LogDLR

Unsupervised Cross-System Log Anomaly Detection Through Domain-Invariant Latent Representation

Zhou, Junwei; Ying, Shaowen; Wang, Shulan; Zhao, Dongdong; Xiang, Jianwen; Liang, Kaitai; Liu, Peng

DOI
10.1109/TDSC.2025.3548050

Publication date
2025

Document Version
Final published version

Published in
IEEE Transactions on Dependable and Secure Computing

Citation (APA)
Zhou, J., Ying, S., Wang, S., Zhao, D., Xiang, J., Liang, K., & Liu, P. (2025). LogDLR: Unsupervised Cross-System Log Anomaly Detection Through Domain-Invariant Latent Representation. *IEEE Transactions on Dependable and Secure Computing*, *22*(4), 4456-4471. https://doi.org/10.1109/TDSC.2025.3548050

LogDLR: Unsupervised Cross-System Log Anomaly Detection Through Domain-Invariant Latent Representation

Junwei Zhou, *Member, IEEE*, Shaowen Ying, Shulan Wang, Dongdong Zhao, Jianwen Xiang, *Member, IEEE*, Kaitai Liang, *Member, IEEE*, and Peng Liu, *Member, IEEE*

***Abstract*—Log anomaly detection aims to discover abnormal events from massive log data to ensure the security and reliability of software systems. However, due to the heterogeneity of log formats and syntaxes across different systems, existing log anomaly detection methods often need to be designed and trained for specific systems, lacking generalization ability. To address this challenge, we propose LogDLR, a novel unsupervised cross-system log anomaly detection method. The core idea of LogDLR is to use universal sentence embeddings and a Transformer-based autoencoder to extract domain-invariant latent representations from log entries, which can effectively adapt to log format changes and capture semantic information and dependencies in log sequences. To obtain domain-invariant latent representations, we adopt a domain-adversarial training strategy, introducing a domain discriminator that competes with the Transformer-based encoder through a gradient reversal layer, forcing the encoder to learn shared knowledge between different system logs. Finally, the Transformer-based decoder detects anomalies based on the domain-invariant representations obtained by the encoder. We evaluate LogDLR in simulated cross-system scenarios using three publicly available log datasets. The experimental results show that LogDLR can handle heterogeneous logs effectively in cross-system scenarios and achieve efficient and accurate anomaly detection on both source and target systems.**

***Index Terms*—Anomaly detection, log analysis, adversarial training.**

I. Introduction

MODERN software systems are prone to various anomalies due to their increasing scale and complexity, which poses a serious threat to their stability and sustainability. Therefore, timely detection and diagnosis of anomalies is essential for ensuring the quality of software services. System logs, which record various events during software operation, are a valuable resource for effective anomaly detection, as they provide crucial insights into the system behavior [1]. However, contemporary software services often involve multiple types of software systems, each generating specific logs with unique structures and formats [2]. This leads to the challenges of scale and heterogeneity of logs for log-based anomaly detection.

System 1

```
# kernel: Dentry cache hash table entries: 1048576 (order: 11, 8388608 bytes)
# CPU: Trace cache: 12K uops, L1 D cache: 16K
# sshd[4677]: Remote host disconnected: Key exchange failed
# sshd[12523]: Public key authentication for user root accepted
# postfix/postdrop[10953]: warning: unable to look up public/pickup: No such file or directory
```

System 2

```
# kernel: Dentry cache hash table entries: 262144 (order: 10, 4096 KB)
# CPU: Trace cache: 12K uops<6>CPU: L2 cache: 1024K
# sshd[1122]: Remote host disconnected: Server host key changed
# sshd[15691]: Password authentication for user #155# accepted
# sshd2[11992]: WARNING: Could not chdir to home directory /home/#151#: No such file or directory
```

Fig. 1. An example of semantic similarity of log entries.

