MetaLog: Generalizable Cross-System Anomaly Detection from Logs with Meta-Learning

Chenyangguang Zhang[†]
Tsinghua University
Beijing, China
zcyg22@mails.tsinghua.edu.cn

Tong Jia[*] [†]
Institute for Artificial Intelligence
Peking University
Beijing, China
jia.tong@pku.edu.cn

Guopeng Shen
Linkedsee Technology (China) Limited
Beijing, China
shenguopeng@linkedsee.com

Pinyan Zhu
Linkedsee Technology (China) Limited
Beijing, China
zhupinyan@linkedsee.com

Ying Li[*]
National Engineering Research Center for Software Engineering
Peking University
Beijing, China
li.ying@pku.edu.cn

ABSTRACT

Log-based anomaly detection plays a crucial role in ensuring the stability of software. However, current approaches for log-based anomaly detection heavily depend on a vast amount of labeled historical data, which is often unavailable in many real-world systems. To mitigate this problem, we leverage the features of the abundant historical labeled logs of mature systems to help construct anomaly detection models of new systems with very few labels, that is, to generalize the model ability trained from labeled logs of mature systems to achieve anomaly detection on new systems with insufficient data labels. Specifically, we propose MetaLog, a generalizable cross-system anomaly detection approach. MetaLog first incorporates a globally consistent semantic embedding module to obtain log event semantic embedding vectors in a shared global space. Then it leverages the meta-learning paradigm to improve the model's generalization ability. We evaluate MetaLog's performance on four public log datasets (HDFS, BGL, OpenStack, and Thunderbird) from four different systems. Results show that MetaLog reaches over 80% F1-score when using only 1% labeled logs of the target system, showing similar performance with state-of-the-art supervised anomaly detection models trained with 100% labeled data. Besides, it outperforms state-of-art transfer-learning-based cross-system anomaly detection models by 20% in the same settings of 1% labeled training logs of the target system.

CCS CONCEPTS

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

[*]Corresponding Authors
[†]Equally Contribution

KEYWORDS

Meta-learning, Anomaly detection, System logs

ACM Reference Format:
Chenyangguang Zhang, Tong Jia, Guopeng Shen, Pinyan Zhu, and Ying Li. 2024. MetaLog: Generalizable Cross-System Anomaly Detection from Logs with Meta-Learning. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3597503.3639205

1 INTRODUCTION

Software systems are becoming increasingly large and complex and are subject to more failures. As system logs record system states and significant events of running processes, they are an excellent source of information for anomaly detection. Log-based anomaly detection is promising for system reliability and has been widely studied.

Existing log-based anomaly detection models can be broadly divided into two categories: unsupervised models and supervised models. Unsupervised models [6, 24, 28, 44] utilize sequential neural networks such as LSTM, GRU, etc. to learn the occurrence possibility of log events in normal event sequences to predict subsequent log events and identify the unmatched log event as an anomaly. Their effectiveness is limited due to the lack of supervision of anomalous logs [43]. Supervised models [35, 41, 47] build classification models to identify anomalous logs and are more effective compared to unsupervised models. However, their effectiveness heavily relies on large amounts of labeled logs.

In real-world software systems, identifying anomalous logs is a significant challenge due to the vast volume of system logs in which such anomalies are deeply buried. Consequently, obtaining accurate data labels, especially for anomalous logs, is a rare and difficult task [21]. To address this issue, existing approaches have attempted two strategies. One approach involves utilizing clustering methods to deduce fuzzy data labels [43]. However, this method still necessitates a considerable number of labeled anomalous logs, making it impractical in many circumstances. As discussed in [7], experiments on HDFS dataset [42] show that this type of approach requires 11,244 labeled anomalous log vectors to achieve 96.53%

recall rate, and when the labeled anomalous log vectors are reduced to 124, the recall rate decreases to 18.43%. The second strategy involves incorporating human labeling actions online through active learning techniques [7, 21]. Nonetheless, this strategy suffers from two key limitations. First, online human labeling can be complex, demanding considerable human effort. For instance, HiLog [21] necessitates the identification of four anti-patterns, requiring humans to label the specific anti-pattern to which each false anomaly belongs. Additionally, these predefined anti-patterns might not cover all complex scenarios, leading to incomplete results. Furthermore, during human labeling, each reported anomaly must be individually assessed by humans to determine its legitimacy, which can be time-consuming and resource-intensive, especially when a system experiences a large number of true anomalies. Second, the anomaly detection models utilizing online human labeling suffer from a cold start problem. This implies that the model's ability is heavily dependent on the accumulation of online human labels, particularly those associated with anomalous logs. For instance, ACLog [7] necessitates 205 online labels of logs from the BGL dataset [31]. Accumulating a sufficient number of labels requires a significant amount of time, during which the model's performance remains limited.

To mitigate these problems, we propose leveraging the features extracted from abundant historical labeled logs of mature systems to facilitate building anomaly detection models for new systems with limited labeled data. Our approach aims to generalize the model's ability, trained on labeled logs from mature systems (source systems), to achieve effective anomaly detection in new systems (target systems) with insufficient data labels. We term this task Generalizable Cross-system Log Anomaly Detection (GCLAD). Previous related works include LogTransfer [3] and LogTAD [14]. These approaches rely on simple transfer learning models, where the source and target systems share part of the neural network architecture. However, the theoretical analysis proposed in [40] demonstrates that transfer learning may struggle to handle situations with a substantial distribution gap between the source and target domains. Moreover, several studies have illustrated that the performance of transfer-learning-based methods can only be guaranteed under mild assumptions and is dependent on the relevance between the source and target systems [1, 2, 11, 45, 50]. For instance, LogTransfer[3] is limited to scenarios where the source and target data belong to the same type of switch logs or the same software family, such as the Hadoop application and Hadoop file system. Consequently, the generalization capability of these methods is restricted, and they cannot meet the requirements of GCLAD.

In this paper, we aim to solve the GCLAD problem where the gap between logs in source and target systems is very large and the labeled logs of the target system are very few. However, solving the GCLAD problem is not easy. First, since both the syntax and the semantics of logs in different systems are very different, how can we bridge the huge gap of log contents between different systems? Second, how can we generalize the model ability trained with logs in the source system to perform well on the target system with very few labeled logs?

Facing these two challenges, we propose MetaLog, a generalizable cross-system anomaly detection approach. To tackle the first challenge, we introduce a globally consistent semantic embedding (GCSE) stage, which facilitates the extraction of log event semantic embedding vectors in a shared global space. The global space indicates that the semantic embeddings of different log events from different systems share the same vector space. In this stage, we leverage pre-trained word embeddings from Glove [33] to capture semantic information distributed in a unified global space. To generate the semantic embedding of a log event, we develop a per-event weighted aggregation technique that combines the semantic information of the words composing the event. The GCSE stage ensures global consistency of log semantic embeddings across different systems, in contrast to some embedding methods like [28] that represent log events separately for each log system. To address the second challenge, we employ meta-learning techniques to enhance the generalization capacity of MetaLog. Meta-learning involves outer optimization, allowing the MetaLog network to handle a broader range of meta-representations beyond just model parameters [10, 16]. The ultimate goal of meta-learning is to enable the MetaLog network to detect log anomalies effectively in any system, in other words, to generalize from any source logs to any target logs. Extensive evidence has shown that meta-learning requires less data to achieve comparable generalization results compared to transfer-learning [12, 13, 39], especially when dealing with limited target data accessibility.

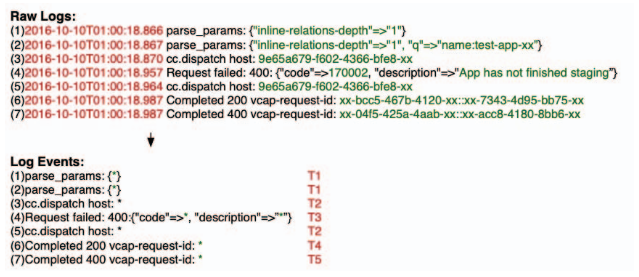
We evaluate MetaLog's performance on four public log datasets (HDFS, BGL, OpenStack, and Thunderbird) from four different systems. Results show that our approach reaches over 80% F1 score when using only 1% labeled logs of the target system, showing similar performance with state-of-the-art supervised anomaly detection models trained with 100% labeled data. Besides, it outperforms the state-of-the-art transfer-learning-based cross-system anomaly detection models by 20% in the same settings of 1% labeled training logs of the target system. In summary, the contributions of this paper are as follows:

- We propose MetaLog, a novel generalizable anomaly detection approach based on the meta-learning paradigm.
- We adopt a globally consistent semantic embedding stage to obtain log event semantic embedding vectors in a shared global space and design a meta-learning process for the GCLAD tasks.
- Evaluation results on four public log datasets show the remarkable effectiveness of our approach.

