Robust and Transferable Log-based Anomaly Detection

PENG JIA, MOE KLINNS Lab, Xi'an Jiaotong University, China
SHAOFENG CAI*, National University of Singapore, Singapore
BENG CHIN OOI, National University of Singapore, Singapore
PINGHUI WANG, MOE KLINNS Lab, Xi'an Jiaotong University, China Shenzhen Research Institute of Xi'an Jiaotong University, China
YIYUAN XIONG, National University of Singapore, Singapore

Log messages provide a valuable source of runtime information for ensuring the safety and consistency of systems. Recently, many machine learning and deep learning methods have been proposed to automatically detect anomalous log messages, obviating the need for manual detection by experts. However, we find that in practice, the effectiveness of existing learning-based methods is severely affected by *incomplete information* and *distribution shift*. Specifically, each log message can actually be parsed into a fixed number of key information fields, while existing methods analyze log messages using only the log event information and ignore other useful information fields that can be critical to anomaly detection. Further, the distribution of real-world log messages changes continuously due to the dynamic nature of the runtime environment and thus, a detection model conventionally trained based on the unrealistic i.i.d. assumption may not provide the expected and consistent performance.

In this paper, we present a robust and transferable anomaly detection framework RT-Log to address the above problems. To perform a comprehensive analysis of log messages, we introduce an *adaptive relation modeling* technique, which captures feature interactions among log information fields selectively and dynamically for effective and interpretable log representations. To establish its robustness and transferability, we propose a general *environment generalization* technique for learning the environment invariant representations that can generalize across different runtime environments. We evaluate the anomaly detection performance of RT-Log on large real-world datasets. Extensive experimental results demonstrate that RT-Log consistently outperforms state-of-the-art methods by a significant margin under different settings.

CCS Concepts: • **Computing methodologies** → **Artificial intelligence**; • **Mathematics of computing**; • **Applied computing**;

Additional Key Words and Phrases: Log-based Anomaly Detection, Adaptive Relation Modeling, Environment Generalization, Invariant Representation Learning

ACM Reference Format:
Peng Jia, Shaofeng Cai, Beng Chin Ooi, Pinghui Wang, and Yiyuan Xiong. 2023. Robust and Transferable Log-based Anomaly Detection. *Proc. ACM Manag. Data* 1, 1, Article 64 (May 2023), 26 pages. https://doi.org/10.1145/3588918

*Shaofeng Cai and Pinghui Wang are the corresponding authors.

Authors' addresses: Peng Jia, MOE KLINNS Lab, Xi'an Jiaotong University, No.28 Xianning West Road, Xi'an, China, pengjia@sei.xjtu.edu.cn; Shaofeng Cai, National University of Singapore, Singapore, Singapore, shaofeng@comp.nus.edu.sg; Beng Chin Ooi, National University of Singapore, Singapore, Singapore, ooibc@comp.nus.edu.sg; Pinghui Wang, MOE KLINNS Lab, Xi'an Jiaotong University, No.28 Xianning West Road, Xi'an, China and Shenzhen Research Institute of Xi'an Jiaotong University, No.19, Gaoxin South 4th Road, Shenzhen, China, phwang@mail.xjtu.edu.cn; Yiyuan Xiong, National University of Singapore, Singapore, Singapore, yiyuan@comp.nus.edu.sg.

2836-6573/2023/5-ART64
https://doi.org/10.1145/3588918

1 INTRODUCTION

Log messages record system runtime information in the form of unstructured texts captured by printing logging statements such as *println*(·) at specific points in source codes. With the ever-increasing deployment and adoption of shared infrastructures and commodity hardware such as data center networks and cloud services, bugs and vulnerabilities in the systems become more prevalent, presenting experienced attackers opportunities for attacks [30]. To mitigate losses caused by these bugs and vulnerabilities, enterprises such as Microsoft [12, 18] and Alibaba [28] perform log-based anomaly detection tasks to examine and analyze sophisticated system faults and performance failures through a variety of log messages.

Existing log-based anomaly detection methods can be grouped into three main categories: manual detection methods, traditional machine learning-based methods, and deep learning-based methods. Manual detection methods [16, 49] match keywords (e.g., "fail" and "down") based on rules predefined by experts. They may generate both false positives and negatives. Further, manual detection methods are labor-intensive and become impractical as the data scale of log messages increases. As a result, manual detection methods have been gradually replaced by learning-based methods. Traditional machine learning-based methods such as Principal Component Analysis (PCA) [63], Logistic Regression (LR) [19], Invariant Mining (IM) [33], and Log Clustering (LC) [32] automatically capture the relationship among log messages using handcrafted statistical features, and discover anomalous log messages whose features are far from the normal ones in high-dimensional space. Compared with manual detection methods, these methods are more efficient in handling large-scale data. However, as suggested by [6], the detection performance of traditional machine learning-based methods is not robust across datasets. For instance, the F1-Score of IM almost reaches 1.0 on the HDFS [63] dataset but decreases to 0.623 on the BGL [50] dataset. In recent years, deep learning models are widely adopted in various applications such as computer vision [52], speech recognition [3], and natural language processing [66], which achieve superhuman performance without the need for laborious feature engineering. Many deep learning-based methods such as DeepLog [10], LogAnomaly [42], and LogRobust [67] have been proposed to extract finer-grained representations from log messages, which achieve effective anomaly detection. For example, DeepLog utilizes a Long Short-Term Memory (LSTM) model [20] to learn sequential patterns from *log sequences*. LogAnomaly and LogRobust further incorporate more semantic information such as synonyms and antonyms in log sequence modeling. Despite higher detection accuracy, these methods only provide anomaly predictions without interpretable details. Moreover, in actual deployment, existing learning-based methods are severely affected by incomplete information and distribution shift. We shall elaborate on these two problems below.

Incomplete Information. Raw log messages can be parsed into the structured data format [5, 29] using standard parsing techniques [68]. After parsing, each structured log comprises a fixed number of information fields, and each field can either be a numerical feature or a categorical feature [4]. Figure 1 shows a typical example of a log message sequence from the HDFS [63] dataset, where each log message consists of six fields of key information, namely *date*, *time*, *pid*, *level*, *component*, and *log event*. All these fields and their interactions can be critical to anomaly detection. For example, there will be a considerable delay before the next log message can be printed when network congestion occurs, which is reflected in the fields of *date* and *time*. Also, the *level* information (e.g., "INFO", "WARNING" and "ERROR") records the significance of the corresponding log message and can thus serve as a valuable indicator for the detection. Therefore, to detect a broader range of anomalies, all information fields need to be taken into account for a more comprehensive analysis. Nonetheless, most existing detection methods [10, 42, 67] exploit only the *log event* field and ignore other key information fields, and in particular, interactions among fields.

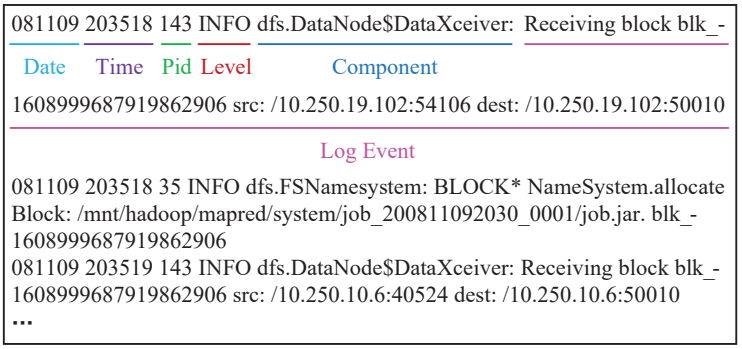
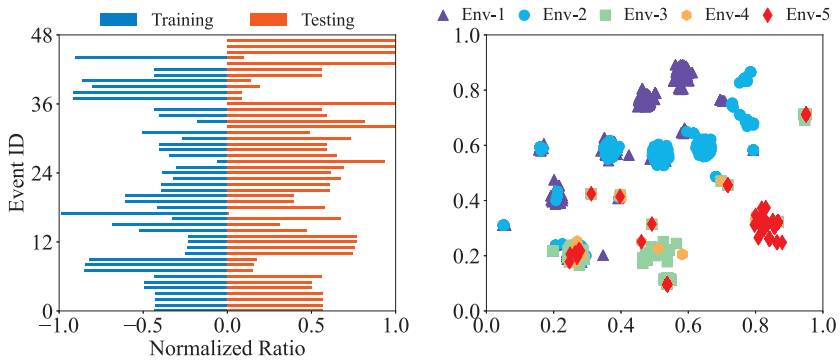


Fig. 1. An example of log messages of the HDFS dataset.

Distribution Shift. Log message sequences can be viewed as data streams collected during system runtime. Due to the volatile nature of the runtime environment, the log distribution changes continuously. This phenomenon is known as *distribution shift*. For instance, Figure 2(a) presents the normalized ratios of all 48 possible log events of HDFS in training and testing respectively. We can observe that the normalized ratios of the same event ID are significantly different in training and testing. Remarkably, some log events occur only during test time, e.g., event 45~47. Further, we partition the log sequences into five consecutive *temporal environments* based on the log collection time and visualize the event count vectors of each log sequence using t-SNE [61] in Figure 2(b). We can find that count vectors of the same environment cluster together, while the distributions of count vectors shift across environments noticeably. In real-world anomaly detection, distribution shift commonly exists, making it difficult for a conventionally trained detection model to guarantee its detection performance after deployment as the runtime environment at test time keeps changing. Existing deep learning-based detection methods [10, 42, 67] sidestep the distribution shift problem by simply shuffling the whole dataset to make training and testing data independently and identically distributed, namely making the unrealistic i.i.d. assumption.

In this paper, we propose a robust and transferable anomaly detection framework RT-Log to address the above two problems that have not been addressed by existing detection methods. First, to conduct a more comprehensive analysis of each structured log, we introduce an *adaptive relation modeling* technique that captures feature interactions among log information fields in a dynamic and selective manner for more effective and interpretable log representations. Next, to detect anomalies in structured log sequences, we propose a novel *log cross attention* mechanism for extracting more effective log sequence representations. To enable robustness and transferability in volatile deployment environments, we introduce a general *environment generalization* technique for learning environment invariant representations that can generalize across runtime environments. We summarize our main contributions as follows:

- As opposed to most existing detection methods that focus only on analyzing the *log event* field, we model structured logs using all information fields and introduce the *adaptive relation modeling* and *log cross attention* techniques to extract effective and interpretable representations for log-based anomaly detection.
- To the best of our knowledge, we are the first to address the distribution shift problem for log-based anomaly detection. We propose a general environment generalization technique *EG-Regularization* for learning environment invariant representations, which greatly improves the robustness and transferability of detection models across environments.



(a) training/testing distribution mismatch (b) distribution shift across environments

Fig. 2. Distribution shift of the HDFS dataset.

- We develop a robust and transferable framework that takes raw log messages as inputs and provides more effective and interpretable anomaly detection. Our RT-Log is way more effective than state-of-the-art detection methods, especially for the small training data scenario.
- We conduct extensive experiments on two large real-world datasets, namely BGL [50] and HDFS [63], to evaluate the performance of RT-Log. Experimental results show that RT-Log consistently outperforms prior arts by a significant margin under different scenarios. Notably, compared to the state-of-the-art methods such as DeepLog [10], LogAnomaly [42] and LogRobust [67], our model improves the detection F1-Score by up to 69.54%, 69.27% and 22.71%, respectively.

The rest of this paper is organized as follows. The background and related works are presented in Section 2. Section 3 introduces the log-based anomaly detection framework and technical details of RT-Log. We provide extensive experimental evaluations with discussions and visualizations in Section 4, and conclude in Section 5.