In recent years, deep learning-based log anomaly detection methods [3], [4], [5], [6], [7], [8] have attracted widespread attention due to their ability to automatically learn complex patterns and representations from large-scale log data. However, most existing methods are usually system-specific, meaning that they cannot be directly applied to a new system without substantial retraining or adaptation efforts [2]. To address this issue, some cross-system log anomaly detection methods based on transfer learning have been proposed [2], [9]. Cross-system log anomaly detection methods aim to detect anomalies in a new system (the target system or domain) by using the knowledge learned from another system (the source system or domain). This can reduce the cost and effort of developing and maintaining separate detection models for different systems. However, cross-system log anomaly detection methods face two main challenges:

1) Differences in log formats and syntaxes between systems can hinder the direct application of detection models across diverse software environments [2]. Cross-system log anomaly detection methods assume that logs from different systems have some semantic similarity, even though they may have different syntactic structures [2], [9]. As shown in Fig. 1, we can observe some semantic

Received 18 February 2024; revised 20 February 2025; accepted 20 February 2025. Date of publication 5 March 2025; date of current version 11 July 2025. This work was supported in part by the National Key Research and Development Program under Grant 2022YFB3104001, in part by the Key Research and Development Program of Hubei Province under Grant 2022BAA050, and in part by the NSF under Grant CNS2019340. *(Corresponding author: Jianwen Xiang.)*

Junwei Zhou, Shaowen Ying, Dongdong Zhao, and Jianwen Xiang are with the School of Computer and Artificial Intelligence, Wuhan University of Technology, Wuhan 430070, China (e-mail: junweizhou@msn.com; jwxiang@whut.edu.cn).

Shulan Wang is with the College of Big Data and Internet, Shenzhen Technology University, Shenzhen 518118, China.

Kaitai Liang is with the Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Delft 2600, The Netherlands.

Peng Liu is with the College of Information Sciences and Technology, Pennsylvania State University, University Park, PA 16801 USA.

Digital Object Identifier 10.1109/TDSC.2025.3548050

similarity between the logs generated on system 1 and system 2. For example, log entry "*# sshd[4677]: Remote host disconnected: Key exchange failed*" on system 1 and log entry "*# sshd[1122]: Remote host disconnected: Server host key changed*" on system 2 are semantically both indicating a failed SSH connection attempt. Therefore, the detection model needs to be generalizable enough to handle heterogeneous logs across systems.

2) The scarcity of sufficient anomaly labels in real-world scenarios. This challenge often arises from the inherent rarity of anomalous events in software systems, especially for newly deployed systems [9]. Supervised detection methods [2], which rely on labeled data to learn the underlying patterns that distinguish normal and anomalous logs, are severely limited by this lack of labeled data. Unsupervised detection methods, which do not require any labeled data, can overcome this limitation, but they may suffer from low accuracy or robustness [10].

To tackle the above two challenges, we propose LogDLR, an unsupervised cross-system log anomaly detection method that leverages domain-invariant latent representation. The key idea of LogDLR is to capture the essential features of log sequences that are invariant across different systems, regardless of their format and syntax differences. For example, in Fig. 1, both System 1 and System 2 have the similar indicative behavior, such as the remote host disconnection, the public key or password authentication, and the missing file or directory. These invariant features can be used to measure the semantic similarity of log sequences and detect anomalies. Therefore, LogDLR aims to map log sequences into a latent space, where the domain-invariant latent representation can be extracted and used for anomaly detection.

Specifically, LogDLR first applies a system-independent preprocessing step to the raw log data, followed by using a shared pre-trained sentence embedding model to extract semantic vectors from the preprocessed log entries. Next, LogDLR uses an autoencoder based on Transformer [11], which consists of an encoder and a decoder, to learn latent representations from the semantic vectors of log entries. To ensure the domain-invariance of the latent representations, LogDLR employs a gradient reversal layer [12] and an extra domain discriminator to engage in domain-adversarial training with the encoder. Finally, the decoder detects anomalies based on whether the latent representations can be correctly reconstructed.