2 PRELIMINARIES

2.1 System Logs and Log Events

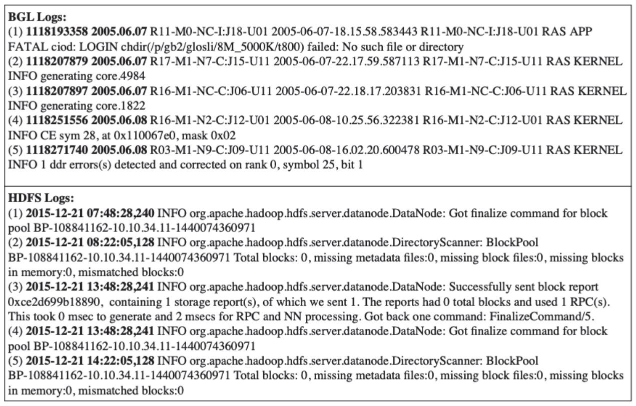
Logs are lines of text messages that contain abundant information (e.g., parameters, execution status, events, etc.) about the running status of the system. A *log sequence* consists of a sequence of log entries in order of output time. An *event* is an abstraction of a print statement in source code, which manifests itself in logs with different embedded parameter values in different executions—represented as a set of invariant keywords and parameters (denoted by parameter placeholder "*"). An event can be used to summarize multiple log entries. An *event sequence* consists of a sequence of log events in one-to-one correspondence with log entries in a log sequence. Figure 1 shows an example of logs and their corresponding log events. The workflow of existing log-based anomaly



```
Raw Logs:
(1)2016-10-10T01:00:18.866 parse_params: {"inline-relations-depth"=>"1"}
(2)2016-10-10T01:00:18.867 parse_params: {"inline-relations-depth"=>"1", "q"=>"name:test-app-xx"}
(3)2016-10-10T01:00:18.870 cc.dispatch host: 9e65a679-f602-4366-bfe8-xx
(4)2016-10-10T01:00:18.957 Request failed: 400: {"code"=>170002, "description"=>"App has not finished staging"}
(5)2016-10-10T01:00:18.964 cc.dispatch host: 9e65a679-f602-4366-bfe8-xx
(6)2016-10-10T01:00:18.987 Completed 200 vcap-request-id: xx-bcc5-467b-4120-xx::xx-7343-4d95-bb75-xx
(7)2016-10-10T01:00:18.987 Completed 400 vcap-request-id: xx-04f5-425a-4aab-xx::xx-acc8-4180-8bb6-xx

Log Events:
(1)parse_params: {*}                                 T1
(2)parse_params: {*}                                 T1
(3)cc.dispatch host: *                               T2
(4)Request failed: 400:{"code"=>*, "description"=>"*"}  T3
(5)cc.dispatch host: *                               T2
(6)Completed 200 vcap-request-id: *                  T4
(7)Completed 400 vcap-request-id: *                  T5
```

Figure 1: Logs and their corresponding log events. The red part in raw logs is timestamps. The green part is variables. The green part is replaced with placeholder “*” in log events.



```
BGL Logs:
(1) 1118193358 2005.06.07 R11-M0-NC-I:J18-U01 2005-06-07-18.15.58.583443 R11-M0-NC-I:J18-U01 RAS APP
FATAL ciod: LOGIN chdir(/p/gb2/glosli/8M_5000K/t800) failed: No such file or directory
(2) 1118207879 2005.06.07 R17-M1-N7-C:J15-U11 2005-06-07-22.17.59.587113 R17-M1-N7-C:J15-U11 RAS KERNEL
INFO generating core.4984
(3) 1118207897 2005.06.07 R16-M1-NC-C:J06-U11 2005-06-07-22.18.17.203831 R16-M1-NC-C:J06-U11 RAS KERNEL
INFO generating core.1822
(4) 1118251556 2005.06.08 R16-M1-N2-C:J12-U01 2005-06-08-10.25.56.322381 R16-M1-N2-C:J12-U01 RAS KERNEL
INFO CE sym 28, at 0x110067e0, mask 0x02
(5) 1118271740 2005.06.08 R03-M1-N9-C:J09-U11 2005-06-08-16.02.20.600478 R03-M1-N9-C:J09-U11 RAS KERNEL
INFO 1 ddr errors(s) detected and corrected on rank 0, symbol 25, bit 1

HDFS Logs:
(1) 2015-12-21 07:48:28,240 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Got finalize command for block
pool BP-108841162-10.10.34.11-1440074360971
(2) 2015-12-21 08:22:05,128 INFO org.apache.hadoop.hdfs.server.datanode.DirectoryScanner: BlockPool
BP-108841162-10.10.34.11-1440074360971 Total blocks: 0, missing metadata files:0, missing block files:0, missing blocks
in memory:0, mismatched blocks:0
(3) 2015-12-21 13:48:28,241 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Successfully sent block report
0xce2d699b18890,  containing 1 storage report(s), of which we sent 1. The reports had 0 total blocks and used 1 RPC(s).
This took 0 msec to generate and 2 msecs for RPC and NN processing. Got back one command: FinalizeCommand/5.
(4) 2015-12-21 13:48:28,241 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Got finalize command for block
pool BP-108841162-10.10.34.11-1440074360971
(5) 2015-12-21 14:22:05,128 INFO org.apache.hadoop.hdfs.server.datanode.DirectoryScanner: BlockPool
BP-108841162-10.10.34.11-1440074360971 Total blocks: 0, missing metadata files:0, missing block files:0, missing blocks
in memory:0, mismatched blocks:0
```

Figure 2: BGL and HDFS Logs.

detection approaches usually consists of two steps: log parsing and model training. Log parsing first mines log events from system logs and transforms the log sequence into an event sequence. Model training aims to capture the features of event sequences and build deep neural network-based anomaly detection models.

2.2 Generalizable Cross-System Log Anomaly Detection

Our objective is to tackle the GCLAD task, which entails utilizing the extensive historical labeled logs from a mature system (source system) to assist in constructing anomaly detection models for a new system (target system) that has limited labeled data. We aim to extend the model’s capability, acquired from labeled logs of the established system, to achieve successful anomaly detection in new systems that lack sufficient data labels. We refer to this task as Generalizable Cross-system Log Anomaly Detection (GCLAD).

The GCLAD possesses two key characteristics. Firstly, the logs from the source system and the target system are completely different. As depicted in Figure 2, for instance, BGL logs and HDFS logs exhibit distinct data structures and semantics. BGL logs primarily contain information related to the RAS kernel, while HDFS logs describe operations on the storage block pool. The considerable data gap between different systems presents a significant challenge. Second, data labels, especially anomalous log labels, in the target system are scarce and insufficient. In real-world software systems, identifying anomalous logs proves to be a daunting task due to the large volume of system logs, resulting in a scarcity of accurate data labels, particularly for anomalies [21].

In this paper, we endeavor to address GCLAD by introducing two scenarios: bilateral generalization and zero-shot generalization. In the bilateral generalization scenario, the model is trained using labeled source system logs and a limited number of labeled target system logs, and it subsequently performs anomaly detection on the target system logs. On the other hand, zero-shot generalization represents a more advanced goal where the model is trained solely on multiple labeled source system logs and directly deployed for anomaly detection on a third target system logs. Our primary focus lies in the bilateral generalization scenario, while we also explore a potential technical direction for the zero-shot generalization scenario.

3 METHOD

3.1 Overview

In this paper, we we present MetaLog, a novel approach that utilizes meta-learning to handle the challenging Generalizable Cross-system Log Anomaly Detection (GCLAD) tasks described in Section 2.2. These GCLAD tasks are commonly encountered in real industrial scenarios. However, existing methods often struggle to generalize effectively across different systems with significant domain gaps. To address this issue, we introduce the MetaLog system, comprising three modules: Log Parsing, Globally Consistent Semantic Embedding (GCSE), and an efficient MetaLog network utilizing an elaborate meta-learning technique. By leveraging the advantages provided by GCSE and the meta-learning technique, the proposed MetaLog system exhibits strong generalization ability among different log systems, enabling it to tackle complex GCLAD tasks, including bilateral generalization and zero-shot generalization.

The MetaLog approach starts by processing unstructured raw logs from various industrial systems using the classical log parsing method Drain [15], to obtain the log events and processed log event sequences. Then, we follow previous works [3, 14, 43, 47] to extract semantic embeddings for each log event, as they have been demonstrated to offer more informative representations compared to traditional indices-based methods. Given the cross-system nature of log anomaly detection, we employ the globally consistent semantic embedding (GCSE) module (Section 3.2) to obtain log event semantic embedding vectors in a shared global space. These globally consistent log event embeddings are then fed into the MetaLog network (Section 3.4), which detect anomalies based on sequences of log event embeddings (Figure 4).