2 PRELIMINARIES

We illustrate an end-to-end workflow of log-based anomaly detection in Figure 3, which consists of four major phases, namely *log collection*, *log parsing*, *log partitioning and representation*, and *anomaly detection*. We shall introduce preliminaries and related works in this section. We note that log collection mainly studies how to effectively collect log messages from system consoles or files without affecting normal operations in large-scale computer systems, which is not the focus of this paper.

2.1 Log Parsing

Log parsing aims to convert raw unstructured log messages into *structured logs*. As the example shown in Figure 3, with log parsing techniques, each log message is parsed into two parts, a *message header* and a *log event*. The message header consists of a fixed number of fields, e.g., *time, date, pid, level*, and *component* for the HDFS dataset. This part depends on the logging framework and can be easily extracted using standard parsing techniques [67, 68]. In contrast, the format of the log event varies across logging systems, which reflects the intrinsic behavior of the systems. Usually, a log event is derived from a constant *log event template* and multiple variable *parameters*. Parameters in a log event are replaced by "<*>" in the log event template.

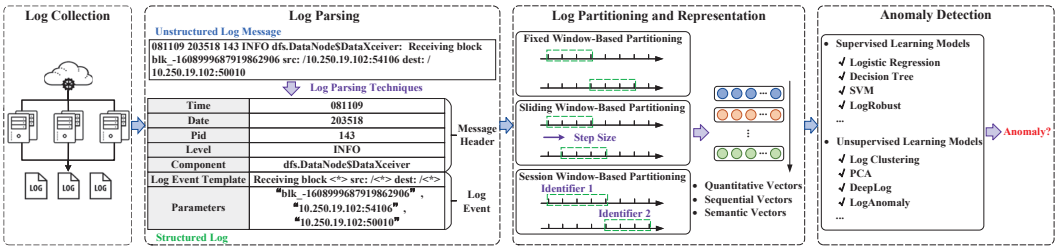


Fig. 3. The workflow of log-based anomaly detection.

Zhu et al. [68] summarize tools and benchmarks of existing log parsing techniques, which can be grouped into five categories, namely *manual parsing*, *frequent pattern mining*, *clustering*, *heuristic mining*, and *others*. Manual parsing splits log event templates and parameters based on handcrafted rules, which is time-consuming and error-prone, especially when the scale of log messages is large or increasing. Frequent pattern mining techniques such as LFA [46], SLCT [59], and Log Cluster [60] extract log event templates that frequently occur in historical log messages. These techniques firstly need to traverse all historical log messages several passes to obtain frequent itemsets. Then, they group frequent itemsets into several clusters and extract a representative log event template for each cluster. Further, since log event templates are constant parts that remain unchanged across occurrences, clustering techniques such as LKE [11], LogSig [57], LogMine [15], SHISO [45], and Lenma [56] treat the log parsing phase as log message clustering. In this case, log messages within the same cluster have a closer distance from each other, and thus are considered to be generated from the same log event template. Heuristic mining techniques such as AEL [23], IPLoM [39], FT-Tree [65], and Drain [17] propose to utilize the characteristics of log messages to separate the constant templates and variable parameters. For example, Drain [17] represents log messages with a fixed-depth tree structure, which also guides how to divide log messages into different clusters. In addition to the above four categories of log parsing techniques, Spell [9] proposes to find the longest common sub-sequences as log event templates. MoLFI [43] treats the log parsing phase as a multi-objective problem and uses an evolutionary approach to solve this problem. To improve the adaptiveness in both an intra-service and a cross-service manner, LogParse [41] adopts an incremental learning strategy and utilizes a word classification model to update log event templates continually. Also, NuLog [47] formulates the log parsing phase as masked language modeling and proposes a self-supervised learning model to extract numerical feature vectors for log event templates.

2.2 Log Partitioning and Representation

Log Partitioning. After log parsing, we partition the structured logs into *log sequences* for anomaly detection, as detecting anomalies on a sequence of logs is much more effective than detecting anomalies using one single log. There are mainly three categories of *log partitioning* strategies depending on the anomaly detection settings, i.e., *fixed window-based partitioning*, *sliding window-based partitioning*, and *session window-based partitioning*.

Fixed window-based partitioning and sliding window-based partitioning are based on the *timestamp*/*time* field of each structured log; while session window-based partitioning is based on the *identifier* field of each structured log. To be specific, a fixed window-based partitioning uses a pre-defined window size to divide all logs into log sequences in the temporal order. Each sequence comprises a fixed number of structured logs, and consecutive sequences have no overlap. As for

the sliding window-based partitioning, besides the pre-defined window size, this strategy also introduces a *step size* to set the temporal forwarding distance. On the contrary, the session window-based partitioning strategy partitions structured logs of the same identifier, e.g., *block_id* for the HDFS dataset, into one log sequence, whose log sequences contain varying numbers of structured logs.

Log Sequence Representation. After log partitioning, we convert each log sequence into numerical feature vectors for the subsequent anomaly detection using a machine learning or deep learning model. There are mainly three common types of feature vectors for log sequences, namely *quantitative vectors*, *sequential vectors*, and *semantic vectors* [24, 42].

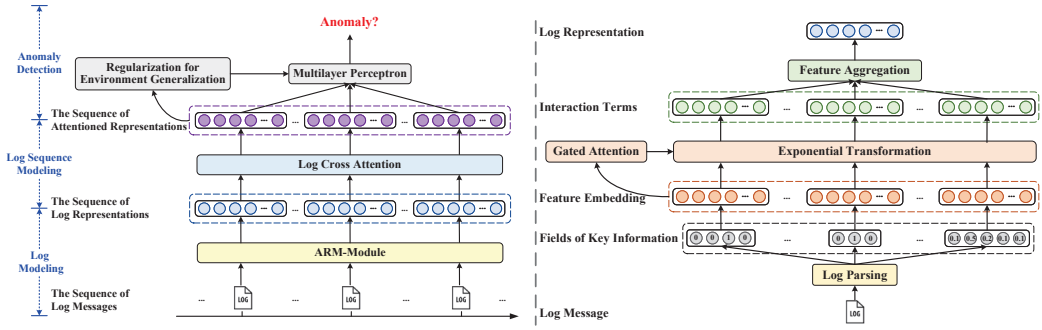
The quantitative vector calculates the statistics of the whole log sequence. Each dimension of this feature vector records the number of occurrences of a corresponding log event template (represented by a unique log event ID) in the log sequence. Instead, the sequential vector records all the structured logs in the log sequence sequentially, whose length corresponds to the number of logs in the sequence. The features extracted to represent each structured log can simply be the log event ID [10, 42]. Similar to the sequential vector, the semantic vector also records the structured log sequence, while it only records the semantic representations of each structured log. To obtain the semantic representations, we can use NLP techniques such as FastText [24] and dLCE [48] to obtain an embedding vector for each word of the log event template, then adopt the TF-IDF strategy [55] to assign an importance weight to each embedding, and finally obtain the semantic representations of the log event template by a weighted average of these embeddings.

2.3 Anomaly Detection

For detecting anomalies, we can adopt one [10, 67], or more types [42] of feature vectors of each log sequence and then train a machine learning model or a deep learning model. Based on whether the detection model is trained using labeled log sequences (i.e., normal or anomaly) or not, existing detection models can be broadly classified into *supervised learning models* or *unsupervised learning models*.

For supervised learning models, Logistic Regression (LR) [19], Decision Tree (DT) [1], and Support Vector Machine (SVM) [31] directly utilize the quantitative vector extracted from the log sequence to predict whether there exist anomalies. LogRobust [67] feeds the sequence of semantic vectors into an attention-based Bi-LSTM model [21] for detection.

As for unsupervised learning models, since labels of the log sequences in the training dataset are not available in advance, methods such as Log Clustering (LC) [32], Principal Component Analysis (PCA) [63], and Invariants Mining (IM) [33] separate normal and anomalous log sequences based on their distances. Log sequences are predicted as anomalies when they are far away from other normal ones based on a pre-determined distance threshold. More recent works [10, 42] propose to detect anomalies by predicting the next log event ID using the *subsequence* of historical log event IDs within a sliding window over the log sequence. In particular, the log sequence is predicted to be anomalous if the actual next log event ID is not in the top-k predicted log event ID candidates, where k is a model hyperparameter. For instance, DeepLog [10] employs an LSTM model to process the sequential vector. LogAnomaly [42] uses an LSTM model to process the quantitative vector, and meanwhile, introduces another LSTM model to learn the semantic patterns of each log sequence. Some similar variants include PLELog [64] and NeuralLog [26], where PLELog replaces LSTMs with GRUs [8] to process the sequential vector, and NeuralLog introduces Transformer [62] to combat out-of-vocabulary words and semantic misunderstandings when extracting semantic vectors. Also, instead of mining sequential patterns using LSTMs, [35] proposes a CNN-based model to extract local patterns of log sequences.



(a) The detection framework of RT-Log. (b) The log modeling module of RT-Log.

Fig. 4. The overview of RT-Log.

2.4 RT-Log Detection Workflow

In this work, we aim to develop an end-to-end anomaly detection framework that can handle the incomplete information and distribution shift problems. To this end, for log parsing, we will parse the raw log messages into structured logs with all the key information fields kept for a more comprehensive subsequent analysis; for log partitioning and representation, we will partition the collected long structured log sequence using a corresponding log partitioning strategy based on the adopted anomaly detection model; and then, we detect anomalies for each log sequence using a detection model. In the next section, we shall elaborate on our detection framework RT-Log, focusing mainly on our novel anomaly detection model and a general technique to further improve the robustness and transferability of the detection model.

3 METHODOLOGY

In this section, we first formally introduce the log-based anomaly detection problem. Next, we present an overview of our framework RT-Log, designed to detect anomalies for each sequence of log messages via three main modules. We provide technical details of these modules of our framework, which are based on two key techniques, namely adaptive relation modeling and an environment generalization technique, for improving its robustness and transferability across different runtime environments.

Problem Formulation. The objective of log-based anomaly detection is to predict whether a sequence of log messages contains anomalies. Formally, given a sequence of T *structured logs* $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T]$, the goal is to predict its anomaly label $y \in \{0, 1\}$, where each structured log $\mathbf{x}_t \in \mathbb{R}^M$ comprises M fields of key information extracted from its raw log message, e.g., *time*, *level* and *log event*. Specifically, each log $\mathbf{x}_t$ can be represented as a vector of categorical or numerical features, i.e., $\mathbf{x}_t = [x_{t,1}, x_{t,2}, \ldots, x_{t,M}]$.

Framework Overview. We present the overview of our RT-Log in Figure 4(a), which consists of three main modules, i.e., the log modeling module, the log sequence modeling module, and the anomaly detection module. As illustrated in Figure 4(a), we first model each *structured log* of the log sequence individually via *adaptive relation modeling* in the log modeling module, and subsequently obtain *sequence representations* of the whole log sequence via a novel *cross attention* mechanism in the log sequence modeling module for the final anomaly detection in the anomaly detection module. Specifically, *adaptive relation modeling* learns log representations by capturing

feature interactions among log information fields selectively and dynamically for effective and interpretable representations. *Log cross attention* further learns sequence representations in an effective and computationally efficient manner. We shall introduce the technical details of these modules in the following subsections.

