUROC scores, respectively.

or the ensemble heads, the generalizability is degraded to varying degrees. Specifically, the adversarial designs bring $+10.9\%$ with Llama and $+5.7\%$ with Mistral while only -1.1% with Qwen, revealing the benefits of domain-invariant representations. Meanwhile, the ensemble heads also provide remarkable promotions by $+6.1\% \sim 9.7\%$, showing the necessity to allow domain individuals.

E. Computational efficiency

As shown in Table II, the detector of MaidLog achieves a nearly 25000× speedup, processing over 8 million embeddings in approximately 14 seconds on a GPU. This demonstrates its capability for online real-time log anomaly detection. In contrast to LLM-centric solutions that require large-memory GPUs, MaidLog operates efficiently on lightweight edge devices due to its shallow architecture (only 0.382M parameters).

Furthermore, when deployed with ONNX Runtime [72] using only a CPU, MaidLog processes all test embeddings in 12 seconds in a single forward pass, with a peak memory consumption of 33GB. This suggests that data transfer overhead is a major factor in the GPU inference time. Memory usage can be further reduced by processing test entries in mini-batches.

A single training epoch of the detector requires only 0.37s on an `A800` GPU, utilizing 630MB of GPU memory and approximately 20% of the GPU time. **Overall, MaidLog achieves significant computational efficiency during inference, making it suitable for resource-constrained scenarios.**

F. Influences from prompts and assignments

Observing divergent performances between LogGPT and LogPrompt prompts on tested LLMs, we modified our prompts and calibration rounds of the assignment process to report the corresponding results for a fair test. The results are in Figure 4, where "MaidLog-LogPrompt" and "MaidLog-LogGPT" denote where the prompt is modified from, respectively. Throughout the calibration process, both Qwen and Mistral exhibit consistent performance improvements across different prompt formats. Moreover, the performance disparity between prompts decreases over calibration rounds, effectively mitigating prompt-induced bias. Since different LLMs respond inconsistently to different prompts, our calibration process is designed to reduce this inherent uncertainty rather than identify a single optimal prompt pair.

The only exception occurs with Llama using LogGPT prompts, where the calibration curves exhibit an initial upward trend followed by a decline. Although early calibration rounds yield strong improvements over LLM-centric solutions, additional rounds eventually degrade performance. We derive two key insights: (1) stronger LLMs (e.g., Mistral) benefit more from our approach, while Llama performs the worst among the three; (2) excessive calibration rounds may introduce noise. Thus, we recommend using stronger LLMs (e.g., Mistral) for stable performance or determining optimal calibration rounds via validation on target applications. Although they are common tricks in general settings or applications, we don't tune them due to the strictly invisible target domain settings.

G. Efficiency and effectiveness on pseudo-labels assignment

To comprehensively disclose the pseudo-labels assignment process, we further conduct the proposed process with labeled samples from H2 and H3. Using the first 12,000 samples from both datasets, we execute the same pseudo-label assignment process as for the training set over 60 rounds without reinitializing the verifier. Corresponding results illustrated in Figure 5 are averaged among three LLMs. First, the pseudo-labels show consistent improvement across 60 calibration rounds, with recall (shaded area) increasing from 0.35 to 0.92. Meanwhile, the precision is improved from 0.38 to 0.68 in 52 rounds. As the anomalies only share 1.1% in the samples, improvements in both precision and recall indicate the corrections have happened to both mislabeled normal (false negative) and anomalous (false positive) logs. It's observed that the precision undergoes a slight decline during subsequent final rounds, indicating an escalation in false positives. This aligns with the performance deterioration observed when using Llama with MaidLog-LogGPT in Figure 4, where noise occurs due to inappropriate over-calibration in MaidLog.

For better clarity, we export the intermediate statistics in Table III, where each number denotes the average result among the corresponding rounds. The observations are as follows:

- As expected, **the verifier can effectively identify mislabeled candidates**, whose error rates ($> 30\%$) are higher than those remaining support set ($< 0.2\%$). Thereby, our statement in Section III-B1 is empirically demonstrated.
- **The augmented generation could correct the selected mislabels**, bringing consistent improvements (F1: $0.44 \rightarrow$

TABLE III
AVERAGE STATISTICS AMONG THE ITERATIONS AS IN FIGURE 5

Number of Iterations	1-20	21-40	41-60
Error rate in candidate set $\mathcal{D}_c$	32.4%	**35.0%**	31.3%
Error rate in support set $\mathcal{D}_s$	0.2%	0.2%	**0.03%**
Average F1 score in all samples	0.44	0.52	**0.70**

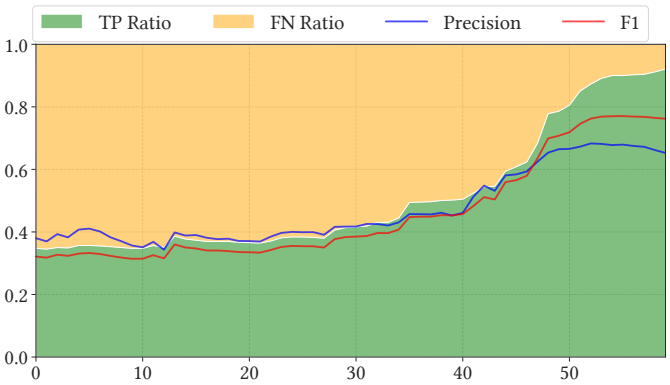

Fig. 5. Improvements throughout assignment iterations, where TP and FN respectively denote true positive and false negative.