The meta-learning-based training process of the MetaLog network plays a pivotal role in solving GCLAD tasks, as illustrated in Section 3.3. Meta-tasks are meticulously designed with the aim of achieving generalizable cross-system log anomaly detection. Each meta-task comprises a meta-training stage using source logs and a meta-testing stage using target logs. The meta-learner aggregates gradients from these meta-tasks to guide the network in acquiring the necessary generalization ability between source and target system logs.

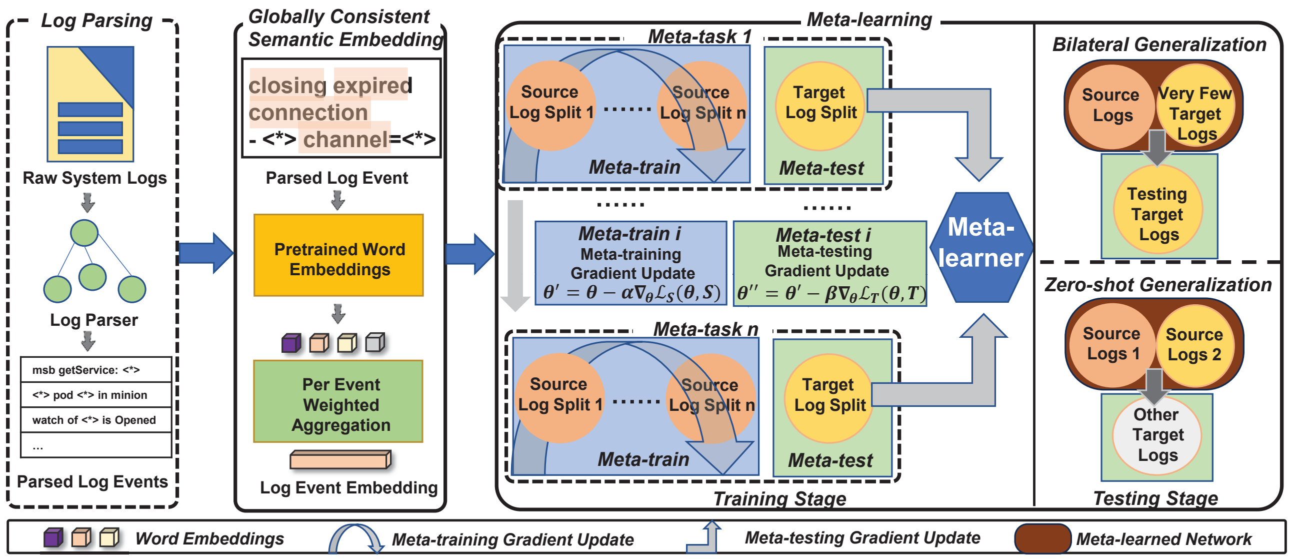


Figure 3: The proposed generalizable cross-system log anomaly detection pipeline for MetaLog. MetaLog pipeline consists of three stages: Log Parsing, Globally Consistent Semantic Embedding, and Meta-learning. In the log parsing stage, raw logs are parsed to log events. In the globally consistent semantic embedding stage, the parsed log events from different systems are fed into a pre-trained word embedding model to obtain globally consistent word embeddings in a shared vector space. The globally consistent log event embeddings are generated by a weighted aggregation module with the input of corresponding word embeddings. Finally, in the meta-learning stage, the MetaLog network utilizes the sequences of log event embeddings to predict whether the log sequences are normal or anomalous. The network is trained using meta-learning techniques, which enable it to generalize well across different systems. It learns from a series of cross-system log anomaly detection meta-tasks, consisting of a meta-training stage with source logs and a meta-testing stage with target logs. This training strategy equips the MetaLog network with the ability to handle both bilateral and zero-shot generalization tasks when tested on complex real-world scenarios.

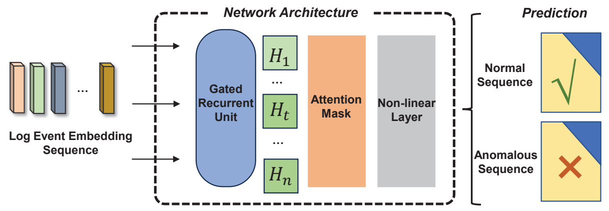


Figure 4: The MetaLog network architecture. Given a sequence of log event embeddings, the network outputs the anomaly prediction. The architecture contains three cascaded modules including a gated recurrent unit, an attention mask, and a non-linear layer.

The whole pipeline of the proposed MetaLog system is exhibited in Figure 3.

3.2 Globally Consistent Semantic Embedding

To effectively address the generalizable cross-system log anomaly detection task, it is crucial to properly represent the original unorganized input logs in a shared representation space. To achieve this, we introduce the globally consistent semantic embedding (GCSE) stage within the MetaLog approach.

After log parsing, the GCSE stage receives each parsed log event and generates a globally consistent log event embedding. Given that log events are primarily created by system developers to capture the operational state of a system, it is common for them to include English words that carry their own meaning. Exploiting this feature, we treat each log event as a sentence in natural language. To begin with, we preprocess the log events by eliminating non-character tokens and breaking down composite tokens into separate words using the Camel Case method [5]. Subsequently, we employ pre-trained word embeddings from Glove [33], which guarantees that word embeddings are distributed in a unified global space. After converting each word into a 300-dimensional embedding vector, we propose a per-event weighted aggregation method to generate the semantic embedding of the log event by its constituent words. To determine the weight, we employ the commonly used term frequency and inverse document frequency from information retrieval [37]. These weights have been proven effective in log semantic embeddings [43]. For a given log event consisting of multiple words, we calculate the term frequency of an individual word, denoted as w, using the formula $\frac{n_w}{n_e}$, where n_w represents the number of times word w appears in the log event, while n_e corresponds to the total number of words in the log event. The inverse document frequency is defined by $log(\frac{n_l}{n_l^w})$, where n_l is the total number of log events and n_l^w is the number of log events containing w. Our per-event weighted aggregation method generates the final log event embedding by

$$V_e = \frac{1}{n_e}\sum_{i=1}^{n_e}\frac{n_{w_i}}{n_e}log(\frac{n_l}{n_l^{w_i}})V_{w_i}, \tag{1}$$

where V_{w_i} represents the global pretrained word embeddings of the i-th word in the log event, and V_e refers to the globally consistent log event semantic embedding. Such aggregation method utilizes both term frequency and inverse document frequency to extract local information within a log event and global information of the whole log dataset.

It is crucial to emphasize that the GCSE module's pipeline guarantees the maintenance of overall consistency in log semantic embeddings across various systems. Unlike approaches that individually represent log events for each log system, our method employs universally shared word embeddings and the per-event weighted aggregation technique. As a result, log event embeddings are distributed within the same global embedding space, irrespective of their originating log systems. This consistency greatly assists in effectively addressing the challenge of cross-system log anomaly detection.

3.3 Meta-Learning for Cross-System Logs

Preliminary. The meta-learning process plays a pivotal role in MetaLog to gain strong generalization ability in handling GCLAD tasks. For a clear illustration, we first introduce some terminologies of meta-learning. In the training stage of the meta-learning process, log data from at least two systems are used, named *source logs* and *target logs* respectively. A general goal of meta-learning is to teach our MetaLog network θ to detect anomalies under any systems, in other words, to generalize from source logs to target logs. During the training stage, the minimum unit for the *meta-learner* to update the gradient is *meta-task*. A meta-task consists of a *meta-train* stage using several *source log splits* and a *meta-test* stage using *target log splits*. The meta-train and meta-test stages are responsible for different adaptive gradient updates. From the perspective of data, source log splits in the meta-train stage are randomly sampled in the source logs, and target log splits in the meta-test stage are randomly sampled in the target logs.

Meta-tasks. Towards the aim of GCLAD tasks, designing proper meta-tasks across different source and target log systems is important for MetaLog. Every meta-task needs to encourage the network to learn how to generate from source logs to target logs. A meta-task is composed of two stages: meta-train and meta-test. During the meta-train stage, the MetaLog network calculates the gradients using multiple source log splits, denoted as $\{S_i, i = 1, ..., n_s\}$. These splits are randomly selected from the set of source logs, denoted as $\{S\}$. Similarly, in the meta-test stage, the parameters of the MetaLog network are updated using the gradients obtained from target log splits, denoted as $\{T_i, i = 1, ..., n_t\}$. The target log splits are randomly sampled from the set of target logs, denoted as $\{T\}$. For every constructed meta-task, the random sampling for source and target log splits the random sampling for source and target log splits is done with replacement.