3.1 Log Modeling Module

We first introduce the log modeling module to effectively learn a representation for each structured log in the log sequence. In anomaly detection, feature interactions among fields of each structured log capture useful information. For example, certain types of log events tend to occur at peak hours, while some components with a particular pid may generate way more warning levels than other components. These feature interactions among the log fields, namely (*log event*, *time*) and (*component*, *pid* and *level*), are key indicators for the presence of anomalies, and therefore should be exploited in the modeling.

To model feature interactions of arbitrary orders, we introduce ARM-Module based on the *adaptive relation modeling* technique [4]. ARM-Module adopts *exponential neurons* [4] to dynamically capture *cross features*, i.e., feature interactions, conditioned on the current log, and in a selective manner with uninformative log features filtered using a *gated attention mechanism*. That is, given a structured log $\mathbf{x}$ with the feature vector $\mathbf{x} = [x_1, x_2, \ldots, x_M]$, we first transform each feature value x_m into an embedding vector $\mathbf{e}_m \in \mathbb{R}^{n_p}$ by $\mathbf{e}_m = \mathbf{E}_m[:, x_m]$ for a categorical feature x_m, or $\mathbf{e}_m = x_m \cdot \hat{\mathbf{e}}_m$ for a numerical feature x_m, where n_p is the feature embedding size, $\mathbf{E}_m$ is a corresponding categorical embedding lookup matrix, and $\hat{\mathbf{e}}_m$ is a numerical embedding vector [4, 7]. Next, we introduce N_o exponential neurons to explicitly model interactions among these features via:

$$\begin{aligned}\mathbf{h}_n &= \exp(\mathbf{e}_1)^{w_{n1}} \circ \cdots \circ \exp(\mathbf{e}_M)^{w_{nM}} \\ &= \exp(\sum_{m=1}^{M} w_{nm}\mathbf{e}_m)\end{aligned} \tag{1}$$

where $\mathbf{h}_n$ is a specific feature interaction term captured by the n-th exponential neuron using the interaction weights $\mathbf{w}_n$, and $\mathbf{w}_n \in \mathbb{R}^M$ is dynamically obtained via gated attention:

$$\begin{aligned}\tilde{z}_{nm} &= \phi_{att}(\mathbf{q}_n, \mathbf{e}_m) = \mathbf{q}_n^{\mathsf{T}}\mathbf{W}_{att}\mathbf{e}_m \\ \mathbf{z}_n &= \alpha\text{-entmax}(\tilde{\mathbf{z}}_n), \tilde{\mathbf{z}}_n \in \mathbb{R}^M \\ \mathbf{w}_n &= \mathbf{z}_n \circ \mathbf{v}_n, \mathbf{v}_n \in \mathbb{R}^M\end{aligned} \tag{2}$$

where $\mathbf{W}_{att} \in \mathbb{R}^{n_p \times n_p}$ is the weight matrix for the bilinear attention function $\phi_{att}(\cdot)$ shared across exponential neurons, $\mathbf{q}_n \in \mathbb{R}^{n_p}$ is the learnable *query vector* associated with the n-th exponential neuron for the bilinear attention alignment with the input features $\mathbf{e}_m$, and thus $\tilde{z}_{nm}$ dynamically captures the relative importance of the m-th field for the n-th neuron. Particularly, α-entmax$(\tilde{\mathbf{z}}_n) = \text{argmax}_{\mathbf{p} \in \Delta^M}\{\mathbf{p}^{\mathsf{T}}\tilde{\mathbf{z}}_n + \mathbf{H}_\alpha^{\mathsf{T}}(\mathbf{p})\}$, where Δ^M is the M-dimension probability simplex, $\mathbf{H}_\alpha^{\mathsf{T}}(\mathbf{p})$ is the Tsallis α-entropies [58]: $\mathbf{H}_\alpha^{\mathsf{T}}(\mathbf{p}) = \frac{1}{\alpha(\alpha-1)}\sum_j(p_j - p_j^\alpha)$ when $\alpha \neq 1$, and $\mathbf{H}_\alpha^{\mathsf{T}}(\mathbf{p}) = \sum_j -p_j \log p_j$ when $\alpha = 1$ [51]. Unlike the softmax function [4, 40] that always generates dense outputs, the argmax solution $\mathbf{p}$ of α-entmax$(\tilde{\mathbf{z}}_n)$ here produces sparse outputs. Therefore, α-entmax$(\tilde{\mathbf{z}}_n)$ can filter out less important features of the M log fields. Further, similar to $\mathbf{q}_n$, $\mathbf{q}_n$, $\mathbf{v}_n$ is a learnable attention weight *value vector* associated with the n-th exponential neuron, which encodes *global attention weights* for each of the respective M fields.

By design, dynamic and selective attention weights $\mathbf{w}_n$ for the feature interaction term $\mathbf{h}_n$ in Equation 1 should result in a more effective and interpretable log modeling. Then, we can aggregate feature representations for each structured log $\mathbf{x}$ by aggregating these feature interaction terms $\{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_{N_o}\}$ via summation and obtain the final log representation $\mathbf{h} \in \mathbb{R}^{n_p}$.

3.2 Log Sequence Modeling Module

With log modeling, the log sequence $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T]$ is transformed into respective log representations $\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_T] \in \mathbb{R}^{T \times n_p}$. We next introduce *log cross attention* to learn log sequence representations for the final anomaly detection. In particular, n_q learnable *log query vectors* $\mathbf{Q} = \mathbb{R}^{n_q \times n_p}$ are introduced for extracting log sequence representation $\hat{\mathbf{H}}$ via cross attention [4, 22]:

$$\begin{aligned} \hat{\mathbf{H}} &= \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \\ &= \text{softmax}(\mathbf{Q}\mathbf{K}^T / \sqrt{n_p})\mathbf{V} \end{aligned} \tag{3}$$

where $\hat{\mathbf{H}} \in \mathbb{R}^{n_q \times n_p}$, $\mathbf{K} = \mathbf{H}\mathbf{W}^K$, $\mathbf{V} = \mathbf{H}\mathbf{W}^V$, and $\mathbf{W}^K, \mathbf{W}^V \in \mathbb{R}^{n_p \times n_p}$ are weights for attention key and value vectors respectively.

Different from the conventional scaled dot-product attention [22, 62], in *log cross attention*, the key and value vectors $\mathbf{K}, \mathbf{V}$ come directly from log representations $\mathbf{H}$ as in the conventional attention mechanism [62], which are dynamically generated based on the current log sequence, whereas each query vector is a learnable *global latent vector* that extracts essential log sequence representations in a corresponding latent space for the subsequent anomaly detection. Empirically, *log cross attention* proves to be far more effective in modeling log sequences than the widely adopted LSTM [10, 20, 42, 67]. Also, *log cross attention* improves efficiency over the prevailing sequence learning model Transformer [62], as the computation complexity of cross attention is linear to the sequence length instead of quadratic, i.e., $O(N_q T)$ as compared to $O(T^2)$, which can therefore scale better to the detection of longer log sequences.

3.3 Anomaly Detection Module

After extracting the representations of the whole log sequence into $\hat{\mathbf{H}} \in \mathbb{R}^{n_q \times n_p}$ using the log modeling and log sequence modeling module, we next aggregate $\hat{\mathbf{H}}$ via summation and obtain the final sequence representations $\mathbf{h} \in \mathbb{R}^{n_p}$ for anomaly detection:

$$\begin{aligned} \hat{y} = \theta_{MLP}(\mathbf{h}),\ \theta_{MLP} : \mathbb{R}^{n_p} \mapsto \mathbb{R} \\ \mathcal{L}(\hat{y}, y) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \end{aligned} \tag{4}$$

where $\theta_{MLP}(\cdot)$ is a multilayer perceptron (MLP) that produces the final anomaly prediction label, and $\mathcal{L}(\cdot, \cdot)$ is the binary cross entropy loss for model training. In this way, the whole model with the above three main modules can be optimized end-to-end using standard gradient-based optimizers such as SGD or Adam [25].

3.4 Transferable Anomaly Detection

As discussed in the introduction, logs are data streams typically collected under different runtime environments which capture essential system wellness information such as locations, times, external interventions and etc. Conventional machine learning paradigms which assume that data are independently and identically distributed, i.e., $(\mathbf{X}_i, y_i) \sim P(\mathbf{X}, y)$, will perform poorly if the i.i.d. trained model is applied to an environment with a different data distribution.

The problem is caused by a mismatch between the training environment(s) and the runtime environment, and is commonly known as *distribution shift* [2, 34], e.g., $P(\mathbf{X}_{train}) \neq P(\mathbf{X}_{test})$ as illustrated in Figure 2(a), which seriously affects the effectiveness of the trained model. To address distribution shift, the conventional wisdom is to adopt *domain adaptation* techniques [34], which however require having access to unlabeled or some labeled data from the target domain. This is unfortunately not feasible in the log-based anomaly detection scenario as the runtime distribution of logs keeps changing all the time.

Formally, log data collected under a specific environment e can be denoted as $\mathcal{D}^e = \{(\mathbf{X}_i^{(e)}, y_i^{(e)})\}_{i=1}^{n_e}$ with n_e labeled log sequences. In log-based anomaly detection, one key observation is that the environment information is readily available, e.g., when, where and how the log data are collected. Figure 2(b) illustrates the visualization of distribution shifts over time, namely log data under different temporal environments.

Environment Generalization. To obtain a more robust and transferable anomaly detection model across runtime environments, we propose to preserve the log environment information and introduce a general regularization technique to extract environment invariant representations that can generalize well to unseen environments. Specifically, environment generalization can be formulated as follows: given E log training environments $\mathcal{D}^{(e)} = \{(\mathbf{X}_i^{(e)}, y_i^{(e)})\}_{i=1}^{n_e}, e \in \{1, 2, \ldots, E\}$, the learning objective is

$$\min_f \mathcal{R}^{(E+1)}(f) = \mathop{\mathbb{E}}_{(\mathbf{X},y)\sim\mathcal{D}^{(E+1)}} [\mathcal{L}(f(\mathbf{X}), y)] \tag{5}$$

which basically requires that the learning model $f(\cdot)$ should perform well in the unseen new environment E+1. This is a new learning paradigm coined as *domain generalization* [2, 14, 54]. To make the learned log sequence representations more environment invariant across different runtime environments, we introduce EG-Regularization based on *invariant risk minimization* [2] to log-based anomaly detection. Recall that our anomaly detection model f can be decomposed into two sub-models: $f = \theta \circ \phi$, where $\phi : \mathbf{X} \rightarrow \mathbf{h}$ is the log sequence modeling sub-model, i.e., the first two modules of RT-Log, and $\theta : \mathbf{h} \rightarrow y$ is the prediction module. Formally, EG-Regularization is to achieve the following:

$$\begin{aligned} &\min_{\phi,\theta} \frac{1}{E} \sum_{e=1}^{E} \mathcal{R}^{(e)}(\theta \circ \phi) \\ &\text{s.t.} \quad \theta \in \operatorname*{argmin}_{\hat{\theta}} \mathcal{R}^{(e)}(\hat{\theta} \circ \phi), \forall e \in \{1, 2, \ldots, E\} \end{aligned} \tag{6}$$

which is a loss minimization objective function with the constraint that the prediction module θ should be *optimal* for every observable environment given the log sequence feature extractor ϕ. To achieve this, ϕ need to extract only environment invariant features, otherwise the prediction module θ is unlikely to be optimal across different environments [2, 54]. This is a challenging bi-leveled optimization problem, which however, can be approximated via *Lagrangian relaxation*:

$$\min_{\phi,\theta} \frac{1}{E} \sum_{e=1}^{E} \mathcal{R}^{(e)}(\theta \circ \phi) + \lambda \left\| \nabla_\theta \mathcal{R}^{(e)}(\theta \circ \phi) \right\|_2^2 \tag{7}$$

where λ controls the regularization strength, and the second term here is to regularize the norm of the gradient of the loss $\mathcal{R}^{(e)}$ w.r.t. θ that uses the extracted log sequence representations for anomaly prediction. Such a gradient norm loss is consistent with the first-order optimality condition, which

requires the gradient to be zero at minimum, and λ is used to balance the relative strength between the two loss terms.