The main contributions of this work are as follows:

1) We propose LogDLR, a novel unsupervised cross-system log anomaly detection method that can handle the variability and heterogeneity of log formats and syntaxes in cross-system scenarios.
2) We combine a Transformer-based autoencoder with domain-adversarial training to extract domain-invariant latent representations of log sequences, which capture the essential features of the system behavior while eliminating the format and syntax differences.
3) We evaluate LogDLR on three public log datasets from different systems, and the simulation results indicate that LogDLR can achieve high F1-scores on both source and target systems. We also open-source the implementation of LogDLR on Github[1] for reproducibility and further research.

II. Background and Motivation

A. Unsupervised Cross-System Anomaly Detection

Cross-system anomaly detection seeks to substantially reduce development and maintenance costs by identifying detection patterns in source systems and transferring them to target systems. This approach eliminates the need to develop separate detection models for each system.

The earliest cross-system method, LogTransfer [2], employs transfer learning to transfer abnormal patterns learned from a source system to a target system through a shared fully connected network. However, LogTransfer is supervised and requires anomaly-annotated data for both systems, which is often challenging to obtain in practical applications, especially in newly deployed or unlabeled system environments. To address this issue, LogTAD [9] introduces an unsupervised cross-system anomaly detection framework. This framework uses a Long Short-Term Memory (LSTM) [13] generator to cluster normal log data from different systems near the center of a hypersphere and applies domain-adversarial adaptation to align the distributions of log data from different systems, detecting anomalies that deviate significantly from the center.

However, LogTAD [9] employs the LSTM model, which has inherent limitations in capturing long-distance dependencies and complex semantic features. For log data with diverse and long sequence feature distributions, LSTM can only focus on previous information at each time step due to its cyclic structure, and therefore cannot capture the global information of the sequence [6]. Although LogTAD utilizes domain-adversarial adaptation to align the distributions of the source and target systems, its generator's ability is constrained by the LSTM model, preventing it from fully capturing fine-grained feature differences between systems.

To address these issues, we utilize a Transformer-based autoencoder to learn the latent representations of log sequences. Through the self-attention mechanism, the Transformer can simultaneously focus on all positions in a sequence, efficiently capturing long-distance dependencies and complex contextual associations within the log sequence [11]. It can extract temporal patterns and deeply explore the semantic features of the logs. Compared to LogTAD, the Transformer-based autoencoder more comprehensively reflects the internal structure and behavioral characteristics of logs, enhancing adaptability to heterogeneous log data across systems.

B. Log Semantic Feature Extraction

System logs are event records generated during software execution, stored in text form, and contain rich semantic information. However, logs from different systems may vary significantly in format, structure, and syntax, complicating the direct

[1]https://github.com/yingsw79/LogDLR

extraction and comparison of log semantics in cross-system scenarios [2].

Traditional methods, such as LogTransfer [2] and LogTAD [9], use word embedding techniques like GloVe [14] and Word2Vec [15] for semantic feature extraction. However, the embeddings generated by GloVe and Word2Vec are static and cannot dynamically adjust word meanings based on context, underperforming when processing semantically complex or ambiguous log entries [7]. To address this issue, several works, such as NeuralLog [6] and SwissLog [8], use pre-trained Bidirectional Encoder Representations from Transformers (BERT) [16] word embeddings to significantly improve the quality of log semantic representations. BERT's primary design goal is to generate contextual word embeddings, dynamically adjusting each input word's vector to capture local contextual relationships [16]. However, for sentence embeddings, BERT typically generates representations by pooling word vectors (e.g., average pooling or using [CLS] token vectors) [6]. This approach is not specifically optimized for sentence-level semantic relationships, especially in cross-system semantic comparisons of log entries [17].