Meta-learning Algorithm. The aim of the proposed meta-learning algorithm for cross-system logs is to teach the MetaLog network by elaborated tasks generalizing from logs of source systems to target systems. The gradient delivery process in meta-training aggregates the domain adaptation information contained by meta-tasks, since the network is optimized not only by the source splits but also by the target splits. Thus, the MetaLog network trained with a meta-learning algorithm is endowed with much stronger cross-system generalization ability, and gains superior performance handling GCLAD tasks. The overall procedure of our proposed meta-learning algorithm is exhibited in Algorithm 1.

To construct the meta-tasks for the meta-learner, we begin by randomly sampling n_s source log splits, denoted as $\{S_i, i = 1, ..., n_s\}$, from the set of source logs $\{S\}$. Similarly, we randomly sample n_t target log splits, denoted as $\{T_i, i = 1, ..., n_t\}$, from the set of target logs $\{T\}$. Then, the cross-system-aware gradient is calculated in the meta-training and meta-testing stages sequentially. In the meta-training stage, the MetaLog network parameters are updated by the source gradient:

$$\theta' = \theta - \alpha\nabla_\theta\mathcal{L}_S(\theta, S_i), \tag{2}$$

where S_i denotes the i-th source log split. After n_s iterations, the network parameters are updated to θ'_S. Moving on to the meta-testing stage, the corresponding loss term $\mathcal{L}_T$ is calculated by the meta-trained parameters θ'_S. The aggregated meta-testing loss term is then represented as $\mathcal{L}_T(\theta'_S, \{T\})$ in sampled target logs.

After obtaining the meta-testing loss term, the meta-learning process optimizes the MetaLog network using the following equation:

$$\theta = \theta - \gamma\frac{\partial(\mathcal{L}_S(\theta, \{S\}) + \beta\mathcal{L}_T(\theta - \alpha\nabla_\theta\mathcal{L}_S(\theta, \{S\}), \{T\}))}{\partial\theta}, \tag{3}$$

where the term $\theta - \alpha\nabla_\theta\mathcal{L}_S(\theta, \{S\})$ refers to the meta-training gradient update, and α, β, γ are meta-learning rate hyperparameters. This comprehensive optimization strategy considers information from both source and target log systems, giving the MetaLog network enough cues to solve the GCLAD tasks.

Discussions. As discussed in Section 2.2, the GCLAD tasks present greater challenges for neural networks in terms of domain gaps and generalization difficulties compared to transfer-learning-based cross-system log anomaly detection tasks defined by [3, 14]. In this section, we aim to explain why the meta-learning-based approach, MetaLog, is better suited for solving GCLAD tasks, while previous transfer-learning-based approaches have shown inferior performance. As explained in Section 2.2, GCLAD tasks typically involve a significant domain gap in the log system, which is not as pronounced in the scenarios of previous transfer-learning-based cross-system log anomaly detection methods [3, 14]. Therefore, based on the theoretical analysis proposed in [40], which demonstrates that transfer-learning struggles with handling dynamic target-source relationships and situations with large distribution gaps, it is apparent that transfer-learning-based methods may not be suitable for the challenging GCLAD tasks. Furthermore, several studies have highlighted that the performance of transfer-learning-based methods can only be guaranteed under mild assumptions and is dependent on the relevance of the source and target data [1, 2, 11, 45, 50]. In

Algorithm 1 Meta-Learning for Cross-System Logs

```
Input: source logs $S$, target logs $T$
Init: MetaLog Network $\theta$, Hyperparameters $\alpha, \beta, \gamma$
1: for epoch in epochs do
2:     Sample: $n_s$ source log splits $\{S_i, i = 1, ..., n_s\}$ from $\{S\}$
3:     Meta-train:
4:     for i in $n_s$ do
5:         Gradients Update: $\theta' = \theta - \alpha\nabla_\theta \mathcal{L}_S(\theta, S_i)$
6:     end for
7:     Note the MetaLog network parameters now as $\theta'_S$
8:     Sample: $n_t$ target log splits $\{T_i, i = 1, ..., n_t\}$ from $\{T\}$
9:     Meta-test:
10:    for i in $n_t$ do
11:        Calculate Loss: $\mathcal{L}_T(\theta'_S, T_i)$
12:    end for
13:    Aggregate: all i $\mathcal{L}_T(\theta'_S, T_i)$ to $\mathcal{L}_T(\theta'_S, \{T\})$
14:    Meta-learner Optimization: update $\theta$ by
15:    $\theta = \theta - \gamma \frac{\partial(\mathcal{L}_S(\theta,\{S\}) + \beta\mathcal{L}_T(\theta - \alpha\nabla_\theta\mathcal{L}_S(\theta,\{S\}),\{T\}))}{\partial\theta}$
16: end for
17: return Final MetaLog network parameters
```

contrast, meta-learning refers to a paradigm that can enhance the network's generalization ability by utilizing outer optimization to handle a broader range of meta-representations, beyond just model parameters [10, 16].

The other specific challenge for GCLAD tasks is data insufficiency. The log data from the target system is strictly restricted in practical industrial scenarios, while the source system data is not sufficient compared with other fields in the machine learning community, such as pre-training for language or vision tasks. Due to data scarcity, the transfer-learning-based methods cannot perform well since they are typically based on fine-tuning and rely on a large pre-trained feature extractor [8, 38]. Since previous baselines for cross-system log-based anomaly detection [3, 14] do not satisfy the need for sufficient pretraining data, they cannot perform well to handle more complex GCLAD tasks with larger domain gaps. However, there is solid evidence demonstrating that meta-learning necessitates less data to achieve comparable generalization results against transfer-learning [12, 13, 39], especially under circumstances of limited target data accessibility. Considering both large data domain gaps and data scarcity, the proposed MetaLog is more suitable and powerful to handle the challenging GCLAD tasks than previous methods.

3.4 MetaLog Network

In this section, it is important to provide a clear explanation of the MetaLog network architecture, as the detailed design of the network presented in Figure 4 also plays a role in enhancing its ability to solve GCLAD tasks effectively. The MetaLog network primarily consists of three interconnected modules: a gated recurrent unit (GRU), an attention mask layer, and a non-linear layer. Given a log event embeddings sequence of $\{V_{e_1}, V_{e_2}, ..., V_{e_n}\}$, where e_t refers to the t-th log event in the sequence. The GRU module maintains a hidden state H_t with the input of each time stamp. This allows the network to retain long-term information from the input log event sequence.

Inspired by [17], who demonstrates that the adaptive attention mechanism benefits the generalization ability of neural networks than static linear or convolution layers, we design an attention mask layer after the GRU module. For timestamp t, the attention module takes in hidden states of the GRU module $\{H_1, ..., H_t\}$ as input and utilizes an adaptive self-attention technique to fuse them by

$$\begin{aligned} A_{in} &= \{H_1, ..., H_t\}, \\ Q &= W_q A_{in},\ K = W_k A_{in},\ V = W_v A_{in}, \\ H_t^A &= Sum(Softmax(QK^T)A_{in}), \end{aligned} \tag{4}$$

where Sum is the sum operation from the dimension of t. We also experimentally demonstrate that such attention mechanism can help enhance the performance of GCLAD tasks (Section 4.5), in accordance with the findings in [17].

Finally, the log sequence reaches its final form H_n^A, which combines all the previous information. H_n^A is fed into a non-linear layer to generate the anomaly probability by $tanh(W_n H_n^A)$, where W_n is the network weights of the non-linear layer.

The losses used by the meta-learning-based training process of MetaLog network $\mathcal{L}_S, \mathcal{L}_T$ are both cross-entropy losses calculated by the predicted anomalies probability and the ground truth anomalies.

4 EXPERIMENTS

In our experiments, we initially conduct a thorough investigation into the bilateral generalization ability of our method and the baselines in Section 4.2. Additionally, we explore the zero-shot setting as a seemingly impossible extension of the bilateral generalization setting in Section 4.3. This particular setting poses significant challenges that are rarely addressed by current methods. To tackle this difficult scenario, we apply MetaLog and suggest potential avenues for future research.

4.1 Experimental Setup

Datasets. We conducted comprehensive experiments using four publicly available log datasets: HDFS [42], BGL [31], Thunderbird [31], and OpenStack [6]. HDFS dataset consists of 575,061 log sessions, BGL dataset has 85,576 sessions, OpenStack dataset has 3,367 log sessions, and Thunderbird dataset has 48,860 sessions. To ensure compatibility, we selected HDFS and BGL datasets for bilateral generalization, while OpenStack and Thunderbird datasets were used for zero-shot generalization. In the bilateral generalization setting, we used HDFS as the source log system, utilizing all its labels to meta-train the MetaLog network. BGL was used as the target system, with its limited anomaly labels (typically 1%) and a minority of normal labels used to meta-test the MetaLog network. Additionally, the majority of BGL data was used for the MetaLog network testing. The bilateral generalization process transferred from BGL and HDFS follows the same setting. For zero-shot generalization, OpenStack and Thunderbird datasets were used solely for testing. The MetaLog network was trained on HDFS and BGL datasets using meta-learning techniques and then directly tested on OpenStack and Thunderbird datasets.