We note that distribution shift is a common phenomenon in real-world anomaly detection. Such a continuous change in the runtime distribution of incoming log sequences makes it difficult for a conventionally trained detection model to perform well. To make the detection model more robust and transferable across runtime environments of different distributions, we explicitly propose EG-Regularization to facilitate learning environment invariant representations. These representations will be more resilient to environment changes, and therefore, can support more reliable modeling of log sequences for anomaly detection. Further, our EG-Regularization is general to anomaly detection models and can be readily integrated into the existing detection framework for improving detection robustness and transferability.

3.5 Analysis and Discussion

Interpretability. The deployment of detection models is in great need of interpretability, which measures the extent to which the detection results can be understood and examined by human [4, 13, 44]. Besides learning more robust and transferable representations for anomaly detection, another significant advantage of RT-Log over existing detection methods is that our model can provide both global and local interpretability along with anomaly predictions. Notably, exponential neurons provide a reliable means for understanding the general behavior of RT-Log, i.e., *global interpretability*, and explain the reasons why specific anomaly predictions are made, i.e., *local interpretability*, via the *adaptive relation modeling* technique.

Specifically, attention weight value vectors $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{N_o}]$, $\mathbf{v}_n \in \mathbb{R}^M$ in Equation 2 shared across log sequences encode the *global attention weights* before attention gating for respective information fields. Hence, these value vectors indicate the general focus of exponential neurons on respective information fields. We can therefore aggregate the absolute values of $\mathbf{V}$ for global interpretability. Likewise, we can also aggregate the dynamically determined interaction weights $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_{N_o}]$, $\mathbf{w}_n \in \mathbb{R}^M$ of each log sequence in Equation 2 to obtain the relative importance of respective fields in making a specific anomaly prediction for local interpretability.

Efficiency. In log-based anomaly detection, the complexity of the detection model is another important criterion as the detection usually requires immediate responses to prevent irreversible damage. Hence, we also analyze the parameter and computational efficiency of RT-Log, focusing on the three main modules, namely the log modeling module, the log sequence modeling module, and the anomaly detection module.

For modeling each individual structured log in the log modeling module, the feature embedding takes totally $O(N_F n_p)$ parameters, with N_F and n_p being the number of distinct features and the embedding size respectively, among which only M embedding vectors are used for the computation of each log. Hence, the parameter size and the computation are both in $O(TMn_p)$ for the embedding of T structured logs. Then, the query and value vectors for the N_o exponential neurons take $O(N_o n_p)$ parameters, and the computational complexity of adaptive relation modeling is $O(TN_o M n_p^2)$. For the log sequence modeling module, the log cross attention via n_q log query vectors takes $O(n_q n_p)$ parameters and $O(Tn_q n_p^2)$ computation. Finally, for detecting anomalies in the anomaly detection module, the computational complexity is $O(n_w)$, where n_w is the MLP parameter size. Therefore, the overall parameter and computational complexity are $O(n_p(TM + N_o + n_q) + n_w)$ and $O(Tn_p^2(N_o M + n_q) + n_w)$ respectively, which scale linearly to the sequence length T and the number of information fields M, and thus are quite efficient [4].

Table 1. Dataset statistics.

Dataset	Duration	#Log Messages	#Anomalies	#Fields	#Features	#Event Types
HDFS	38.7 hours	11, 175, 629	16, 838 blocks	7	28, 002	48
BGL	214.7 days	4, 747, 963	348, 460 logs	9	71, 363	1845

4 EXPERIMENTS

In this section, we first introduce the benchmark datasets and experimental settings. We then present the overall performance of our RT-Log compared with baseline models, and robustness and transferability analysis of RT-Log. We subsequently present efficiency and interpretability studies for a better understanding of the anomaly detection results of RT-Log.

4.1 Datasets

In our experiments, we adopt two large public datasets to evaluate the performance of RT-Log in comparison with other state-of-the-art models. The statistics of the two datasets are summarized in Table 1, and the details are as follows:

• **HDFS** is generated by Hadoop-based map-reduce jobs on more than 200 Amazon EC2 nodes. It consists of 575, 061 blocks, where each block involves a sequence of log messages that are collected during the execution of a program, e.g., writing, closing, or deleting files. Each log message includes a *block_id* to identify its corresponding block, and each block in HDFS is labeled as normal or anomaly by experienced Hadoop domain experts. There are 11, 175, 629 log messages with 16, 838 anomalous blocks in HDFS. HDFS contains seven fields of key information and 28, 002 different features in total, with fields of *hour*, *minute*, *second*, *pid*, *level*, *component*, and *log event template*. Specifically, the fields of hour, minute, and second are extracted based on the timestamp of the log message.

• **BGL** is collected from a BlueGene/L supercomputer system with 131, 072 processors at Lawrence Livermore National Labs. Each log message is labeled as non-alert with a "-" tag, and alert otherwise. BGL comprises 4, 747, 963 log messages and 348, 460 of them are anomalous. There are nine fields of key information and 71, 363 features in BGL, with fields of *hour*, *minute*, *second*, *millisecond*, *node*, *type*, *component*, *level*, and *log event template*. The fields of hour, minute, second, and millisecond are generated again from the log timestamp.

For HDFS, we partition raw log messages with the same identifier *block_id* into one log sequence. While for BGL, since log messages contain no identifiers for partitioning, we instead partition its log messages into log sequences via sliding window partitioning. Specifically, we fix the partition window size to be 100 logs and the forwarding step size to be 20 logs in all the experiments. A log sequence is labeled as an anomaly if the sequence contains at least one alert log message for BGL.

4.2 Experimental Settings

4.2.1 Preprocessing.

Log Parsing. According to the experimental results reported by [68], Drain [17] achieves higher parsing accuracy and is more efficient in comparison with other log parsing techniques. Therefore, before feeding raw log message sequences into our detection framework for analysis, we first parse these unstructured log messages into structured logs of a fixed number of information fields using the Drain parser. We then partition these structured logs into log sequences using a corresponding partitioning strategy.

Field Embedding. All fields in the message header, e.g., *time* and *pid*, are further preprocessed into corresponding embedding vectors via embedding lookup or linearly-scaled embedding [4, 7, 37, 38] respectively before modeling via RT-Log. Particularly, for the log event template field, we utilize the FastText [24] model to obtain a semantic embedding vector.

4.2.2 Baseline models. We compare our RT-Log with a machine learning-based model Logistic Regression (LR) [19], and three deep learning-based models, DeepLog [10], LogAnomaly [42], and LogRobust [67]. We briefly introduce these baseline models as follows:

• **LR** [19] takes the quantitative vector of each log sequence as the input and directly predicts whether the log sequence contains anomalous log messages or not. Each dimension of the quantitative vector counts the number of occurrences for the corresponding log event ID.

• **DeepLog** [10] detects log anomalies in an autoregressive fashion via an LSTM model, which sequentially predicts the next *log event ID* using the *subsequence* of historical log event IDs within a sliding window over the log sequence. The whole log sequence is detected as an anomaly if any actual next log event ID is not in the top-k predicted candidates.

• **LogAnomaly** [42] is also an autoregressive model based on the next log event ID prediction. LogAnomaly introduces two LSTM models for processing semantic vectors and quantitative vectors of the *subsequence* respectively. Semantic vectors are obtained via a Template2Vec model, which considers synonyms and antonyms and aggregates word embedding vectors for each log event template based on their TF-IDF weights. Quantitative vectors again count the number of occurrences of log event IDs within the sliding window.

• **LogRobust** [67] adopts an attention-based Bi-LSTM to model the log sequence based on the semantic vectors of its structured logs. LogRobust first uses the FastText model to produce an embedding vector for each word in the log event template and aggregates these vectors using their TF-IDF weights as the final semantic embedding of the log message. The anomaly prediction of the log sequence is achieved by aggregating the Bi-LSTM hidden states via attention and then predicting using an MLP.

4.2.3 Evaluation Metrics. Following the conventions [10, 42, 67], we adopt four metrics, Recall, Precision, F1-Score, and Accuracy, to evaluate the performance of our RT-Log in comparison with the other baseline models, which are defined as follows:

- **Recall** is the ratio of correctly detected anomalies to all real anomalies. Recall = $\frac{TP}{TP+FN}$.
- **Precision** is the ratio of correctly detected anomalies to all detected anomalies. Precision = $\frac{TP}{TP+FP}$.
- **F1-Score** is the harmonic mean of Recall and Precision. F1-Score = $\frac{2\times\text{Recall}\times\text{Precision}}{\text{Recall+Precision}}$.
- **Accuracy** is the ratio of correctly detected log sequences (including normal and anomalous log sequences) to all log sequences. Accuracy = $\frac{TP+TN}{TP+FP+FN+TN}$. For next event ID prediction based detection, accuracy is the ratio of correctly predicted next event ID.

In the above evaluation metrics, TP is the number of correctly detected anomalies. FP is the number of normal log sequences that are falsely detected as anomalies. TN is the number of correctly detected normal log sequences. FN is the number of anomalies that are not detected.

4.2.4 Hyperparameter Settings & Implementation Details. For a fair comparison, we evaluate all the models in the same experimental settings following common practice [10, 42, 67]. Notably, for the next event ID prediction based detection models, the number of candidates k is the best in 1∼15. We adopt the Adam optimizer with a learning rate grid searched in $1e^{-4}$∼$1e^{-1}$ and train all models on three GPUs with a batch size of 256.

Table 2. Overall anomaly detection performance using small and standard training data sizes.

Model	HDFS								BGL							
	training data size 20%				training data size 80%				training data size 20%				training data size 80%			
	Accuracy	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score
LR	0.9964	0.8711	0.9938	0.9305	0.9995	1.0000	0.9661	0.9827	0.8065	0.1145	0.4305	0.1804	0.9009	0.4667	0.8381	0.6139
DeepLog	0.9904	0.7834	0.9884	0.8740	0.9985	0.8556	0.8971	0.8747	0.5933	0.1154	0.9594	0.2060	0.5451	0.1575	0.9522	0.2703
LogAnomaly	0.9909	0.7939	0.9882	0.8804	0.9982	0.7992	0.9262	0.8582	0.6425	0.1245	0.9602	0.2205	0.5480	0.1593	0.9455	0.2727
LogRobust	0.9951	0.8573	0.9212	0.9016	0.9993	0.9840	0.9829	0.9772	0.9661	0.8613	0.4895	0.6172	0.9909	0.9891	0.9102	0.9453
RT-Log	0.9986	0.9589	0.9820	**0.9707**	0.9999	1.0000	0.9920	**0.9960**	0.9829	0.8823	0.8175	**0.8443**	0.9940	0.9530	0.9771	**0.9654**

For our RT-Log, in particular, the embedding size is 64, the sparsity of ARM-Module α is 2.0/1.7, the number of exponential neurons N_o is 32/16, the number of query vectors N_q of cross attention is 8/128 for HDFS and BGL respectively. We note that RT-Log is not sensitive to the hyperparameter settings whose detection performance is rather consistent empirically.