0.70) as anticipated in Section III-B2.

Additionally, the contributions of the three augmented prompts (i.e., induce, filter, and refine) are presented in Table IV, **showing that all three prompts improve the final label assignment**. This demonstrates our approach of fully leveraging LLM through diverse prompting strategies.

To demonstrate applicability for resource-constrained users (e.g., those with limited LLM access during training), we also report the total LLM inference calls. For the improvements shown in Figure 5, Mistral, Qwen, and Llama require 5,305, 16,532, and 18,089 additional inference calls across all 60 rounds, respectively. **MaidLog requires only 55% more calls (on average) across all 60 rounds while achieving significant label quality improvements, demonstrating strong applicability for resource-constrained users.**

Additionally, we also try to see if the LLM itself can act as an in-place mislabel verifier by outputting an extra confidence score ($[0, 1]$) to its response. Based on the results of a subset from H2 with Mistral, it's found to be inapplicable since (a) the confidence distribution is highly skewed, where 90.73% of all samples and 45.37% of mislabeled samples get confidence 1.0; (b) there is only a weak positive relationship between higher confidence and more accurate assignments (AUROC 0.73), even a negative correlation for those anomalous entries (AUROC 0.32). Overall, the confidence of LLMs cannot imply the correctness of pseudo-labels, underlining the motivation of the auxiliary verifier.

H. Benefits from multiple domains

One may also be interested in the multiple-domain settings. To demonstrate the improvement achieved by incorporating knowledge from multiple domains, we first rank each training domain's contribution based on its average weights from the discriminator across test domains. Then, we progressively remove less contributing domains (with their corresponding classification heads) while maintaining the same optimization steps. As shown in Figure 6, the average performance is improved when more domains are involved, except for a slight degradation on BG. Overall, consistent with the improvements observed in ensemble learning and domain generalization designs during ablative experiments, this demonstrates that MaidLogenhances generalization ability through explicit multi-domain settings. We also visualize the features to disclose from a parametric view, which illustrates **a better in-distribution embedding space** with DG and can be found in the repository. Meanwhile, unexpectedly observed embedding clusters, indicating that **the encoder cannot distinguish these entries, occur in the test entries without alignment (DG)**, whereas there are none with DG. **As discussed in Section III-C, leveraging the homogeneity and heterogeneity can improve the generalization of the log anomaly detector.**

TABLE IV
ABLATIVE RESULTS AT 60 ITERATIONS

Used prompts	All	- Induce	- Filter	- Refine
F1 score	**0.80**	0.65	0.66	0.27

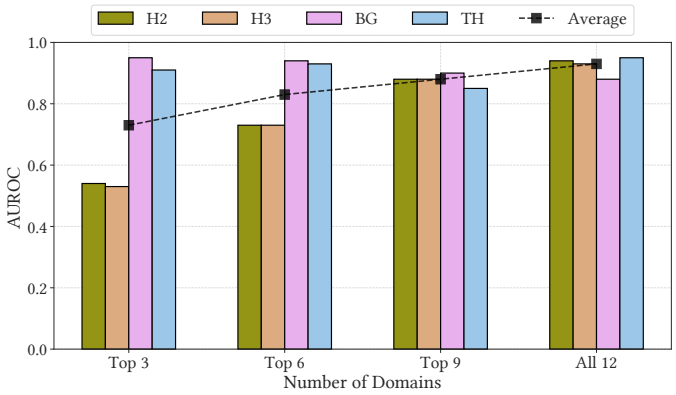

Fig. 6. AUROC when domains are removed with the Mistral, where Top-x denotes that a remaining x domains are utilized during training the detector.

I. Case studies on pseudo labels

We further conduct a case study to empirically evaluate the pseudo labels on the training set with Mistral, including the following observations:

- **The anomalies that have general and clear keywords are well recalled.** For instance, entries containing explicit abnormal keywords (bold), like "psu **`failure`** ambient=28" or "mtalk.google.com:5228 **`error`**: Could not connect through proxy proxy.cse.cuhk.edu.hk:5070 - Proxy server cannot establish a connection with the target, status code **`403`**", are well recalled as anomalies.
- **The pseudo labels are less precise for logs describing ambiguous events.** Certain entries, such as "PacketResponder 2 for block blk_-6670958622368987959 **`terminating`**" and "authentication **`failure`**; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=65.166.159.14" (whether a failed SSH login is considered an anomaly or not, depending on the user cases), are marked as anomalous. Such ambiguity also explains higher recalls than precisions in Figure 5. **As a consequence, MaidLog trains the detector with soft labels where the ambiguous ones can obtain intermediate states and are distinguished from more abnormal ones.**