Table 1: Bilateral generalization experiments transferring from HDFS to BGL. For our MetaLog, only 1% BGL labels are used in the training stage with the meta-learning technique.

	Method	Setting	Precision	Recall	F1-score
(a1)	PLELog [43]	Semi-supervised by 1% anomaly BGL labels	82.10	67.42	74.04
(a2)	LogRobust [47]	Supervised by 1% anomaly BGL labels	94.60	72.95	82.38
(b1)	PLELog [43]	Semi-supervised by 100% anomaly BGL labels	94.88	89.62	92.18
(b2)	LogRobust [47]	Supervised by 100% anomaly BGL labels	97.52	91.27	<u>94.29</u>
(c)	DeepLog [6]	Unpervised by BGL normal labels	66.13	48.79	56.16
(d1)	PLELog [43]	Zero-shot testing on BGL, trained by HDFS	38.80	99.87	55.89
(d2)	LogRobust [47]	Zero-shot testing on BGL, trained by HDFS	39.08	93.67	55.15
(d3)	NeuralLog [25]	Zero-shot testing on BGL, trained by HDFS	57.23	52.79	54.38
(d4)	PCA [9, 43]	Zero-shot testing on BGL, trained by HDFS	27.13	14.97	19.29
(e1)	PLELog [43]	Zero-shot testing on BGL, trained by HDFS with TCA	43.33	86.34	57.70
(e2)	LogRobust [47]	Zero-shot testing on BGL, trained by HDFS with TCA	37.72	83.99	52.06
(f1)	LogTAD [14]	Unsupervised transfer-learning based method	78.01	68.51	72.95
(f2)	LogTransfer [3]	Supervised transfer-learning based method	74.42	76.73	75.56
Ours	MetaLog	Supervised meta-learning based method	96.89	89.28	**92.93**

Table 2: Bilateral generalization experiments transferring from BGL to HDFS. For our MetaLog, only 1% HDFS labels are used in the training stage with the meta-learning technique.

	Method	Setting	Precision	Recall	F1-score
(a1)	PLELog [43]	Semi-supervised by 1% anomaly HDFS labels	65.86	71.11	68.38
(a2)	LogRobust [47]	Supervised by 1% anomaly HDFS labels	100.00	62.30	76.77
(b1)	PLELog [43]	Semi-supervised by anomaly 100% HDFS labels	96.30	83.81	<u>89.62</u>
(b2)	LogRobust [47]	Supervised by anomaly 100% HDFS labels	82.54	99.20	<u>90.11</u>
(c)	DeepLog [6]	Unpervised by HDFS normal labels	53.96	34.07	41.77
(d1)	PLELog [43]	Zero-shot testing on HDFS, trained by BGL	1.69	92.85	3.32
(d2)	LogRobust [47]	Zero-shot testing on HDFS, trained by BGL	2.25	62.12	4.35
(d3)	NeuralLog [25]	Zero-shot testing on HDFS, trained by BGL	33.13	58.04	42.06
(d4)	PCA [9, 43]	Zero-shot testing on BGL, trained by HDFS	7.21	10.32	8.58
(e1)	PLELog [43]	Zero-shot testing on HDFS, trained by BGL with TCA	3.26	91.37	6.29
(e2)	LogRobust [47]	Zero-shot testing on HDFS, trained by BGL with TCA	2.76	65.14	5.29
(f1)	LogTAD [14]	Unsupervised transfer-learning based method	78.80	71.22	74.82
(f2)	LogTransfer [3]	Supervised transfer-learning based method	100.00	43.30	60.43
Ours	MetaLog	Supervised meta-learning based method	89.29	74.98	**81.51**

For all four datasets, we follow the code provided by [43] using Drain [15] to parse the log events and organize the log sessions. This ensures that the contents of all datasets are the same as previous methods, allowing for a fair comparison. Specifically, for the HDFS-BGL bilateral transferring, 30% of the normal log sessions and only 1% of the anomaly log sessions of BGL and 30% of log sessions of HDFS are used in the training stage, to imitate the real demand of GCLAD task in industrial scenarios, *i.e.* the targeting anomaly labels are scarce. Other remaining BGL log sessions are organized for the testing stage. For the BGL-HDFS bilateral transferring, 10% of the normal log sessions and only 1% of the anomaly log sessions of HDFS and all log sessions of BGL are used in the training stage, since the capacity of the BGL dataset is much smaller than HDFS. Other remaining HDFS log sessions are organized for the testing stage. In the zero-shot generalization setting, 30% of BGL and HDFS log sessions are extracted for meta-testing and meta-training respectively, and all OpenStack and Thunderbird logs are used for testing.

Baselines. Various baselines and experimental settings are adopted to conduct GCLAD tasks including both bilateral and zero-shot generalization settings and make fair comparisons with our proposed MetaLog. Block (a) in Table 1 reveals a semi-supervised PLELog [43] and a fully supervised baseline LogRobust [47], trained by 30% of the normal log sessions and only 1% of the anomaly log sessions of BGL. Block (a) in Table 2 is trained by 10% of the normal log sessions and only 1% of the anomaly log sessions of HDFS. This block shows the capability of baseline methods only trained on target datasets with scarce anomaly labels. Block (b) in Table 1 reveals PLELog [43] and LogRobust [47] trained by 30% of the log sessions of BGL, while (b) in Table 2 is trained by 10% of the log sessions of HDFS. This block (b) exhibits baseline methods trained on target datasets with sufficient (100%) anomaly labels. Block (c) shows an unsupervised baseline DeepLog [6] trained by only normal labels (kept the same as other baselines) of target datasets. Block (d) is PLELog, LogRobust PCA [9, 43] and NeuralLog [25] under zero-shot generalization setting, which is trained only by source dataset splits (30% of log sessions of HDFS in Table 1 and all log sessions of

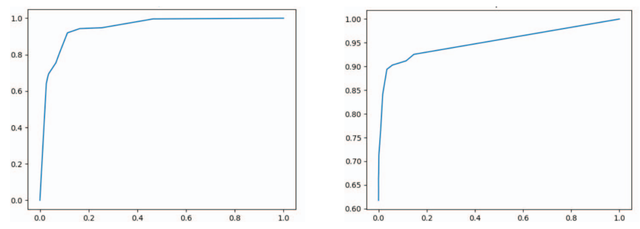


Figure 5: ROC curves on both HDFS-BGL transferring (left) and BGL-HDFS transferring (right).

BGL in Table 2) but directly tested on target dataset splits. Block (e) adds a typical transfer component analysis (TCA) technique [32] based on (d) to try to release the source and target domain gaps. Block (f) presents two transfer-learning baselines LogTAD [14] and LogTransfer [3], sharing the same source and target data setting with MetaLog. Note that all baselines above share exactly the same testing data with MetaLog for fair comparison.

Evaluation Metrics. We choose precision, recall and F1-score as the evaluation measurements, with the definition of $Precision = \frac{TP}{TP+FP}$, $Recall = \frac{TP}{TP+FN}$ and $F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$. For a fair comparison, all the competitors share the exactly same threshold of 0.5 to determine the anomaly of system logs. We show the ROC (Receiver Operating Characteristic) curve and report the corresponding AUC (Area under Curve) value as well to fully exhibit the performance of MetaLog. These can reflect the robustness of MetaLog when facing extremely imbalanced scenarios for anomaly detection.

Implementation Details. In the meta-learning-based training process, all source log data are split into 10 parts by uniform sampling. We randomly select 5 of them for each epoch to construct source log splits used by the meta-train stage of meta-tasks, *i.e.* $n_s = 5$. Since the amount of target logs is limited and much smaller than the source logs used for training, all target logs are organized into a single log split, *i.e.* $n_t = 1$. The MetaLog network is trained using the Adam optimizer on a single NVIDIA 2080Ti GPU with a batch size of 100 for 5 epochs. The hyperparameters are set as $\alpha = 2e - 3$, $\beta = 4$, and $\gamma = 2e - 3$, where β is selected based on the corresponding ablation studies in Section 4.5. Our globally consistent semantic embedding follows the settings of [33, 43] and generates input log event embeddings with a dimension of 300. As for MetaLog network architecture, 2 GRU layers with a hidden state dimension of 100 are adopted. Our code is open-sourced in *https://github.com/jia-tong-FINE/MetaLog*.