Experimental Environment. Experiments are conducted in a server with Xeon(R) Gold 6248R CPU @ 3.00GHz (24x2 Cores), 791G memory and 8 Tesla V100 SXM2 32GB. Models are implemented in PyTorch 1.10.0 with CUDA 10.1.

4.3 Overall Anomaly Detection Performance

We first evaluate the overall anomaly detection performance of RT-Log in comparison with other state-of-the-art baseline models. The experimental results are summarized in Table 2. Here, we introduce the EG-Regularization technique to the training of all models to facilitate learning environment invariant representations, which makes these models more resilient to distribution shift as the runtime environment changes. In particular, we consider the following two scenarios: (1) we only have a small number of labeled log sequences available for training; (2) we have enough labeled log sequences that can be used for training. The former scenario is more common in practice as typically labeling log sequences requires expert knowledge and takes a considerable amount of manual labor [6, 27]. On the other hand, the latter is a standard setting for benchmarking machine learning and deep learning anomaly detection models [3, 52, 66]. In the experiments, we use "*small training data size*" and "*standard training data size*" to denote the above two respective scenarios, where the training data size is set to 20% and 80% of the entire dataset respectively.

From Table 2, we can observe that RT-Log achieves overall the best anomaly detection performance among all models. Specifically, under the *standard training data size*, RT-Log obtains up to 0.9960 and 0.9654 F1-score for the HDFS and BGL datasets respectively, which is substantially higher than that of prior arts on this primary benchmark metric. For the HDFS dataset, all models can accurately detect anomalies and achieve an F1-Score higher than 0.8. While for the BGL dataset, most baseline models perform significantly worse, e.g., the F1-scores of DeepLog and LogAnomaly are only 0.2703 and 0.2727 respectively, although both of them obtain a high Recall score. The lower detection F1-Score on the BGL dataset is consistent with the results reported by prior works [42, 67], and can be mainly attributed to a larger number of possible log event templates, i.e., 1845 for BGL as compared to 48 for HDFS, and a greater mismatch between the training and testing data distribution, which will be discussed in the ablation studies in Section 4.4. By contrast, RT-Log can still perform well on the more difficult BGL dataset with an F1-Score over 0.95.

Further, we can find that RT-Log outperforms all the baseline models by a wider margin using the *small training data size setting* than the *standard training data size setting*. We note that the scenario of small training data size is of greater practical value since obtaining training data typically requires expensive labeling by experts. Empirically, RT-Log obtains an F1-Score of 0.9707 and 0.9654 for the HDFS and BGL datasets respectively using only a quarter of the standard training data size

for training, which achieves 0.0402 and 0.2271 improvement over the corresponding best baseline models, i.e., 0.9305 of LR for HDFS and 0.6172 of LogRobust for BGL. When given more training data, all the log-based anomaly detection models can be exposed to more types of anomalies and thus can learn more effective and reliable patterns. This suggests that all the detection models could obtain reasonably good detection performance with a standard training data size. However, we find that trained using the standard data size, RT-Log still improves the F1-Score of the best baseline models by 0.0133 and 0.0201, i.e., 0.9827 of LR for HDFS and 0.9453 of LogRobust for BGL respectively. As noted in [10, 42, 67], an F1-Score improvement greater than 0.01 can be regarded as significant, which indicates that RT-Log is able to outperform baselines significantly even on larger training datasets.

Another notable finding is that the detection performance of existing baseline models is not consistent across datasets and training data sizes. For instance, DeepLog and LogAnomaly achieve a high F1-Score of 0.8747 and 0.8582 respectively for the HDFS dataset, while the F1-score drops dramatically to 0.2703 and 0.2727 for BGL. Also, under the standard training data size, LogRobust achieves an F1-Score of 0.9772 and 0.9453 for the HDFS and BGL datasets respectively, whereas the detection F1-Score decreases by 7.74% to 0.9016 and 34.71% to 0.6172 for HDFS and BGL respectively when using 20% of the labeled log sequences for training. In comparison, the detection performance of RT-Log is quite stable across datasets and training data sizes. Remarkably, even using only 20% of training data, RT-Log can still obtain a very competitive F1-Score of 0.9707 and 0.8443 for HDFS and BGL respectively, which is comparable to that of baseline models using four times as much training data.

The difference in the detection performance mainly results from two aspects, namely the log information used for anomaly detection and the way how the information is processed. LR, DeepLog, LogAnomaly, and LogRobust model each log sequence using only the *log event template* field, which ignores a considerable amount of information contained in other fields that can be exploited for detecting anomalies. While RT-Log retains all the information available by taking all the fields of each structured log as the input for detection, and further, processes the information by extracting key feature interactions among these fields dynamically and selectively using adaptive relation modeling for a more comprehensive log sequence modeling. As we will show in Section 4.4 and Section 4.5, the extra and more detailed information not only makes the detection results more effective and interpretable, but also greatly improves the robustness and transferability, as the log representations are now extracted more reliably from the complete structured log instead of only part of the log. Also, the EG-Regularization technique explicitly regularizes the log sequence modeling process of RT-Log, which makes the learned representations more resistant to distribution shift and improves the effectiveness of detection.

4.4 Robustness and Transferability Analysis

For log-based anomaly detection, many factors can profoundly affect the performance of detection models, e.g., the problem of distribution shift, the difficulty of detection tasks, the size of the training data, the robustness of detection models, etc. In this section, we therefore adopt two markedly different real-world datasets, HDFS and BGL, to empirically evaluate the impact of these factors on the detection performance of RT-Log and baseline models.

We summarize the statistics of the two datasets in Table 1. Briefly, HDFS comprises a larger number of log messages collected during a much shorter period of time, i.e., 38.7 hours vs. 214.7 days, while each log message of BGL consists of more information fields and also more distinct features. In particular, BGL contains way more types of log event templates than HDFS. There are 1845 types in BGL compared to 48 types in HDFS. As a result, BGL has a larger number of unseen log event templates that only occur during testing and a greater distribution mismatch between

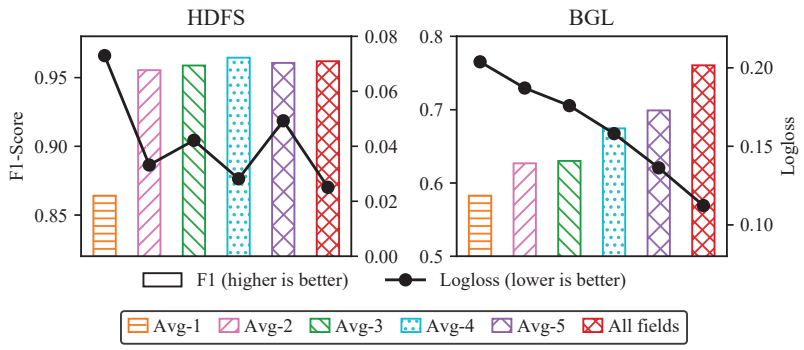


Fig. 5. The impact of introducing more fields of information to the log modeling of RT-Log.

the training and testing data. These key differences make it difficult for existing anomaly detection models to provide stable performance when they are deployed for the testing environment on BGL, as also reported by prior works [6, 19, 27].

In what follows, we will examine the impact of several key factors on the detection performance, including extra information fields, the new EG-Regularization technique, and the size of training data, on the performance of RT-Log and all baseline models on HDFS and BGL datasets respectively.

Impact of Extra Information Fields. To assess the impact of utilizing more information fields on the anomaly detection performance, we vary the number of information fields that are accessible to RT-Log for the modeling of structured logs, and then report the average anomaly detection results obtained using the same number of fields. We set the training data size to 20% and evaluate the performance of RT-Log using metrics of F1-Score and Logloss (binary cross entropy loss). Figure 5 illustrates the experimental results, where *Avg-n* denotes the average detection performance considering all combinations of the *n* fields. A higher F1-Score and a lower Logloss indicate better performance.

From Figure 5, we can observe that as we introduce more fields of information, the anomaly detection performance of RT-Log improves significantly. In particular, when using only one field, the average F1-Score is 0.8641 and 0.5827 respectively for the HDFS and BGL datasets, whereas the F1-Score can be substantially improved by 10.57% and 32.23% to 0.9554 and 0.7605 respectively by using all information fields available. Meanwhile, the Logloss is decreased from 0.0730 and 0.0251 to 0.2038 and 0.1123 for the HDFS and BGL datasets. Notably, the performance improvement by exploiting more information fields is more prominent and consistent for the BGL dataset than HDFS as we vary the number of fields. This is mainly due to the fact that RT-Log has already achieved a very high F1-Score for HDFS and thus further improvement is difficult.

These observations are consistent with the intuition behind our detection framework. Specifically, besides the field of log event templates, other fields such as *level, time* and the feature interactions among different fields such as (*component, pid, level*) have a strong correlation with attacks, systemic faults, and other types of anomalies. In the log modeling module of RT-Log, the exponential neurons capture the feature interactions among these key information fields adaptively via the gated attention mechanism, which greatly improves the effectiveness of RT-Log for more comprehensive modeling of the log sequences for anomaly detection.

Impact of EG-Regularization. Besides performing a more comprehensive analysis on all structured log fields via adaptive relation modeling, we also explicitly introduce the EG-Regularization

Table 3. Impact of introducing EG-Regularization to anomaly detection models in small and standard training data size settings. EG-R denotes the EG-Regularization technique. F1-Score and the percentage of improvement are reported.

Model	HDFS						BGL					
	training data size 20%			training data size 80%			training data size 20%			training data size 80%		
	w/o EG-R	w/ EG-R	Imp.	w/o EG-R	w/ EG-R	Imp.	w/o EG-R	w/ EG-R	Imp.	w/o EG-R	w/ EG-R	Imp.
LR	0.9302	0.9305	0.03%	0.9731	0.9827	0.99%	0.1792	0.1804	0.7%	0.4854	0.6139	26.47%
DeepLog	0.8737	0.8740	0.03%	0.8222	0.8747	6.39%	0.2031	0.2060	1.4%	0.2682	0.2703	0.78%
LogAnomaly	0.5956	0.8804	47.82%	0.8251	0.8582	4.01%	0.2148	0.2205	2.7%	0.2689	0.2727	1.41%
LogRobust	0.8713	0.9016	3.48%	0.9745	0.9772	0.28%	0.5668	0.6172	8.9%	0.9364	0.9453	0.95%
RT-Log	**0.9530**	**0.9707**	1.86%	**0.9812**	**0.9960**	1.51%	**0.7266**	**0.8443**	16.2%	**0.9644**	**0.9654**	0.10%

technique to combat the problem of distribution shift. We evaluate the impact of introducing EG-Regularization to different anomaly detection models in both small and standard training data size settings and summarize their detection F1-Score and the percentage of improvement in Table 3.

We have the following findings. First, EG-Regularization can effectively improve the performance of all anomaly detection models on both datasets in the two training data size settings. Notably, the F1-Score of LR is improved by 26.47% from 0.4854 to 0.6139 in the standard training data size setting on the BGL dataset. Also, with EG-Regularization, LogAnomaly achieves an even greater improvement of 47.82% in the small dataset training setting on HDFS. This finding confirms the effectiveness and general applicability of EG-Regularization in improving the robustness and transferability of anomaly detection models. Second, the percentage of improvement varies greatly, which is expected and contingent on the difficulty of datasets, anomaly detection models being used, and the specifics of training settings. For example, the F1-Score of LogAnomaly is improved by 0.2848 on HDFS with EG-Regularization, and some of its improvements are not significant. Third, trained with EG-Regularization, RT-Log consistently outperforms all baseline models by a large margin, where a 0.01 F1-Score improvement is considered significant. E.g., EG-Regularization significantly improves the detection F1-Score by 16.2% from 0.7266 to 0.8443 in the more difficult small training data size setting on BGL.