In LogDLR, we adopt the BERT-based Sentence-BERT (SBERT) [17] sentence embedding model for tasks requiring sentence-level semantic representation in cross-system log anomaly detection. SBERT is designed to generate high-quality sentence embeddings for computing semantic similarity between sentences. It introduces a sentence-pair architecture based on BERT and is fine-tuned through contrastive learning or sentence-pair tasks (such as natural language inference) to make the generated sentence embeddings more suitable for semantic comparison and measurement [17]. By leveraging SBERT's pre-trained sentence embedding model, we treat log entries as natural language text and efficiently adapt to the diversity of log formats and syntax.

C. Domain Shift

In cross-domain tasks, differences in data distribution between the source domain and the target domain (i.e., domain shift) often hinder models trained on the source domain from performing well on the target domain [18]. This issue is particularly prominent when models trained on source system logs cannot generalize to other target systems due to changes in data distribution [9].

To address the domain shift problem, domain adaptation methods enable models to share feature representations across different domains by reducing data distribution differences between the source and target domains, thereby improving generalization ability [19], [20], [21]. Domain-adversarial neural networks (DANN) [19] are fundamental domain adaptation methods that aim to learn domain-invariant features through adversarial training. The core idea is to introduce a domain discriminator and a gradient reversal layer (GRL) [12] to make the features generated by the model indistinguishable between the source and target domains through adversarial training. This mechanism encourages the model to focus on extracting task-relevant, domain-invariant features while ignoring domain-specific differences.

In this work, we extend the principles of DANN [19] to autoencoder-based anomaly detection tasks to achieve universal anomaly detection across system log data. Specifically, LogDLR integrates domain-adversarial training with the reconstruction mechanism of autoencoders. Domain-adversarial training guides the encoder through a domain discriminator to generate domain-invariant latent representations, significantly reducing distribution differences between the source and target systems. The autoencoder's reconstruction loss ensures that the latent representation effectively captures the structural and semantic patterns of normal logs, thereby maintaining the ability to distinguish between normal and abnormal logs. This synergy enables LogDLR to overcome the differences in log distribution and achieve cross-system anomaly detection.

III. Threat Model and Assumptions

This work focuses on detecting anomalies in cross-system log scenarios, where logs from different systems may differ in structure, format, and semantics. The proposed LogDLR method is designed for unsupervised anomaly detection and is not specific to any particular type of threat or attack. A log entry is considered an anomaly if its pattern deviates significantly from the typical pattern learned during training on normal log sequences. The anomaly detection process targets point anomalies, where individual log entries or sequences are evaluated based on their domain-invariant representation and reconstruction error.

We assume that logs contain meaningful textual descriptions of system events. LogDLR leverages a pre-trained sentence embedding model to extract semantic features from logs. However, if the log contains unreadable or uninformative entries (such as hexadecimal memory addresses or raw binary data), the semantic extraction process may not capture their potential meaning. In such cases, LogDLR treats these entries as part of the normal training data, assuming that their patterns do not significantly affect the model's ability to detect anomalies. Additionally, we assume that logs from different systems exhibit a certain degree of semantic similarity despite differences in format and syntax.

IV. Design of LogDLR

A. Overview

We consider a source system s and a target system t, each of which generates a log sequence $B = \{b_i\}_{i=1}^{w}$, where b_i is the i-th log entry and w is the length of the log sequence. Given a mixed training dataset $\mathcal{D} = \{B_i\}_{i=1}^{n=n_s+n_t}$, which contains n_s normal log sequences $\mathcal{D}_s = \{B_i^s\}_{i=1}^{n_s}$ from the source system and n_t ($< n_s$) normal log sequences $\mathcal{D}_t = \{B_i^t\}_{i=1}^{n_t}$ from the target system, we expect LogDLR to extract domain-invariant representations and achieve unified anomaly detection regardless of log formats and syntaxes.