4.2 Evaluation on Bilateral Generalization Setting

Bilateral generalization is one of the most significant GCLAD tasks. Specifically, we conduct relative experiments on HDFS and BGL by regarding one of them as the source system and the other as the target system. Table 1 presents the results of transferring from HDFS (the source) to BGL (the target), with the objective of using only a small portion of BGL labels that include few anomalies for training, while achieving satisfied performance during the testing phase. Table 2 displays the results in the reverse scenario.

In general, MetaLog outperforms all other cross-system log anomaly detection methods including the direct zero-shot setting (block (d)), TCA-based transferring (block (e)), and recent transfer-learning-based methods (block (f)) by a significant margin. Furthermore, MetaLog even achieves superior results compared to (semi-)supervised methods directly trained on target log data (block (a)), surpassing unsupervised method (block (c)) and yielding comparable results with (semi-)supervised methods trained only on target data with sufficient (100%) anomaly labels, which is an ideal but challenging situation in industrial scenarios. Note that MetaLog gains such overwhelming performance by only using a minority of target log data with scarce anomaly labels.

Specifically, for bilateral generalization experiments transferring from HDFS to BGL (Table 1), MetaLog achieves significantly better results compared to recent state-of-the-art supervised transfer-learning-based method LogTransfer (f2) and unsupervised method LogTAD (f1) by 17% and 20% in F1-score respectively. This significant improvement suggests that meta-learning is more suitable than transfer-learning for GCLAD tasks, as discussed in Section 3.3. Furthermore, MetaLog presents much more superiority when compared with directly transferring methods ((d1) - (d4)) or TCA-based methods ((e1) and (e2)), by at most 40% in F1-score. In fact, MetaLog even outperforms (semi-)supervised methods ((a1) and (a2)) trained directly on target data which is the same as what MetaLog uses, by at most 18% in F1-score. This interesting result illustrates the source system information introduced by meta-learning may assist the network to gain more capacity to perform better on target log systems. As a result, the performance on target logs of MetaLog is comparable with the fully supervised LogRobust (underlined (b2)) trained by all the anomaly labels of target data, while only 1% is used by MetaLog on the contrast.

As for experiments transferring from BGL to HDFS (Table 2), similar results are observed. MetaLog gains 20% improvement compared with the current transfer-learning-based method at maximum (f2). Since the characteristic of BGL-HDFS transferring that BGL data is much more insufficient than HDFS, classic TCA-based ((e1) and (e2)) or zero-shot generalization methods ((d1) - (d4)) typically fail under this scenario. Considering methods directly trained on the target dataset, MetaLog still yields overwhelming performance under the same training setting (1% anomaly labels, (a1) and (a2)) by 13% at maximum (a1), and gains comparable results with much easier settings trained with all anomaly labels ((b1) and (b2)). Note that generally all the performance of Table 2 is lower than Table 1, including all baselines and MetaLog. This can be attributed to the fact that the capacity of BGL data is significantly less than HDFS, making transferring from BGL more challenging than from HDFS.

To fully illustrate the performance of MetaLog in extremely imbalanced anomaly detection scenarios, we exhibit the ROC (Receiver Operating Characteristic) curve and report the corresponding AUC (Area under Curve) value in Figure 5. As for HDFS-BGL transferring, MetaLog obtains an AUC of 0.948, while for BGL-HDFS, the corresponding AUC is 0.921.

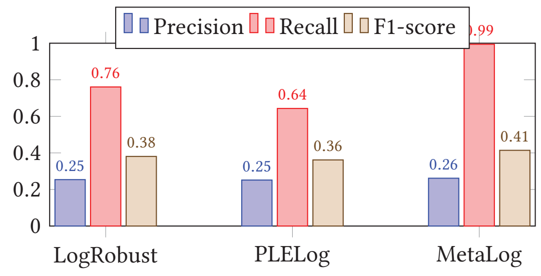


Figure 6: Zero-shot generalization experiments on OpenStack.

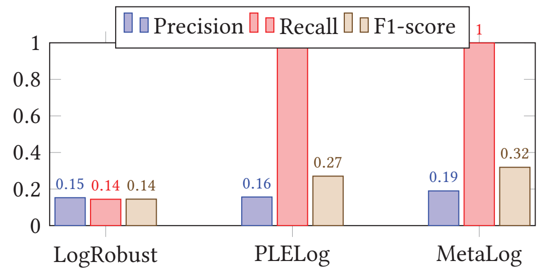


Figure 7: Zero-shot generalization experiments on Thunderbird.

4.3 Evaluation on Zero-shot Generalization Setting

Zero-shot generalization setting is an ultimate and very challenging GCLAD task, which is to directly test the network on a new target system while only trained on the log data of the source system. In this particular setting, we conducted experiments using three different methods: supervised LogRobust, semi-supervised PLELog, and our proposed MetaLog. For both LogRobust and PLELog, we utilized all HDFS logs as the source logs. These two methods were trained using HDFS data and then tested directly on either OpenStack or Thunderbird logs. As for MetaLog, we used all HDFS logs to construct the meta-training set, while 10% of the BGL logs were used for meta-testing. Finally, we evaluated the performance of MetaLog on the same split of the OpenStack logs and Thunderbird logs as the baseline methods.

Figure 6 and Figure 7 exhibit the zero-shot performance of three methods on OpenStack and Thunderbird respectively. Results show that on OpenStack, MetaLog outperforms LogRobust with a 1% higher precision, a 23% higher recall, and a 3% higher F1-score. Compared with PLELog, MetaLog has a 1% advantage in precision, a 35% advantage in recall, and a 5% advantage in F1-score. On Thunderbird, the lead gap is 5% of F1-score against PLELog and 18% against LogRobust. This demonstrates that the meta-learning technique even helps the network enhance the generalization ability towards unknown environments. For the zero-shot generalization experiments, results reveal that although MetaLog gains state-of-the-art performance, the F1-score is still far from satisfactory. The reason is that in the settings of our GCLAD problem, the logs of source systems and target systems are totally different. Thus, the data distribution bias is extremely large. The zero-shot setting trains the model simply on the logs of source systems and directly applies the model to target systems without learning any information from the target systems. As a result, the model lacks the ability to recognize new data distributions of target systems leading to unsatisfied results. The large data distribution bias is the key obstacle for the zero-shot setting [14]. Therefore, follow-up research may cast a view on it.

4.4 Run Time Analysis

To demonstrate the inference efficiency of MetaLog, we conducted a runtime experiment using a single NVIDIA 2080Ti. During the experiment, MetaLog achieved a speed of 0.113ms per sequence for each input log sequence. For the HDFS-BGL bilateral generalization experiment, we used meta-learning-based training for MetaLog, which took approximately 20 minutes. The inference on BGL, on the other hand, only required about 8 seconds. In comparison, one of the top-performing methods, PLELog, took 15 minutes for training and 8 seconds for testing. The slightly increased training time of MetaLog can be attributed to its sophisticated and effective meta-learning-based gradient update strategy. As for the supervised method, LogRobust, the training time was accelerated to 12 minutes since there was no need for probabilistic label generation. Regarding the transfer-learning-based methods, LogTransfer had a similar training time as MetaLog. However, LogTAD took longer to train (30 minutes) due to the slower convergence of the unsupervised method.

4.5 Ablation Studies

We conducted two main ablation studies on the meta-learning rate β (Figure 8) and the target anomaly labels used in the training process (Figure 9). The former provides the rationale for our selection of hyperparameters, while the latter demonstrates the robustness of MetaLog in handling extreme conditions.

In practice, the hyperparameters α and γ in meta-learning are usually selected to be the same [26]. However, the hyperparameter β plays a critical role in determining the extent of influence the network receives from the source and target data. Intuitively, a relatively larger value of β gives the network a greater influence from the target logs. However, if β is too large, it can cause the network to lose the source information, which is crucial for providing the anomaly detection paradigm through sufficient training data. As demonstrated in Figure 8, for both HDFS-BGL and BGL-HDFS bilateral generalization settings, a moderate value of $\beta = 4$ achieves the highest F1-score. On the other hand, excessively large or small values of β result in significantly inferior performance, thus confirming the intuitive reasoning mentioned above.