TABLE V
AUROC WITH VARIOUS ARCHITECTURES WHERE THE TOP-3 ARE BOLD

-	Basic method	Embedder	Classifier (Params.)	Avg.
a	**MaidLog**	FastText	Proposed (**0.3M**)	**0.93**
	MaidLog	FastText-r	Proposed (**0.3M**)	0.88
b	MaidLog	BERT	Proposed (0.5M)	0.60
	MaidLog	RoBERTa	Proposed (0.5M)	**0.93**
	MaidLog	BART⋆	Proposed (0.5M)	0.37
c	**NeuralLog**	BERT	Trans. block (31.5M)	**0.93**
	NeuralLog	RoBERTa	Trans. block (31.5M)	0.65
	PreLog	BART⋆	Trans. decoder (134.7M)	**0.96**

J. Hyperparameter analysis and deployment guidelines

To evaluate the stability of the proposed detector with respect to hyperparameters, we adjust β in Equation (10) and γ in Equation (7) to test if the proposed detector is stable. Due to the page limitation, the detailed results can be found in our GitHub repository. For a short summarization, The detector exhibits stable performance with general γ and β values that are of non-extreme value. For deployment, we suggest setting $\gamma = 2$ and $\beta = 1$. And, $\{\alpha.\}$ are class balancing parameters set by Equation (12). These settings confirm the stability of the generalizable detector with respect to its hyperparameters.

For the hyperparameters during pseudo-label assignment, we have varied the following three key factors and observed the improvements of pseudo-labels in most cases. In short, the resulting principles for deployment are as follows: (a) keep the weights w_i in Equation (3) roundly increasing; (b) roundly decrease the candidate portion and (c) set a lower than 0.5 evidence portion in Section III-B1. With the above settings, one can obtain satisfactory pseudo-labels. Still, it's recommended to adopt our primary, verified implementation in Section IV-C at first.

K. Alternative preprocessors, embedders, and detectors

We further analyze MaidLog with alternative components, as summarized in Table V. Specifically: (a) *FastText-r* uses no preprocessing (as in PreLog [30]) and employs FastText as the embedder; (b) replacing the embedder with BERT [73] and RoBERTa [74] (as in NeuralLog [28]), and BART⋆ (a post-trained BART [75] in PreLog). (c) Meanwhile, we include NeuraLog and PreLog as baselines, which are applicable and modified for entry-level identification. Unlike the shallow detector with FastText in MaidLog, these baselines employ BERT and BART⋆ with a transformer encoder block and an entire transformer decoder as detectors, respectively. Notice that these are supervised methods, with training labels generated via our assignment process based on Mistral.

We have the following observations. **Firstly, the preprocessing step is helpful in these settings.** Secondly, FastText performs comparably to the best pretrained transformers (i.e., RoBERTa). **Noticing that FastText doesn't require a GPU or high-performance CPU, it's a more attractive choice under computational resource constraints.** Ultimately, MaidLog shows comparable performance to NeuralLog with significantly fewer parameters. Although PreLog yields the highest AUROC (+0.03 compared to MaidLog), it is less attractive under computational constraints: the official implementation requires over twenty hours to infer all test entries. **To summarize, the components within the current MaidLog implementation strike an excellent balance between performance and efficiency.** Please refer to the repository for extended results.

V. RELATED WORKS

In terms of automated log data management, related tasks can be coarsely categorized into four hierarchical levels:

1) **Logging level**: Comprehensively log system states to enable further utilization of logs. Related studies focus on logging techniques for specific scenarios [1]–[3], etc.
2) **Processing level**: Analyze log attributes or organize raw log entries into structural formats, e.g., structure extraction [4], parsing [5]–[8], anomaly detection [9]–[12], etc.
3) **Organization level**: Store and organize processed log entries for further access. Specifically, this level supports accessing massive logs in a database-like manner for various scenarios, such as cloud log engines [13], [14], etc.
4) **Application level**: Mine knowledge from logs and utilize it for system optimization. Unlike the previous levels, the application level includes a vast number of diversified tasks such as user profiling [15], [16], large system diagnosing [17], query optimization [18], natural language to SQL [19], or error recovery [20], and so on.