Fig. 2 illustrates the framework of LogDLR. The framework takes log sequences from both the source and target systems as input. It preprocesses the log sequences with a system-independent method and then transforms them into semantic vectors using a universal sentence embedding. The Transformer-based autoencoder consists of a Transformer-based encoder and

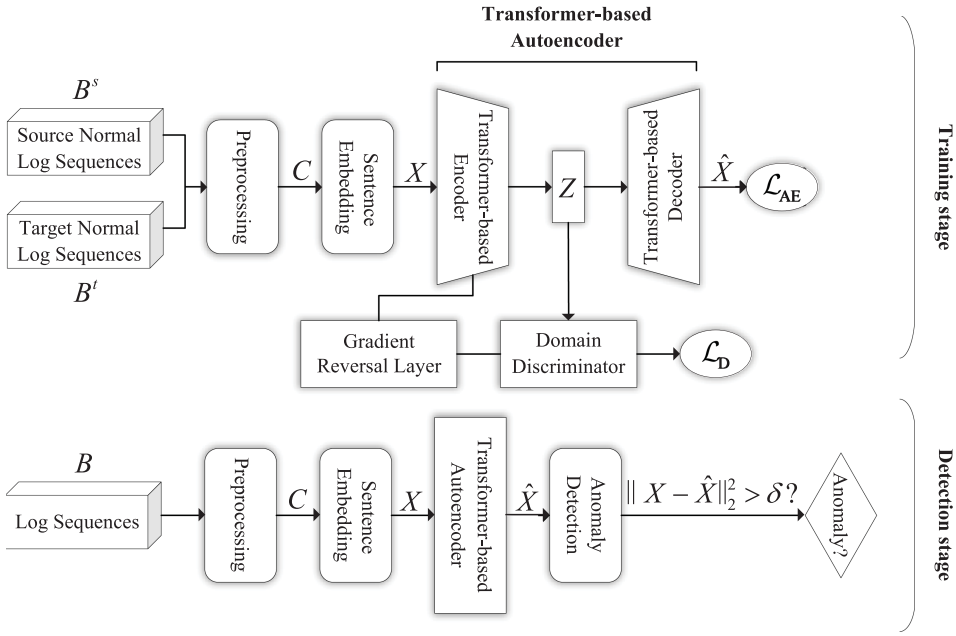


Fig. 2. The overall framework of LogDLR.

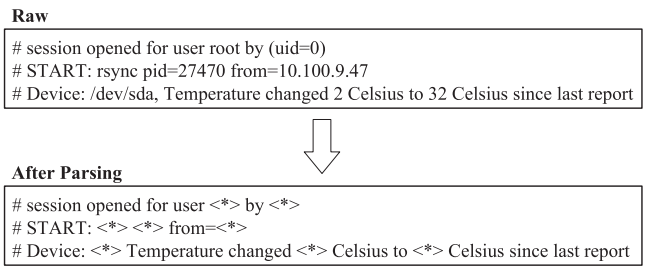


Fig. 3. An example of log parsing (for illustration only, not used in LogDLR).

a Transformer-based decoder. In the training stage, the encoder learns the latent representations of the normal log sequences from their semantic vectors. The domain discriminator, connected to the encoder through a gradient reversal layer, enables domain-adversarial training to obtain domain-invariant representations. The Transformer-based decoder reconstructs the semantic vectors of the log sequences from the representations. In the detection stage, the decoder detects anomalies based on the reconstruction errors of the log sequences exceeding a certain threshold.

B. Preprocessing

Existing log anomaly detection methods typically rely on log parsers (such as Drain [22], Spell [23], etc.) to parse the raw logs into event templates during the preprocessing step [2], [9]. Fig. 3 shows an example of the log parsing. However, log parsing errors can compromise the integrity of log entries, leading to semantic loss or misunderstanding [6]. Moreover, the parsing process is system-specific, and usually requires writing suitable regular expressions for the log structure of a specific system, to ensure the quality of templates [24], [25], [26]. This processing step is time-consuming and laborious, and different systems generally have different templates due to the diversity of log formats and syntaxes [27]. Therefore, parsing is not suitable for cross-system anomaly detection scenarios.