We note that besides distribution shift, many other factors can also affect the performance of detection models. While the EG-Regularization technique is mainly introduced to address the problem of distribution shift. As a consequence, there are noticeable variances of improvements for different models trained with EG-Regularization. Such results are consistent with our goal to introduce a general environment generalization technique for improving detection robustness and transferability.

Impact of the Training Data Size. We also evaluate the impact of varying the size of training data on the anomaly detection performance of different models. With such evaluations, we can firstly measure the robustness and transferability of different detection models, and secondly, find out the amount of training that is needed for obtaining plateaued performance for respective models. To this end, we vary the training data size from 10% to 90% for both the HDFS and BGL datasets, and train all models *without EG-Regularization* to focus on the robustness and transferability of models. The results are shown in Figure 6.

We can observe that RT-Log achieves the best anomaly detection performance under all the different training data sizes for both HDFS and BGL datasets, and its F1-Score steadily increases as we use more data for model training. Specifically, the F1-Score is increased by 4.72% from 0.9526 to 0.9976 for HDFS and by 35.69% from 0.7182 to 0.9745 respectively for BGL. By contrast, although

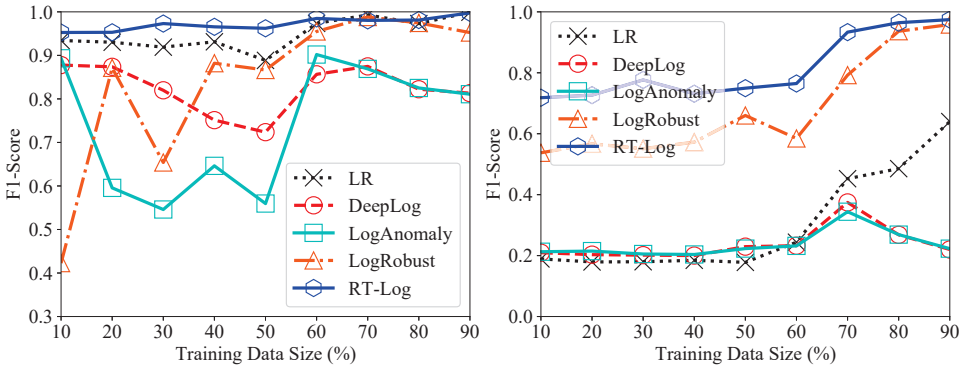


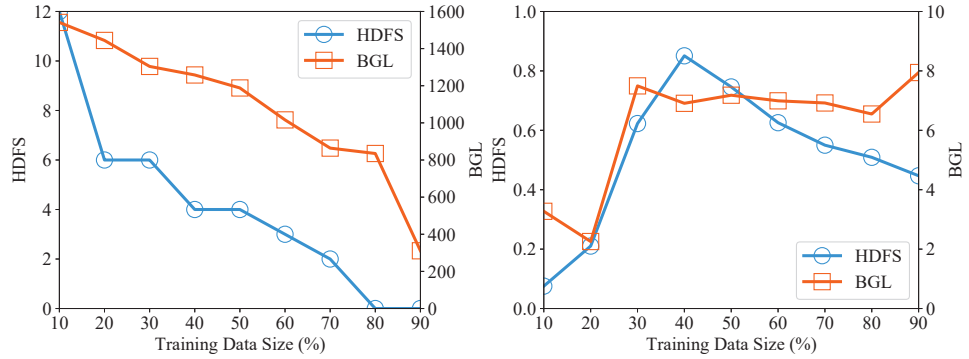
Fig. 6. The F1-Score of RT-Log and other anomaly detection models under different training data sizes.

the baseline models tend to improve their detection performance by training on more data, their performance is rather unstable. In particular, nearly all the baseline models suffer from certain decreases in the F1-Score even when trained using more data. E.g., we notice dramatic decreases and increases of F1-Scores for LogAnomaly using different ratios of training data sizes on HDFS.

As the training data size varies, the anomaly detection performance of the model is mainly affected by the following two factors: (1) the number and diversity of log sequences in the training data. Increasing the training data size typically leads to better anomaly detection performance. This is a general observation across models and datasets that when given enough training data, i.e., log sequences in log-based anomaly detection, the detection models can learn more effective and reliable patterns that are more likely to generalize to unseen data for better performance. Meanwhile, the learning model can also be exposed to more types of normal/anomalous log sequences, and thus can support the detection of a broader range of anomalies at runtime. (2) The degree of the distribution mismatch between the training and testing data. The model tends to perform worse under a greater degree of mismatch between the training and the testing data distribution where the trained model is deployed for detection.

To understand how these two factors affect detection performance, we show in Figure 7 the number of testing log events unseen during training and the KL divergence from training to testing data for the HDFS and BGL datasets. The KL divergence is computed using the event count vectors of each log sequence. From Figure 7(a), we can notice a clear decreasing trend of unseen log events for both datasets as more log sequences are used in training. From Figure 7(b), we find that the KL divergence of HDFS first increases to around 0.85 and then gradually decreases after a training data size of 40%, while the KL divergence of BGL first increases and remains relatively high for a training data size larger than 30%. Another important finding is that both the number of unseen log events and the KL divergence of the BGL dataset is one or several orders of magnitude larger than HDFS. This suggests that detecting anomalies on BGL is indeed more difficult than HDFS, which is in line with the worse prediction results on BGL.

As a result of these two factors, the F1-Scores of all the baseline models, namely LR, DeepLog, LogAnomaly and LogRobust, fluctuate drastically on both datasets with the increase of the training data size. The fluctuation is mainly because these models are more susceptible to the negative impact of the distribution mismatch. Specifically, their detection performance may still decline due to a larger KL divergence from training to testing distribution despite more data being used for



(a) Num. of unseen log events (b) KL Divergence from training to testing

Fig. 7. (a) the total number of testing log events unseen during training, and (b) KL divergence from training to testing distribution under different training data sizes of the HDFS and BGL datasets.

training. While the F1-Score of RT-Log is relatively stable and increases steadily as more training data are used, which suggests that RT-Log is intrinsically robust and transferable across different runtime distributions.

4.5 Efficiency and Interpretability

Parameter and Computational Efficiency. We empirically evaluate the parameter and computational efficiency of RT-Log compared with baseline detection models on the two large real-world datasets HDFS and BGL. We report the parameter size and the *training and inference throughput*, namely the number of log sequences processed per second. For a fair comparison, we adopt the same best settings as used for the main results of Table 2 with EG-Regularization (incurring negligible overhead by only 0.9~6.7%) and measure the efficiency either on a CPU or a GPU. Results are summarized in Table 4 and illustrated in Figure 8.

From Table 4 and Figure 8, we can observe that the simple baseline LR is extremely efficient in parameter and computation, which is expected as LR only has one weight vector of parameters and obtains its prediction by a simple dot product of this weight vector and the feature vector of the log sequence. For the more effective deep learning-based detection models, we notice that the parameter size of these models on the two datasets is quite comparable. As for the computational efficiency, the throughput of LogAnomaly is relatively lower, and the throughput of our RT-Log is comparable to the best baseline model LogRobust. Notably, all these detection models can achieve a detection throughput of up to hundreds to thousands per second, supporting real-time log-based anomaly detection in real-world applications. Another notable finding here is that all these detection models can be significantly accelerated by using a GPU for both training and inference, with an average speedup of 3.98x for training and 3.26x for inference. This suggests that the computational efficiency of RT-Log can be greatly boosted by using higher-end hardware such as GPUs.

Interpretability via Field Importance. As introduced in Section 3.1 and discussed in Section 3.5, after obtaining embedding vectors for the information fields of each structured log, we introduce N_o exponential neurons to adaptively capture the interaction weights among different fields to learn more effective and interpretable structured log representations. Based on the gated attention mechanism, we can aggregate the absolute values of the shared attention value vectors $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{N_o}]$, $\mathbf{v}_n \in \mathbb{R}^M$ in Equation 2 to show the general focus of all these exponential

Table 4. The parameter size and training/inference throughput of RT-Log and other anomaly detection models.

Model	HDFS					BGL				
	Param	Training		Inference		Param	Training		Inference	
		CPU	GPU	CPU	GPU		CPU	GPU	CPU	GPU
LR	98	11130	19692	12800	28444	202	19692	25600	15059	32000
DeepLog	54k	795	5333	1153	5953	57k	316	686	432	701
LogAnomaly	107k	256	2098	456	3241	114k	52	267	64	381
LogRobust	303k	883	2081	2327	4741	303k	347	892	698	1067
RT-Log	161k	458	2462	1515	4129	176k	210	898	489	1020

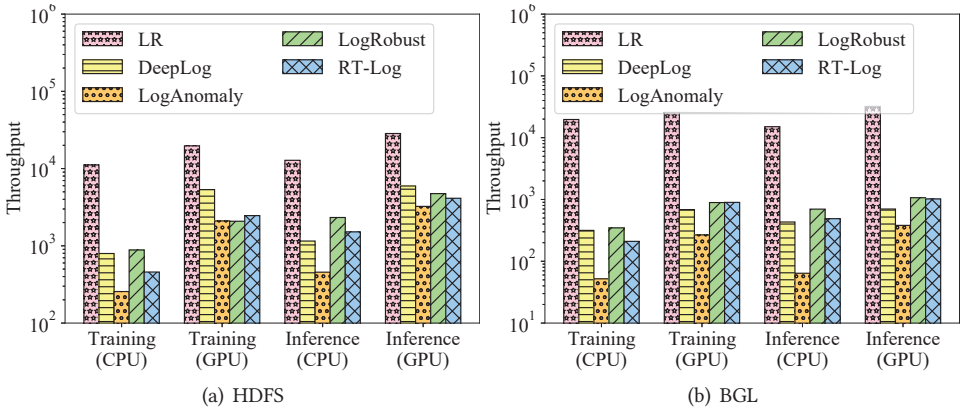


Fig. 8. Training and inference throughput of RT-Log and other anomaly detection models.

neurons on respective fields when detecting anomalies across log sequences. Meanwhile, we can also aggregate the interaction weights $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_{N_o}]$ in Equation 2 to obtain the local importance of respective fields when deciding whether the current log sequence contains anomalies or not.