The primary objective of all levels is to diagnose system runtime comprehensively with logs. Subsequently, we review related studies from the following two technical lines:

Conventional Log Data Management and ML in Log Data Management. For the primary parsing step, numerous studies, including Drain [76], Spell [7], Prefix-Graph [6], etc., are proposed and build fundamentals for sequential analysis. In particular, Zhu et al. [77] provided a benchmark suite and tested various parsing solutions, while Song et al. [12] proposed a pipeline for anomaly detection development. Based on parsers, PleLog [33], DeepLog [31], MoniLog [10], SpikeLog [9] and [11], etc., have been derived to conduct anomaly detections based on parsed event sequences. The majority of them utilized ML techniques like recurrent neural networks, meta-learning [36], and domain adaptation [34], pioneering the power of data-driven automated log analysis. Meanwhile, researchers point out that parser-based log analysis always suffers from parsing errors, especially when log format changes [40], [47], inspiring efforts on parser-free methods. This is mainly done by treating logs as semi-structured natural language, utilizing tools like Fasttext [51] to embed logs. Then, for instance, RobustLog [28] utilizes LSTM [78] to conduct supervised anomaly detection.

Log Data Management and Mining in the Era of LMs & LLMs. As LM shows remarkable generalizability, researchers are devoted to exploring their power in log data management and mining. Firstly, for log anomaly tasks, early trials include NeuralLog [29] and LogBERT [42] take BERT [73] as a log encoder, then utilize Transformers [79] decoder and prediction scores to detect anomalies, respectively. Additionally,

EvLog [47] and PreLog [30] follow up and extend bonus mechanisms, such as auxiliary clustering and contrastive learning, achieving remarkable performance. After the power of even larger LMs has been certified, LLMs for log data mining have also been preliminarily discovered. Specifically, Zhong et al. [39] propose a mixed log parser with LLM, demonstrating the potential of LLM in the area of Log. Concurrently, Ma et al. [38] and Liu et al. [80] also propose their competitive solution for the same task. Particularly, as LLMs show excellent adaptability to multiple tasks, researchers also explore all-in-one management with LLMs, e.g., [41], [43], [81].

VI. Discussion

A. *Trading Effectiveness for Generalization and Efficiency*

We further discuss the standalone (i.e., one-by-one) entry-level log anomaly identification task. Except for the studied entry anomalies, log anomalies also include session-based anomalies, such as time anomalies or repeat anomalies, etc. [45]. Specifically, we choose the emerging entry-level detection [48], [82], since relying on context has the following critical drawbacks:

- We prioritize generalization, which is harder to achieve with session-based detection. Recall that the organization of sessions varies across systems, and the entries will be interleaved as multiple applications logging simultaneously, session organization will be unexpectedly complicated and involve expensive engineering efforts to tune such factors system-by-system.
- Entry-level detection is more computationally efficient than modern session-based detection. The mainstream non-LLM solutions (e.g, [27], [31]) or LLM-centric solutions (e.g., [40], [42]) require processing each entry at least once as MaidLog, but incur additional consumption to mine the sequence features by recurrent neural networks or self-attention. Depending on the implementation, the computational complexity of session-based methods grows at least linearly with session length, worse than that of MaidLog.
- In applications, the majority of entries are independent. Specifically, [46] has analyzed the application logs from the IBM Cloud and points out that 94.2% of entries are entry-level independent *operational* logs, while only 0.7% of entries are *transactional* entries in logical sessions. Despite the existing study [48], Amazon CloudWatch logs (for AWS) also detect anomalies at the entry-level according to their documents and video demonstrations [82].

In summary, MaidLog primarily employs entry-level identification to achieve superior generalization and efficiency. We leave the implementation to session-level detection as an intriguing future work by, e.g., treating multiple entries as a whole *session entry* as in LogPrompt or LogGPT, since the users may still want alternative tasks.

B. *Heuristics for Calibration Early Stopping*

Currently, we employ a pseudo-label cache (Equation (3)) as a heuristic with a single hyperparameter w to improve the convergence of pseudo-labels, rather than using a deterministic early-stopping mechanism. This design choice is motivated by the fact that representative early-stopping strategies (e.g., for training loss) often require additional hyperparameters (such as patience and threshold), which we found during the development of MaidLog to be more difficult to tune than w. Given the acceptable performance of the cache-based approach, we leave the implementation of a rigorous early-stopping mechanism as future work.

In practice, users may run the calibration for a sufficiently large number of rounds (e.g., 60) or validate using their own data. However, in rare cases where over-calibration occurs (e.g., Llama with MaidLog-LogGPT in Figure 4), a heuristic to detect such behavior and trigger early stopping could be beneficial. We plan to investigate whether signals from noise-aware learning, cross-domain matching [83], or improved confidence estimation methods for LLMs can serve as effective criteria for early termination.

VII. Conclusion

In this study, we present MaidLog, a novel large language model (LLM)-assisted framework for efficient and effective log anomaly identification. Motivated by the potential of LLMs to serve as cost-effective label generators, we have developed a pseudo-label assignment process, rendering MaidLog a label-free architecture. Furthermore, we introduce a generalizable anomaly detector tailored to the unique characteristics of log data, thereby endowing MaidLog with zero-shot transfer capabilities. The performance of MaidLog has been validated through extensive experiments on real-world datasets, confirming our initial motivations and setting a foundation for the future integration of LLMs into various functionalities within log data management.