The results of MetaLog under different levels of scarcity in the availability of target anomaly labels during the training process are shown in Figure 9. The horizontal axis represents the percentage (ranging from 0.1% to 10%) of all anomaly labels in the target dataset. In the HDFS-BGL transferring scenario, even when the proportion of anomaly labels drops to 0.1%, MetaLog still maintains an F1-score of 84.57%, demonstrating its strong robustness in handling such extremely scarce target anomaly labels. With a 10% increase in the proportion, MetaLog achieves an F1-score of 94.97%, surpassing the performance of LogRobust trained with all BGL labels (row (b2) in Table 1). This indicates that the combination of sufficient source logs and an effective meta-learning mechanism empowers

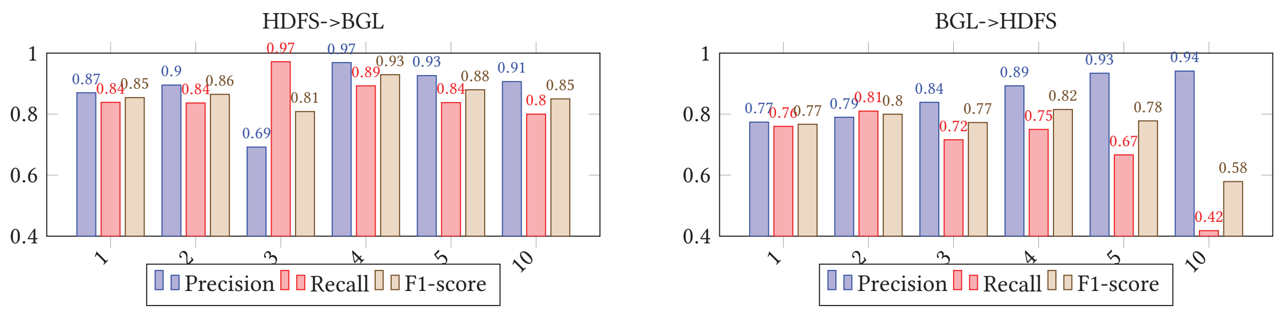


Figure 8: Ablation studies on the meta-learning rate β.

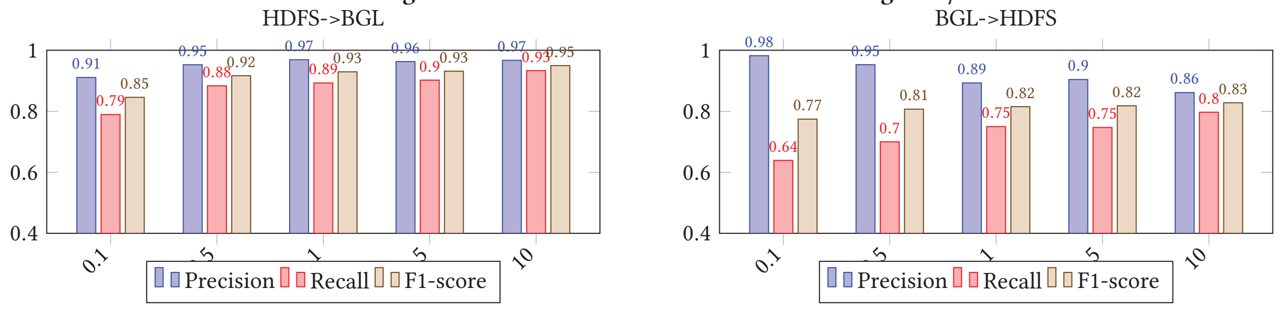


Figure 9: Ablation studies on the proportion of used training target anomaly labels.

MetaLog with a stronger ability to comprehend the log anomaly task, resulting in even better performance compared to methods supervised solely by the entire target dataset. Similarly, in the BGL-HDFS setting, the consistent F1-score ranging from 77.42% (using 0.1% HDFS anomaly labels) to 82.79% (using 10% HDFS anomaly labels) illustrates the robustness of MetaLog in handling the scarcity of target log anomaly labels in GCLAD tasks.

4.6 Threats to Validity

In our experiment on bilateral generalization, we utilized 1% anomaly log sessions for training. In the ablation study, we evaluated the performance of MetaLog using anomaly training log sessions of various sizes (ranging from 0.1% to 10%). However, due to the following considerations, we did not include the performance results of state-of-the-art baselines trained with different sizes of anomaly log sessions.

One important factor to note is that the number of training anomaly log sessions is significantly smaller (by two orders of magnitude) compared to the number of original anomaly log sessions. This substantial reduction by 99% makes it suitable for evaluating different baselines in the context of the GCLAD problem.

In addition, when using an even smaller number of anomaly log sessions for training, such as 0.1% of the total training data (specifically selecting 17 and 36 anomaly log sessions from the HDFS and BGL datasets), most state-of-the-art models struggle to complete the training process or achieve convergence with such a limited number of anomaly log sessions. Furthermore, these models fail to effectively detect anomalies in the testing datasets. In contrast, MetaLog demonstrates impressive performance with around 85% and 0.77% F1-scores using only 0.1% training anomaly log sessions. Remarkably, this outperforms the state-of-the-art models that utilize ten times the number of training anomaly log sessions. These results further emphasize the robust generalization capability of our approach.

5 RELATED WORKS

5.1 Log Anomaly Detection

Analyzing logs for problem detection and identification has been an active research area [6, 7, 20–24, 28, 29, 44, 47]. These works first parse logs into log events, and then build anomaly detection models. Some state-of-art approaches [20–23, 29] extract event sequences at first, and then generate a graph-based model to compare with log sequences in the production environment to detect conflicts. Other approaches often build deep learning-based models [7, 24, 28, 43, 44, 47] to capture the sequence features of log events. Deeplog [6] is a classic work that utilizes the LSTM network to model the sequence of log events and the sequence of variables in log text. LogAnomaly [28] utilizes a word2vec model to transform events into vectors with semantic features to improve the anomaly detection result. LogRobust [47] utilizes TF-IDF and word vectorization to transform logs into semantic vectors. In this way, updated new logs can be transformed into semantic vectors and participate in model training and deduction. PLELog[43] proposes a semi-supervised model that leverages an unsupervised clustering method to deduce data labels and construct a supervised anomaly detection model with the deduced data labels. LogTransfer[3] and LogTAD[14] propose transfer learning models where the source and target system share part of the neural network to achieve cross-system anomaly detection. However, transfer learning cannot handle situations with too large distribution gap between the source and target domain. Besides, the performance of these methods can be only guaranteed under mild assumptions and depends on how relevant the source and target data are. As a result, they are not sufficient for solving the GCLAD problem.

5.2 Meta-learning

Meta-learning is summarized as a strong technique to strengthen the generalization capacity for neural networks with the thought of '*learning to learn*' [10, 16, 50]. Current works that are close to our topic to solve GCLAD tasks include two lines of meta-learning-based tasks, domain generalization, and few-shot learning. Domain generalization concentrates on enabling neural networks to perform equally well on both source and target domains, while few-shot learning aims at transferring the network from source domains to target domains using a limited amount of target data. After [26] proposes the conception of meta-learning for domain generalization, many works follow this thought to gain success in downstream tasks such as long-tailed visual recognition [18] and person re-identification [48], as well as to develop the theory for both supervised [34] and unsupervised [30] domain generalization learning. In the line of few-shot learning, [36] first presents a strong semi-supervised few-shot classification baseline. Subsequent work [4, 19] extends this idea and makes it more efficient and robust. Meanwhile, many works aim to solve downstream few-shot learning tasks under various scenarios [27, 46, 49]. These methods of handling domain generalization and few-shot learning tasks successfully inspire us to adopt the meta-learning mechanism to solve the challenging GCLAD tasks.

6 CONCLUSION

In this paper, we propose MetaLog, a generalizable cross-system anomaly detection approach based on the meta-learning paradigm. In comparison to supervised models, MetaLog achieves comparable performance with only 1% of the data labels from the target system, as compared to models trained with 100% labeled data. Furthermore, when compared to existing transfer-learning-based cross-system anomaly detection models, MetaLog outperforms them by approximately 20% in scenarios where only 1% of the training logs from the target system are labeled.

Moving forward, our main focus will be on addressing the ultimate and most challenging zero-shot generalization scenario. To achieve this, we plan to explore the use of large language models (LLMs) to capture the features of anomalous logs from multiple mature systems. By employing prompt learning techniques, we aim to generalize the anomaly detection ability of MetaLog to new systems and achieve available performance.

ACKNOWLEDGMENTS

This work was supported by the PKU-ZTE Cooperation Research Project.

REFERENCES

[1] David Acuna, Guojun Zhang, Marc T Law, and Sanja Fidler. 2021. f-domain adversarial learning: Theory and algorithms. In *International Conference on Machine Learning*. PMLR, 66–75.

[2] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. 2010. A theory of learning from different domains. *Machine learning* 79 (2010), 151–175.

[3] Rui Chen, Shenglin Zhang, Dongwen Li, Yuzhe Zhang, Fangrui Guo, Weibin Meng, Dan Pei, Yuzhi Zhang, Xu Chen, and Yuqing Liu. 2020. Logtransfer: Cross-system log anomaly detection for software systems with transfer learning. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 37–47.

[4] Yinbo Chen, Zhuang Liu, Huijuan Xu, Trevor Darrell, and Xiaolong Wang. 2021. Meta-baseline: Exploring simple meta-learning for few-shot learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 9062–9071.