The left panel of Figure 9 illustrates the global interaction weights of three representative exponential neurons and the average global interaction weights of all exponential neurons of RT-Log on HDFS, i.e., global field importance of the anomaly detection using RT-Log for all log sequences. Also, we visualize the local interaction weights of one representative anomalous log sequence, showing the interaction weights of three representative neurons and the average local interaction weights of all neurons of RT-Log. Here, the average values measure the importance of respective fields voted by all exponential neurons. Compared with baseline models that use only *log event template* for detection, RT-Log performs in-depth analysis on more information fields, namely *hour, minute, second, pid, level, component* for the HDFS dataset, to characterize each log message. Therefore, we mainly demonstrate interpretability using the importance of these fields. Although other mainstream explanation methods, e.g., LIME [53], SHAP [36], etc, can also be used to obtain the global and local importance of respective information fields, as reported in [4], these explanation methods are, however, agnostic of the model to be explained and are presented in terms of individual feature contributions. This can make it difficult to directly understand complex

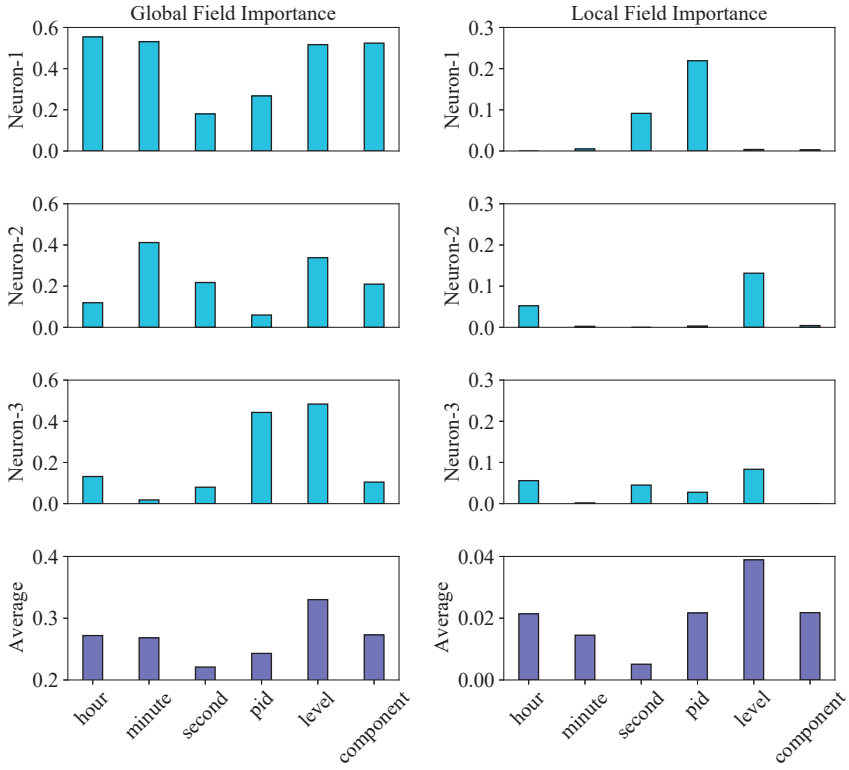


Fig. 9. Visualization of the field importance via exponential neurons for interpretable anomaly detection on HDFS.

relationships between features and model predictions. In contrast, our RT-Log supports built-in and thus more reliable interpretation based explicitly on feature interactions among fields.

From Figure 9, we can observe that the most important field identified by RT-Log is *level*. On the HDFS dataset, the field *level* has two categorical feature values, namely "INFO" and "WARNING", which faithfully records the significance of each log message. Meanwhile, the fields of *hour, minute* and *component* are of relatively high importance, which are identified by the three representative exponential neurons. The fields of *hour* and *minute* are closely related to periodical anomalies, emergencies, and etc., and some specific *components* of the distributed systems are more likely to suffer from systemic faults. These interpretation results further corroborate such empirical observations. Furthermore, the local field importance values reflect the importance of respective fields for detecting anomalies in the current log sequence. We can notice that different exponential neurons capture the interaction weights among the information fields in a sparse manner. For instance, Neuron-1 finds that the most important fields are *pid* and *second*, which indicates that Neuron-1 is responsible for these fields in the current log sequence. In other words, the high local importance assigned to "pid" and "second" by Neuron-1 shows that this exponential neuron finds out that certain processes are highly related to bugs , vulnerabilities, etc, and identifies such a periodic feature to be important in the current anomaly detection. Neuron-2 and Neuron-3 also identify their own key information fields for the current detection. Further, We notice that the

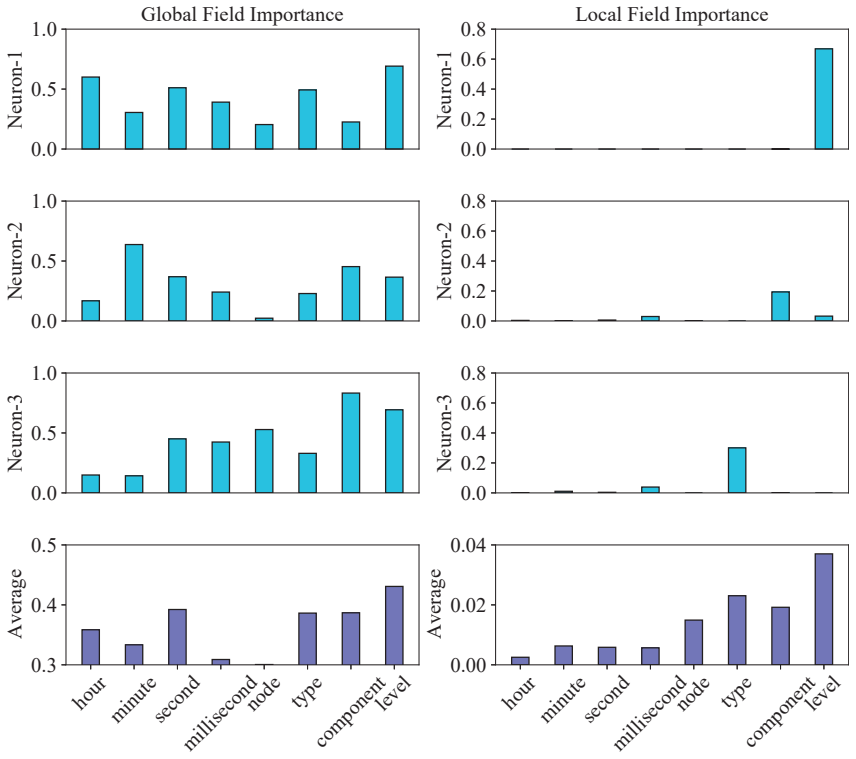
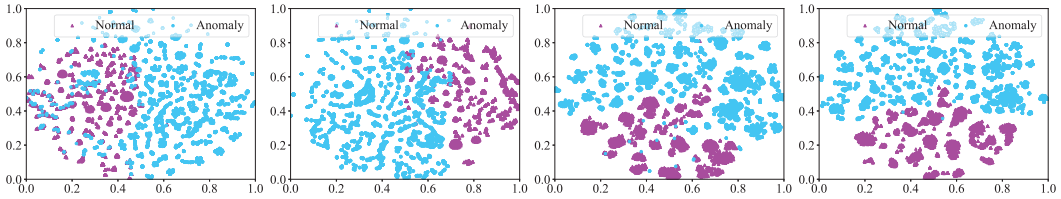


Fig. 10. Visualization of the field importance via exponential neurons for interpretable anomaly detection on BGL.

average interaction weights extracted by all the exponential neurons for the selected log sequence are in line with that of the whole dataset.

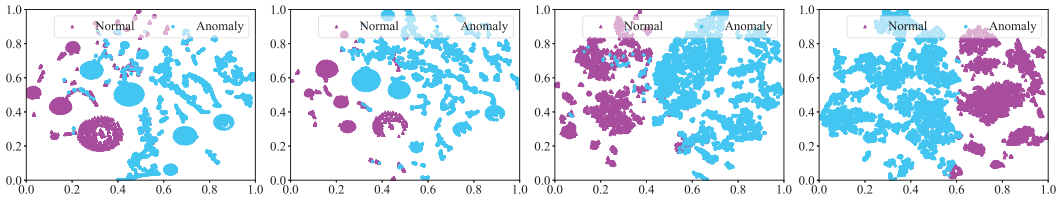
Similarly, Figure 10 shows the interaction weights of three representative exponential neurons and the average interaction weights of all the exponential neurons both globally and locally on the BGL dataset. We can notice that different exponential neurons process the information fields using different interaction weights. Other findings are generally consistent with the interpretability results on the HDFS dataset, e.g., the most important field identified by all the exponential neurons is *level* for both global and local interpretability. With such field importance interpretability, RT-Log can thus provide interpretable anomaly detection and analysis on a per log sequence basis.

Visualization of learned representations. We also visualize the representations learned with and without EG-Regularization for LogRobust and RT-Log on the HDFS and BGL datasets respectively. Specifically, as introduced in Section 2 and Section 3, each detection model extracts a representation vector for each log sequence and then feeds this vector into a final anomaly detection module to predict whether the log sequence contains anomalies. Therefore, we can visualize the learned log sequence representations via t-SNE [61] for a qualitative evaluation of the detection model and the proposed regularization technique. To this end, we randomly sample 5, 000 normal and anomalous log sequences for LogRobust and RT-Log, and then visualize their representations on the two datasets in Figure 11 and Figure 12 respectively.



(a) LogRobust w/o EG-R (b) LogRobust w/ EG-R (c) RT-Log w/o EG-R (d) RT-Log w/ EG-R

Fig. 11. Visualization of representations learned with and without EG-Regularization on the HDFS dataset using t-SNE.



(a) LogRobust w/o EG-R (b) LogRobust w/ EG-R (c) RT-Log w/o EG-R (d) RT-Log w/ EG-R

Fig. 12. Visualization of representations learned with and without EG-Regularization on the BGL dataset using t-SNE.

From Figures 11 and 12, we can notice that the representation vectors learned by RT-Log is qualitatively better than LogRobust on both datasets. Comparing the representations learned by RT-Log and LogRobust, we can notice that representation vectors learned by RT-Log cluster together, and there is a clear separation between the normal and anomalous log sequence representations for both datasets. This is desirable as with such representations, the subsequent anomaly detection module can easily derive a decision boundary that partitions the normal and anomalous log sequences into two separate sets in the representational space, and the anomaly detection will thus be more accurate. While for LogRobust, the representation vectors learned scatter around the representational space, and there are more overlaps between normal and anomalous ones. The better representations learned by RT-Log can be mainly ascribed to the comprehensive analysis conducted upon all key information fields for each log message via adaptive relation modeling. In addition, we can also find that the representations learned with EG-Regularization cluster closer together, and the boundaries of normal and anomalous log sequences are more clear for both LogRobust and RT-Log on the two datasets. This suggests that EG-Regularization indeed improves the robustness and transferability of the representations, which is consistent with the experimental results summarized in Table 3.

5 CONCLUSIONS

Log messages are an invaluable source for ensuring the safety and consistency of large systems. Recently, an increasing number of learning-based anomaly detection methods have been developed to automate the log-based anomaly detection process. However, existing methods analyze log messages using only part of the information available and fall short in handling the problem

of distribution shift. In this paper, we propose a robust and transferable framework RT-Log, to explicitly address these two problems. Specifically, we extract structured logs from raw log messages and introduce the adaptive relation modeling technique to conduct a more comprehensive analysis of each structured log. Further, we propose the EG-Regularization to facilitate learning environment invariant representations for improving the robustness and transferability. Extensive experimental results on real-world datasets confirm that our RT-Log significantly outperforms existing models and provides more interpretable anomaly detection results.

6 ACKNOWLEDGMENTS

This work is supported by the National Research Foundation, Singapore under its Emerging Areas Research Projects (EARP) Funding Initiative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore. Also, this work is supported by the National Key R&D Program of China (2021YFB1715600), National Natural Science Foundation of China (U22B2019, 62272372), Shenzhen Basic Research Grant (JCYJ20170816100819428).