VIII. Acknowledgment

Lei Chen's work is partially supported by National Key Research and Development Program of China Grant No. 2023YFF0725100, National Science Foundation of China (NSFC) under Grant No. U22B2060, Guangdong-Hong Kong Technology Innovation Joint Funding Scheme Project No. 2024A0505040012, the Hong Kong RGC GRF Project 16213620, RIF Project R6020-19, AOE Project AoE/E-603/18, Theme-based project TRS T41-603/20R, CRF Project C2004-21G, Key Areas Special Project of Guangdong Provincial Universities 2024ZDZX1006, Guangdong Province Science and Technology Plan Project 2023A0505030011, Guangzhou municipality big data intelligence key lab, 2023A03J0012, Hong Kong ITC ITF grants MHX/078/21 and PRP/004/22FX, Zhujiang scholar program 2021JC02X170, Microsoft Research Asia Collaborative Research Grant, HKUST-Webank joint research lab and 2023 HKUST Shenzhen-Hong Kong Collaborative Innovation Institute Green Sustainability Special Fund, from Shui On Xintiandi and the InnoSpace GBA. Jiachuan Wang's work is supported in part by JST CREST (JPMJCR22M2). Libin Zheng is sponsored by the National Natural Science Foundation of China, No. 62472455 and No. U22B2060.

AI-Generated Content Acknowledgment

Generative large language models were ethically utilized in this work for manuscript refinement and code assistance purposes. Specifically, DeepSeek-V3 was used to enhance the clarity and readability of the manuscript, and GPT-4 Copilot was employed to aid in implementing and debugging code. The authors carefully reviewed and, where necessary, edited all AI-assisted content to ensure reliability, accuracy, and consistency with the author's intention, underlying data, and methods. The authors take full responsibility for the integrity and originality of all scientific content.

References

[1] S. Lee and B. Moon, "Transactional in-page logging for multiversion read consistency and recovery," in *ICDE*. IEEE Computer Society, 2011, pp. 876–887.

[2] R. Fang, H. Hsiao, B. He, C. Mohan, and Y. Wang, "High performance database logging using storage class memory," in *ICDE*. IEEE Computer Society, 2011, pp. 1221–1231.

[3] S. Oh, W. Kim, J. Seo, H. Song, S. H. Noh, and B. Nam, "Doubleheader logging: Eliminating journal write overhead for mobile DBMS," in *ICDE*. IEEE, 2020, pp. 1237–1248.

[4] Y. Gao, S. Huang, and A. G. Parameswaran, "Navigating the data lake with DATAMARAN: automatically extracting structure from log datasets," in *SIGMOD Conference*. ACM, 2018, pp. 943–958.

[5] X. Chen, J. Chen, J. Shi, P. Wang, and W. Wang, "EPAS: Efficient online log parsing via asynchronous scheduling of llm queries," in *ICDE*. IEEE, 2025, pp. 4025–4037.

[6] G. Chu, J. Wang, Q. Qi, H. Sun, S. Tao, and J. Liao, "Prefix-graph: A versatile log parsing approach merging prefix tree with probabilistic graph," in *ICDE*. IEEE, 2021, pp. 2411–2422.

[7] M. Du and F. Li, "Spell: Online streaming parsing of large unstructured system logs," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 11, pp. 2213–2227, 2019.

[8] A. Agrawal, R. Karlupia, and R. Gupta, "Logan: A distributed online log parser," in *ICDE*. IEEE, 2019, pp. 1946–1951.

[9] J. Qi, Z. Luan, S. Huang, C. J. Fung, H. Yang, and D. Qian, "Spikelog: Log-based anomaly detection via potential-assisted spiking neuron network," *IEEE Trans. Knowl. Data Eng.*, vol. 36, no. 12, pp. 9322–9335, 2024.

[10] A. Vervaet, "Monilog: An automated log-based anomaly detection system for cloud computing infrastructures," in *ICDE*. IEEE, 2021, pp. 2739–2743.

[11] Y. Tang, Z. Zhang, K. Zhao, L. Fang, Z. Li, and W. Chen, "Substructure-aware log anomaly detection," *Proc. VLDB Endow.*, vol. 18, no. 2, pp. 213–225, 2024.

[12] X. Song, Y. Zhu, J. Wu, B. Liu, and H. Wei, "Adops: An anomaly detection pipeline in structured logs," *Proc. VLDB Endow.*, vol. 16, no. 12, pp. 4050–4053, 2023.

[13] W. Cao, X. Feng, B. Liang, T. Zhang, Y. Gao, Y. Zhang, and F. Li, "Logstore: A cloud-native and multi-tenant log database," in *SIGMOD Conference*. ACM, 2021, pp. 2464–2476.

[14] Y. Zhang, G. Cong, J. Qu, R. Xu, Y. Fu, W. Li, F. Hu, J. Liu, W. Zhang, and K. Zheng, "ESTELLE: an efficient and cost-effective cloud log engine," in *SIGMOD Conference Companion*. ACM, 2024, pp. 201–213.