[5] Bogdan Dit, Latifa Guerrouj, Denys Poshyvanyk, and Giuliano Antoniol. 2011. Can better identifier splitting techniques help feature location?. In *2011 IEEE 19th International Conference on Program Comprehension*. IEEE, 11–20.

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

[7] Chiming Duan, Tong Jia, Ying Li, and Gang Huang. 2023. AcLog: An Approach to Detecting Anomalies from System Logs with Active Learning. In *Proceedings of the 27th IEEE International Conference on Web Services*. 1021–1030.

[8] Tiehang Duan, Mohammad Abuzar Shaikh, Mihir Chauhan, Jun Chu, Rohini K Srihari, Archita Pathak, and Sargur N Srihari. 2020. Meta learn on constrained transfer learning for low resource cross subject EEG classification. *IEEE Access* 8 (2020), 224791–224802.

[9] Ricardo Dunia and S Joe Qin. 1997. Multi-dimensional fault diagnosis using a subspace approach. In *American Control Conference*, Vol. 5. Citeseer.

[10] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*. PMLR, 1126–1135.

[11] Muhammad Ghifary, David Balduzzi, W Bastiaan Kleijn, and Mengjie Zhang. 2016. Scatter component analysis: A unified framework for domain adaptation and domain generalization. *IEEE transactions on pattern analysis and machine intelligence* 39, 7 (2016), 1414–1430.

[12] Micah Goldblum, Liam Fowl, and Tom Goldstein. 2020. Adversarially robust few-shot learning: A meta-learning approach. *Advances in Neural Information Processing Systems* 33 (2020), 17886–17895.

[13] Jiatao Gu, Yong Wang, Yun Chen, Kyunghyun Cho, and Victor OK Li. 2018. Meta-learning for low-resource neural machine translation. *arXiv preprint arXiv:1808.08437* (2018).

[14] Xiao Han and Shuhan Yuan. 2021. Unsupervised cross-system log anomaly detection via domain adaptation. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 3068–3072.

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

[16] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. 2021. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence* 44, 9 (2021), 5149–5169.

[17] Lukas Hoyer, Dengxin Dai, and Luc Van Gool. 2022. Daformer: Improving network architectures and training strategies for domain-adaptive semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9924–9935.

[18] Muhammad Abdullah Jamal, Matthew Brown, Ming-Hsuan Yang, Liqiang Wang, and Boqing Gong. 2020. Rethinking class-balanced methods for long-tailed visual recognition from a domain adaptation perspective. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 7610–7619.

[19] Muhammad Abdullah Jamal and Guo-Jun Qi. 2019. Task agnostic meta-learning for few-shot learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 11719–11727.

[20] Tong Jia, Pengfei Chen, Lin Yang, Ying Li, Fanjing Meng, and Jingmin Xu. 2017. An Approach for Anomaly Diagnosis Based on Hybrid Graph Model with Logs for Distributed Services. In *2017 IEEE International Conference on Web Services (ICWS)*. 25–32. https://doi.org/10.1109/ICWS.2017.12

[21] Tong Jia, Ying Li, Yong Yang, Gang Huang, and Zhonghai Wu. 2022. Augmenting Log-based Anomaly Detection Models to Reduce False Anomalies with Human Feedback. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 3081–3089.

[22] Tong Jia, Yifan Wu, Chuanjia Hou, and Ying Li. 2021. LogFlash: Real-time Streaming Anomaly Detection and Diagnosis from System Logs for Large-scale Software Systems. In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. 80–90. https://doi.org/10.1109/ISSRE52982.2021.00021

[23] Tong Jia, Lin Yang, Pengfei Chen, Ying Li, Fanjing Meng, and Jingmin Xu. 2017. LogSed: Anomaly Diagnosis through Mining Time-Weighted Control Flow Graph in Logs. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*. 447–455. https://doi.org/10.1109/CLOUD.2017.64

[24] Jinhan Kim, Valeriy Savchenko, Kihyuck Shin, Konstantin Sorokin, Hyunseok Jeon, Georgiy Pankratenko, Sergey Markov, and Chul-Joo Kim. 2020. Automatic Abnormal Log Detection by Analyzing Log History for Providing Debugging Insight. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice* (Seoul, South Korea) *(ICSE-SEIP*

'20). Association for Computing Machinery, New York, NY, USA, 71–80. https://doi.org/10.1145/3377813.3381371

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

[26] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy Hospedales. 2018. Learning to generalize: Meta-learning for domain generalization. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

[27] Xiaoxu Li, Zhuo Sun, Jing-Hao Xue, and Zhanyu Ma. 2021. A concise review of recent few-shot meta-learning methods. *Neurocomputing* 456 (2021), 463–468.

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

[29] Animesh Nandi, Atri Mandal, Shubham Atreja, Gargi B. Dasgupta, and Subhrajit Bhattacharya. 2016. Anomaly Detection Using Program Control Flow Graph Mining From Execution Logs. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 215–224. https://doi.org/10.1145/2939672.2939712

[30] Vaasudev Narayanan, Aniket Anand Deshmukh, Urun Dogan, and Vineeth N Balasubramanian. 2022. On Challenges in Unsupervised Domain Generalization. In *NeurIPS 2021 Workshop on Pre-registration in Machine Learning*. PMLR, 42–58.

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

[32] Sinno Jialin Pan, Ivor W Tsang, James T Kwok, and Qiang Yang. 2010. Domain adaptation via transfer component analysis. *IEEE transactions on neural networks* 22, 2 (2010), 199–210.

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

[34] Fengchun Qiao, Long Zhao, and Xi Peng. 2020. Learning to learn single domain generalization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12556–12565.

[35] Thomas Reidemeister, Mohammad A. Munawar, and Paul A.S. Ward. 2010. Identifying symptoms of recurrent faults in log files of distributed information systems. In *2010 IEEE Network Operations and Management Symposium - NOMS 2010*. 187–194. https://doi.org/10.1109/NOMS.2010.5488459

[36] Mengye Ren, Eleni Triantafillou, Sachin Ravi, Jake Snell, Kevin Swersky, Joshua B Tenenbaum, Hugo Larochelle, and Richard S Zemel. 2018. Meta-learning for semi-supervised few-shot classification. *arXiv preprint arXiv:1803.00676* (2018).

[37] Gerard Salton and Christopher Buckley. 1988. Term-weighting approaches in automatic text retrieval. *Information processing & management* 24, 5 (1988), 513–523.

[38] Qianru Sun, Yaoyao Liu, Tat-Seng Chua, and Bernt Schiele. 2019. Meta-transfer learning for few-shot learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 403–412.

[39] Richa Upadhyay, Ronald Phlypo, Rajkumar Saini, and Marcus Liwicki. 2021. Sharing to learn and learning to share-fitting together meta-learning, multi-task learning, and transfer learning: A meta review. *arXiv preprint arXiv:2111.12146* (2021).

[40] Jun Wu and Jingrui He. 2022. A unified meta-learning framework for dynamic transfer learning. *arXiv preprint arXiv:2207.01784* (2022).

[41] Wensheng Xia, Ying Li, Tong Jia, and Zhonghai Wu. 2019. BugIdentifier: An Approach to Identifying Bugs via Log Mining for Accelerating Bug Reporting Stage. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*. 167–175. https://doi.org/10.1109/QRS.2019.00033

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

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

[44] Kun Yin, Meng Yan, Ling Xu, Zhou Xu, Zhao Li, Dan Yang, and Xiaohong Zhang. 2020. Improving Log-Based Anomaly Detection with Component-Aware Analysis. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 667–671. https://doi.org/10.1109/ICSME46990.2020.00069

[45] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks? *Advances in neural information processing systems* 27 (2014).

[46] Shen Zhang, Fei Ye, Bingnan Wang, and Thomas G Habetler. 2020. Few-shot bearing anomaly detection via model-agnostic meta-learning. In *2020 23rd International Conference on Electrical Machines and Systems (ICEMS)*. IEEE, 1341–1346.

[47] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, et al. 2019. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 807–817.

[48] Yuyang Zhao, Zhun Zhong, Fengxiang Yang, Zhiming Luo, Yaojin Lin, Shaozi Li, and Nicu Sebe. 2021. Learning to generalize unseen domains via memory-based multi-source meta-learning for person re-identification. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 6277–6286.

[49] Fan Zhou, Chengtai Cao, Kunpeng Zhang, Goce Trajcevski, Ting Zhong, and Ji Geng. 2019. Meta-gnn: On few-shot node classification in graph meta-learning. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 2357–2360.

[50] Fengwei Zhou, Bin Wu, and Zhenguo Li. 2018. Deep meta-learning: Learning to learn in the concept space. *arXiv preprint arXiv:1802.03596* (2018).