REFERENCES

[1] Shivam Agarwal. 2013. Data Mining: Data Mining Concepts and Techniques. In *ICMIRA*. IEEE, 203–207.
[2] Martín Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. 2019. Invariant Risk Minimization. *CoRR* abs/1907.02893 (2019), 1–31.
[3] Alexei Baevski, Wei-Ning Hsu, Alexis Conneau, and Michael Auli. 2021. Unsupervised Speech Recognition. In *NeurIPS*. 1–14.
[4] Shaofeng Cai, Kaiping Zheng, Gang Chen, H. V. Jagadish, Beng Chin Ooi, and Meihui Zhang. 2021. ARM-Net: Adaptive Relation Modeling Network for Structured Data. In *SIGMOD*. ACM, 207–220.
[5] Lingjiao Chen, Arun Kumar, Jeffrey Naughton, and Jignesh M Patel. 2017. Towards Linear Algebra over Normalized Data. *PVLDB* 10, 11 (2017), 1214–1225.
[6] Zhuangbin Chen, Jinyang Liu, Wenwei Gu, Yuxin Su, and Michael R Lyu. 2021. Experience Report: Deep Learning-Based System Log Analysis for Anomaly Detection. *CoRR* abs/2107.05908 (2021), 1–10.
[7] Weiyu Cheng, Yanyan Shen, and Linpeng Huang. 2020. Adaptive Factorization Network: Learning Adaptive-Order Feature Interactions. In *AAAI*. AAAI Press, 3609–3616.
[8] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. In *SSST*. ACL, 103–111.
[9] Min Du and Feifei Li. 2016. Spell: Streaming Parsing of System Event Logs. In *ICDM*. IEEE, 859–864.
[10] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. Deeplog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In *CCS*. ACM, 1285–1298.
[11] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. 2009. Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis. In *ICDM*. IEEE, 149–158.
[12] Moises Goldszmidt, Mihai Budiu, Yue Zhang, and Michael Pechuk. 2010. Toward Automatic Policy Refinement in Repair Services for Large Distributed Systems. *SIGOPS* 44, 2 (2010), 47–51.
[13] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. 2019. A Survey of Methods for Explaining Black Box Models. *CSUR* 51, 5 (2019), 1–42.
[14] Ishaan Gulrajani and David Lopez-Paz. 2021. In Search of Lost Domain Generalization. In *ICLR*. OpenReview.net, 1–27.
[15] Hossein Hamooni, Biplob Debnath, Jianwu Xu, Hui Zhang, Guofei Jiang, and Abdullah Mueen. 2016. LogMine: Fast Pattern Recognition for Log Analytics. In *CIKM*. ACM, 1573–1582.
[16] Stephen E Hansen and E Todd Atkins. 1993. Automated System Monitoring and Notification with Swatch. In *LISA*. USENIX, 145–152.
[17] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. 2017. Drain: An online Log Parsing Approach with Fixed Depth Tree. In *ICWS*. IEEE, 33–40.
[18] Shilin He, Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, Michael R Lyu, and Dongmei Zhang. 2018. Identifying Impactful Service System Problems via Log Analysis. In *ESEC/FSE*. ACM, 60–70.
[19] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. 2016. Experience Report: System Log Analysis for Anomaly Detection. In *ISSRE*. IEEE, 207–218.
[20] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *NC* 9, 8 (1997), 1735–1780.

[21] Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional LSTM-CRF Models for Sequence Tagging. *CoRR* abs/1508.01991 (2015), 1–10.
[22] Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, Olivier J. Hénaff, Matthew M. Botvinick, Andrew Zisserman, Oriol Vinyals, and João Carreira. 2022. Perceiver IO: A General Architecture for Structured Inputs & Outputs. In *ICLR*. OpenReview.net, 1–24.
[23] Zhen Ming Jiang, Ahmed E Hassan, Parminder Flora, and Gilbert Hamann. 2008. Abstracting Execution Logs to Execution Events for Enterprise Applications. In *QSIC*. IEEE, 181–186.
[24] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. 2016. Fasttext.zip: Compressing Text Classification Models. *CoRR* abs/1612.03651 (2016), 1–13.
[25] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*. OpenReview.net, 1–15.
[26] Van-Hoang Le and Hongyu Zhang. 2021. Log-Based Anomaly Detection without Log Parsing. In *ASE*. IEEE, 492–504.
[27] Van Hoang Le and Hongyu Zhang. 2022. Log-Based Anomaly Detection with Deep Learning: How Far Are We?. In *ICSE*. IEEE, 1–12.
[28] Rui Li, Zhinan Cheng, Patrick PC Lee, Pinghui Wang, Yi Qiang, Lin Lan, Cheng He, Jinlong Lu, Mian Wang, and Xinquan Ding. 2021. Automated Intelligent Healing in Cloud-Scale Data Centers. In *SRDS*. IEEE, 244–253.
[29] Side Li, Lingjiao Chen, and Arun Kumar. 2019. Enabling and Optimizing Non-Linear Feature Interactions in Factorized Linear Algebra. In *SIGMOD*. ACM, 1571–1588.
[30] Xiaoyun Li, Pengfei Chen, Linxiao Jing, Zilong He, and Guangba Yu. 2020. Swisslog: Robust and Unified Deep Learning Based Log Anomaly Detection for Diverse Faults. In *ISSRE*. IEEE, 92–103.
[31] Yinglung Liang, Yanyong Zhang, Hui Xiong, and Ramendra Sahoo. 2007. Failure Prediction in IBM Bluegene/l Event Logs. In *ICDM*. IEEE, 583–588.
[32] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. 2016. Log Clustering Based Problem Identification for Online Service Systems. In *ICSE*. ACM, 102–111.
[33] Jian-Guang Lou, Qiang Fu, Shenqi Yang, Ye Xu, and Jiang Li. 2010. Mining Invariants from Console Logs for System Problem Detection. In *ATC*. USENIX Association, 1–14.
[34] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, João Gama, and Guangquan Zhang. 2018. Learning under Concept Drift: A Review. *TKDE* 31, 12 (2018), 2346–2363.
[35] Siyang Lu, Xiang Wei, Yandong Li, and Liqiang Wang. 2018. Detecting Anomaly in Big Data System Logs Using Convolutional Neural Network. In *DASC*. IEEE, 151–158.
[36] Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In *NIPS*. 4765–4774.
[37] Zhaojing Luo, Shaofeng Cai, Can Cui, Beng Chin Ooi, and Yang Yang. 2021. Adaptive Knowledge Driven Regularization for Deep Neural Networks. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI*. 8810–8818.
[38] Zhaojing Luo, Shaofeng Cai, Jinyang Gao, Meihui Zhang, Kee Yuan Ngiam, Gang Chen, and Wang-Chien Lee. 2018. Adaptive Lightweight Regularization Tool for Complex Analytics. In *34th IEEE International Conference on Data Engineering, ICDE*. 485–496.
[39] Adetokunbo AO Makanju, A Nur Zincir-Heywood, and Evangelos E Milios. 2009. Clustering Event Logs Using Iterative Partitioning. In *SIGKDD*. ACM, 1255–1264.
[40] André F. T. Martins and Ramón Fernández Astudillo. 2016. From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification. In *ICML*. PMLR, 1614–1623.
[41] Weibin Meng, Ying Liu, Federico Zaiter, Shenglin Zhang, Yihao Chen, Yuzhe Zhang, Yichen Zhu, En Wang, Ruizhi Zhang, Shimin Tao, et al. 2020. Logparse: Making Log Parsing Adaptive through Word Classification. In *ICCCN*. IEEE, 1–9.
[42] 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*. 4739–4745.
[43] Salma Messaoudi, Annibale Panichella, Domenico Bianculli, Lionel Briand, and Raimondas Sasnauskas. 2018. A Search-Based Approach for Accurate Identification of Log Message Formats. In *ICPC*. IEEE, 167–16710.
[44] Tim Miller. 2019. Explanation in Artificial Intelligence: Insights from the Social Sciences. *AIJ* 267 (2019), 1–38.
[45] Masayoshi Mizutani. 2013. Incremental Mining of System Log Format. In *SCC*. IEEE, 595–602.
[46] Meiyappan Nagappan and Mladen A Vouk. 2010. Abstracting Log Lines to Log Event Types for Mining Software System Logs. In *MSR*. IEEE, 114–117.
[47] Sasho Nedelkoski, Jasmin Bogatinovski, Alexander Acker, Jorge Cardoso, and Odej Kao. 2020. Self-Supervised Log Parsing. In *ECML-PKDD*. Springer, 122–138.
[48] Kim Anh Nguyen, Sabine Schulte im Walde, and Ngoc Thang Vu. 2016. Integrating Distributional Lexical Contrast into Word Embeddings for Antonym-Synonym Distinction. In *ACL*. ACL, 454–459.

[49] Adam Oliner, Archana Ganapathi, and Wei Xu. 2012. Advances and Challenges in Log Analysis. *CACM* 55, 2 (2012), 55–61.
[50] Adam Oliner and Jon Stearley. 2007. What Supercomputers Say: A Study of Five System Logs. In *DSN*. IEEE, 575–584.
[51] Ben Peters, Vlad Niculae, and André F. T. Martins. 2019. Sparse Sequence-to-Sequence Models. In *ACL*. ACL, 1504–1519.
[52] Charles R Qi, Or Litany, Kaiming He, and Leonidas J Guibas. 2019. Deep Hough Voting for 3d Object Detection in Point Clouds. In *ICCV*. IEEE, 9277–9286.
[53] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why should i trust you?" Explaining the predictions of any classifier. In *SIGKDD*. ACM, 1135–1144.
[54] Elan Rosenfeld, Pradeep Kumar Ravikumar, and Andrej Risteski. 2021. The Risks of Invariant Risk Minimization. In *ICLR*. OpenReview.net, 1–27.
[55] Gerard Salton and Christopher Buckley. 1988. Term-weighting Approaches in Automatic Text Retrieval. *IPM* 24, 5 (1988), 513–523.
[56] Keiichi Shima. 2016. Length Matters: Clustering System Log Messages Using Length of Words. *CoRR* abs/1611.03213 (2016), 1–10.
[57] Liang Tang, Tao Li, and Chang-Shing Perng. 2011. LogSig: Generating System Events from Raw Textual Logs. In *CIKM*. ACM, 785–794.
[58] Constantino Tsallis. 1988. Possible generalization of Boltzmann-Gibbs statistics. *JSP* 52, 1 (1988), 479–487.
[59] Risto Vaarandi. 2003. A Data Clustering Algorithm for Mining Patterns from Event Logs. In *IPOM*. IEEE, 119–126.
[60] Risto Vaarandi and Mauno Pihelgas. 2015. Logcluster-A Data Clustering and Pattern Mining Algorithm for Event Logs. In *CNSM*. IEEE, 1–7.
[61] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing Data Using T-SNE. *JMLR* 9, 11 (2008), 2579–2605.
[62] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All You Need. In *NeurIPS*. 5998–6008.
[63] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. 2009. Detecting Large-Scale System Problems by Mining Console Logs. In *SOSP*. ACM, 117–132.
[64] Lin Yang, Junjie Chen, Zan Wang, Weijing Wang, Jiajun Jiang, Xuyuan Dong, and Wenbin Zhang. 2021. PLELog: Semi-Supervised Log-Based Anomaly Detection via Probabilistic Label Estimation. In *ICSE*. IEEE, 230–231.
[65] Shenglin Zhang, Weibin Meng, Jiahao Bu, Sen Yang, Ying Liu, Dan Pei, Jun Xu, Yu Chen, Hui Dong, Xianping Qu, et al. 2017. Syslog Processing for Switch Failure Diagnosis and Prediction in Datacenter Networks. In *IWQoS*. IEEE, 1–10.
[66] Shuo Zhang, Junzhou Zhao, Pinghui Wang, Nuo Xu, Yang Yang, Yiting Liu, Yi Huang, and Junlan Feng. 2021. Learning to Check Contract Inconsistencies. In *AAAI*. AAAI Press, 14446–14453.
[67] 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 *ESEC/FSE*. IEEE, 807–817.
[68] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R Lyu. 2019. Tools and Benchmarks for Automated Log Parsing. In *ICSE-SEIP*. IEEE, 121–130.

Received July 2022; revised October 2022; accepted November 2022