[15] Y. Fan, Y. Chen, K. Tung, K. Wu, and A. L. P. Chen, "A framework for enabling user preference profiling through wi-fi logs," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 3, pp. 592–603, 2016.

[16] C. Zhang, S. Zhang, C. Lei, and P. Lin, "Burstiness in query log: Web search analysis by combining global and local evidences," in *ICDE*. IEEE Computer Society, 2018, pp. 1388–1391.

[17] M. Markakis, B. Youngmann, T. Gao, Z. Zhang, R. Shahout, P. B. Chen, C. Liu, I. Sabek, and M. J. Cafarella, "From logs to causal inference: Diagnosing large systems," *Proc. VLDB Endow.*, vol. 18, no. 2, pp. 158–172, 2024.

[18] K. Vaidya, A. Dutt, V. R. Narasayya, and S. Chaudhuri, "Leveraging query logs and machine learning for parametric query optimization," *Proc. VLDB Endow.*, vol. 15, no. 3, pp. 401–413, 2021.

[19] C. Baik, H. V. Jagadish, and Y. Li, "Bridging the semantic gap with SQL query logs in natural language interfaces to databases," in *ICDE*. IEEE, 2019, pp. 374–385.

[20] T. Talius, R. Dhamankar, A. Dumitrache, and H. Kodavalla, "Transaction log based application error recovery and point in-time query," *Proc. VLDB Endow.*, vol. 5, no. 12, pp. 1781–1789, 2012.

[21] S. Lu, X. Wei, Y. Li, and L. Wang, "Detecting anomaly in big data system logs using convolutional neural network," in *DASC/PiCom/DataCom/CyberSciTech*. IEEE Computer Society, 2018, pp. 151–158.

[22] J. Breier and J. Branišová, "Anomaly detection from log files using data mining techniques," in *Information Science and Applications*. Springer, 2015, pp. 449–457.

[23] Y. Zhang and A. Sivasubramaniam, "Failure prediction in IBM bluegene/l event logs," in *IPDPS*. IEEE, 2008, pp. 1–5.

[24] V. Le and H. Zhang, "Log-based anomaly detection with deep learning: How far are we?" in *ICSE*. ACM, 2022, pp. 1356–1367.

[25] J. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection," in *USENIX ATC*. USENIX Association, 2010.

[26] W. Xu, L. Huang, A. Fox, D. A. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *ICML*. Omnipress, 2010, pp. 37–46.

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

[28] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, J. Chen, X. He, R. Yao, J. Lou, M. Chintalapati, F. Shen, and D. Zhang, "Robust log-based anomaly detection on unstable log data," in *ESEC/SIGSOFT FSE*. ACM, 2019, pp. 807–817.

[29] V. Le and H. Zhang, "Log-based anomaly detection without log parsing," in *ASE*. IEEE, 2021, pp. 492–504.

[30] ——, "Prelog: A pre-trained model for log analytics," *Proc. ACM Manag. Data*, vol. 2, no. 3, p. 163, 2024.

[31] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *CCS*. ACM, 2017, pp. 1285–1298.

[32] Q. Lin, H. Zhang, J. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *ICSE (Companion Volume)*. ACM, 2016, pp. 102–111.

[33] L. Yang, J. Chen, Z. Wang, W. Wang, J. Jiang, X. Dong, and W. Zhang, "Semi-supervised log-based anomaly detection via probabilistic label estimation," in *ICSE*. IEEE, 2021, pp. 1448–1460.

[34] 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 *ISSRE*. IEEE, 2020, pp. 37–47.

[35] P. Jia, S. Cai, B. C. Ooi, P. Wang, and Y. Xiong, "Robust and transferable log-based anomaly detection," *Proc. ACM Manag. Data*, vol. 1, no. 1, pp. 64:1–64:26, 2023.

[36] C. Zhang, T. Jia, G. Shen, P. Zhu, and Y. Li, "Metalog: Generalizable cross-system anomaly detection from logs with meta-learning," in *ICSE*. ACM, 2024, pp. 154:1–154:12.

[37] D. Roy, X. Zhang, R. Bhave, C. Bansal, P. H. B. Las-Casas, R. Fonseca, and S. Rajmohan, "Exploring llm-based agents for root cause analysis," in *SIGSOFT FSE Companion*. ACM, 2024, pp. 208–219.

[38] Z. Ma, A. R. Chen, D. J. Kim, T. Chen, and S. Wang, "Llmparser: An exploratory study on using large language models for log parsing," in *ICSE*. ACM, 2024, pp. 99:1–99:13.

[39] A. Zhong, D. Mo, G. Liu, J. Liu, Q. Lu, Q. Zhou, J. Wu, Q. Li, and Q. Wen, "Logparser-llm: Advancing efficient log parsing with large language models," in *KDD*. ACM, 2024, pp. 4559–4570.

[40] F. Hadadi, Q. Xu, D. Bianculli, and L. C. Briand, "Anomaly detection on unstable logs with GPT models," *CoRR*, vol. abs/2406.07467, 2024.

[41] Y. Liu, S. Tao, W. Meng, J. Wang, W. Ma, Y. Chen, Y. Zhao, H. Yang, and Y. Jiang, "Interpretable online log analysis using large language models with prompt strategies," in *ICPC*. ACM, 2024, pp. 35–46.

[42] H. Guo, S. Yuan, and X. Wu, "Logbert: Log anomaly detection via BERT," in *IJCNN*. IEEE, 2021, pp. 1–8.

[43] T. Cui, S. Ma, Z. Chen, T. Xiao, S. Tao, Y. Liu, S. Zhang, D. Lin, C. Liu, Y. Cai, W. Meng, Y. Sun, and D. Pei, "Logeval: A comprehensive benchmark suite for large language models in log analysis," *CoRR*, vol. abs/2407.01896, 2024.

[44] J. Pan, S. L. Wong, and Y. Yuan, "Raglog: Log anomaly detection using retrieval augmented generation," *CoRR*, vol. abs/2311.05261, 2023.

[45] M. Landauer, F. Skopik, and M. Wurzenberger, "A critical review of common log data sets used for evaluation of sequence-based anomaly detection techniques," *Proc. ACM Softw. Eng.*, vol. 1, no. FSE, pp. 1354–1375, 2024.

[46] T. Jia, L. Yang, P. Chen, Y. Li, F. Meng, and J. Xu, "Logsed: Anomaly diagnosis through mining time-weighted control flow graph in logs," in *CLOUD*. IEEE Computer Society, 2017, pp. 447–455.

[47] Y. Huo, C. Lee, Y. Su, S. Shan, J. Liu, and M. R. Lyu, "Evlog: Identifying anomalous logs over software evolution," in *ISSRE*. IEEE, 2023, pp. 391–402.

[48] W. Zhang, Q. Zhang, E. Yu, Y. Ren, Y. Meng, M. Qiu, and J. Wang, "Lograg: Semi-supervised log-based anomaly detection with retrieval-augmented generation," in *ICWS*. IEEE, 2024, pp. 1100–1102.

[49] L. Ma, L. Cao, P. M. VanNostrand, D. M. Hofmann, Y. Su, and E. A. Rundensteiner, "Pluto: Sample selection for robust anomaly detection on polluted log data," *Proc. ACM Manag. Data*, vol. 2, no. 4, pp. 203:1–203:25, 2024.

[50] L. Ruff, N. Görnitz, L. Deecke, S. A. Siddiqui, R. A. Vandermeulen, A. Binder, E. Müller, and M. Kloft, "Deep one-class classification," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 80. PMLR, 2018, pp. 4390–4399.

[51] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, "Fasttext.zip: Compressing text classification models," *CoRR*, vol. abs/1612.03651, 2016.

[52] M. Li and C. Zhu, "Noisy label processing for classification: A survey," *CoRR*, vol. abs/2404.04159, 2024.

[53] E. Arazo, D. Ortego, P. Albert, N. E. O'Connor, and K. McGuinness, "Unsupervised label noise modeling and loss correction," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 97. PMLR, 2019, pp. 312–321.

[54] K. Zhou, Z. Liu, Y. Qiao, T. Xiang, and C. C. Loy, "Domain generalization: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 4, pp. 4396–4415, 2023.

[55] Y. Li, X. Tian, M. Gong, Y. Liu, T. Liu, K. Zhang, and D. Tao, "Deep domain generalization via conditional invariant adversarial networks," in *ECCV (15)*, ser. Lecture Notes in Computer Science, vol. 11219. Springer, 2018, pp. 647–663.

[56] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 2, pp. 318–327, 2020.

[57] Y. Ganin and V. S. Lempitsky, "Unsupervised domain adaptation by backpropagation," in *ICML*, ser. JMLR Workshop and Conference Proceedings, vol. 37. JMLR.org, 2015, pp. 1180–1189.

[58] J. Zhu, S. He, P. He, J. Liu, and M. R. Lyu, "Loghub: A large collection of system log datasets for ai-driven log analytics," in *ISSRE*. IEEE, 2023, pp. 355–366.

[59] A. Makanju, N. Zincir-Heywood, and E. E. Milios, "Clustering event logs using iterative partitioning," in *KDD*. ACM, 2009, pp. 1255–1264.

[60] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "An evaluation study on log parsing and its use in log mining," in *DSN*. IEEE Computer Society, 2016, pp. 654–661.

[61] T. Jia, Y. Li, Y. Yang, G. Huang, and Z. Wu, "Augmenting log-based anomaly detection models to reduce false anomalies with human feedback," in *KDD*. ACM, 2022, pp. 3081–3089.

[62] J. Liu, J. Huang, Y. Huo, Z. Jiang, J. Gu, Z. Chen, C. Feng, M. Yan, and M. R. Lyu, "Log-based Anomaly Detection based on EVT Theory with feedback," Sep. 2023.

[63] J. A. Hanley and B. J. McNeil, "The meaning and use of the area under a receiver operating characteristic (roc) curve." *Radiology*, vol. 143, no. 1, pp. 29–36, 1982.

[64] F. Research, "Word vectors for 157 languages · fastText — fasttext.cc," https://fasttext.cc/docs/en/crawl-vectors.html, [Accessed 02-09-2025].

[65] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *ICLR (Poster)*. OpenReview.net, 2019.

[66] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *NeurIPS*, 2019, pp. 8024–8035.

[67] P. Izmailov, D. Podoprikhin, T. Garipov, D. P. Vetrov, and A. G. Wilson, "Averaging weights leads to wider optima and better generalization," in *UAI*. AUAI Press, 2018, pp. 876–885.

[68] Huggingface, "GitHub - huggingface/text-generation-inference: Large Language Model Text Generation Inference — github.com," https://github.com/huggingface/text-generation-inference, [Accessed 02-09-2025].

[69] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de Las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, "Mistral 7b," *CoRR*, vol. abs/2310.06825, 2023.

[70] A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, H. Lin, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Lin, K. Dang, K. Lu, K. Bao, K. Yang, L. Yu, M. Li, M. Xue, P. Zhang, Q. Zhu, R. Men, R. Lin, T. Li, T. Xia, X. Ren, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Wan, Y. Liu, Z. Cui, Z. Zhang, and Z. Qiu, "Qwen2.5 technical report," *CoRR*, vol. abs/2412.15115, 2024.

[71] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, A. Goyal, A. Hartshorn, A. Yang, A. Mitra, A. Sravankumar, A. Korenev, A. Hinsvark, A. Rao, A. Zhang, A. Rodriguez, A. Gregerson, A. Spataru, B. Rozière, B. Biron, B. Tang, B. Chern, C. Caucheteux, C. Nayak, C. Bi, C. Marra, C. McConnell, C. Keller, C. Touret, C. Wu, C. Wong, C. C. Ferrer, C. Nikolaidis, D. Allonsius, D. Song, D. Pintz, D. Livshits, D. Esiobu, D. Choudhary, D. Mahajan, D. Garcia-Olano, D. Perino, D. Hupkes, E. Lakomkin, E. AlBadawy, E. Lobanova, E. Dinan, E. M. Smith, F. Radenovic, F. Zhang, G. Synnaeve, G. Lee, G. L. Anderson, G. Nail, G. Mialon, G. Pang, G. Cucurell, H. Nguyen, H. Korevaar, H. Xu, H. Touvron, I. Zarov, I. A. Ibarra, I. M. Kloumann, I. Misra, I. Evtimov, J. Copet, J. Lee, J. Geffert, J. Vranes, J. Park, J. Mahadeokar, J. Shah, J. van der Linde, J. Billock, J. Hong, J. Lee, J. Fu, J. Chi, J. Huang, J. Liu, J. Wang, J. Yu, J. Bitton, J. Spisak, J. Park, J. Rocca, J. Johnstun, J. Saxe, J. Jia, K. V. Alwala, K. Upasani, K. Plawiak, K. Li, K. Heafield, K. Stone, and et al., "The llama 3 herd of models," *CoRR*, vol. abs/2407.21783, 2024.

[72] O. R. developers, "Onnx runtime," https://onnxruntime.ai/, 2021, version: 1.22.0.

[73] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *NAACL-HLT (1)*. Association for Computational Linguistics, 2019, pp. 4171–4186.

[74] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized BERT pretraining approach," *CoRR*, vol. abs/1907.11692, 2019.

[75] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," in *ACL*. Association for Computational Linguistics, 2020, pp. 7871–7880.

[76] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *ICWS*. IEEE, 2017, pp. 33–40.

[77] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *ICSE (SEIP)*. IEEE / ACM, 2019, pp. 121–130.

[78] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[79] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS*, 2017, pp. 5998–6008.

[80] Y. Liu, X. Zhang, S. He, H. Zhang, L. Li, Y. Kang, Y. Xu, M. Ma, Q. Lin, Y. Dang, S. Rajmohan, and D. Zhang, "Uniparser: A unified log parser for heterogeneous log data," in *WWW*. ACM, 2022, pp. 1893–1901.

[81] C. Ouyang, L. Yue, S. Di, L. Zheng, L. Yue, S. Pan, J. Yin, and M. Zhang, "Code2mcp: Transforming code repositories into MCP services," *CoRR*, vol. abs/2509.05941, 2025.

[82] A. Cloud, "Log anomaly detection - Amazon CloudWatch Logs — docs.aws.amazon.com," https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/LogsAnomalyDetection.html, [Accessed 02-09-2025].

[83] Z. Wang, Z. Gao, Y. Yang, G. Wang, C. Jiao, and H. T. Shen, "Geometric matching for cross-modal retrieval," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 36, no. 3, pp. 5509–5521, 2025.