Instead of parsing the raw logs with complex methods, we use simple text preprocessing methods from the natural language processing (NLP) field. We first extract the content field of log entries, then convert it to lowercase and remove non-alphabetic characters. Table I illustrates examples of preprocessing steps applied to three different datasets. Our processing step is simple and does not involve manual addition of system-specific regular expressions. It can be easily implemented with standard NLP toolkits. We denote the preprocessed log sequence as $C = \{c_i\}_{i=1}^{w}$, where $c_i = \text{preprocessing}(b_i)$ is the i-th preprocessed log entry.

C. Sentence Embedding

The sentence embedding model we use is based on Sentence-BERT (SBERT)[2] [17], a variation of BERT [16] specifically designed for generating high-quality sentence embeddings. Unlike standard BERT, which primarily focuses on token-level representations, SBERT adopts a siamese network architecture. This architecture consists of two identical BERT models that share parameters and are trained on sentence pairs [17]. Each sentence is processed independently by one of the BERT models to produce an embedding, and the similarity between the two embeddings is then measured.

The goal of SBERT is to generate sentence embeddings that reflect semantic similarity by fine-tuning BERT on labeled sentence pairs. This ensures that embeddings of semantically similar sentences are closer in the vector space, while embeddings of dissimilar sentences are farther apart [17]. This training

[2]https://www.sbert.net

TABLE I
EXAMPLES OF THE PREPROCESSING STEP ON THREE DIFFERENT DATASETS: THUNDERBIRD, SPIRIT, AND BGL

Dataset	Raw Log	Preprocessed Log
Thunderbird	- 1131566461 2005.11.09 eadmin2 Nov 9 12:01:01 src@eadmin2 crond(pam_unix)[12636]: session opened for user root by (uid=0)	session opened for user root by uid
Spirit	- 1104566437 2005.01.01 sadmin1 Jan 1 00:00:37 sadmin1/sadmin1 dhcpd: DHCPDISCOVER from 00:11:85:6b:10:a6 via eth1: network 172.31.0/16: no free leases	dhcpdiscover from via eth network no free leases
BGL	APPSEV 1134630981 2005.12.14 R37-M1-N8-I:J18-U11 2005-12-14-23.16.21.740107 R37-M1-N8-I:J18-U11 RAS APP FATAL ciod: Error reading message prefix on CioStream socket to 172.16.96.116:49934, Link has been severed	ciod error reading message prefix on ciostream socket to link has been severed

strategy enables the model to effectively capture sentence-level semantics, making it particularly well-suited for tasks involving semantic similarity and comparison. By leveraging this capability, SBERT generates robust sentence embeddings that can be directly used to encode log entries, preserving their contextual and semantic information.

In our framework, we utilize the SBERT model E to transform preprocessed log entries into d_e-dimensional embedding vectors. Formally, given a log sequence with w entries, the embedded log sequence is represented as $X = \{x_i\}_{i=1}^{w}$, where $x_i = E(c_i)$ denotes the embedding vector of the i-th log entry.

D. Transformer-Based Autoencoder

The Transformer-based autoencoder consists of a Transformer-based encoder and a Transformer-based decoder, as shown in Fig. 2. Given an embedded log sequence $X_i \in \mathbb{R}^{w \times d_e}$ in $\mathcal{D}$, the encoder compresses it into a latent representation, and the decoder reconstructs the log sequence from this latent representation.

The Transformer-based encoder is composed of l identical Transformer layers, each with two sub-layers. The first sub-layer is multi-head self-attention, and the second sub-layer is a position-wise feed-forward network (FFN) [11]. The core component of the Transformer layer is the multi-head attention mechanism, which consists of h parallel self-attention heads that allow the model to weigh the importance of different parts of the input sequence. It is expressed as:

$$\begin{aligned} H_i &= \text{MultiHeadAttn}(X_i) \\ &= \text{Concat}(head_1, \ldots, head_j, \ldots, head_h)\, \theta_O, \end{aligned} \tag{1}$$

where

$$head_j = \text{Attention}\left(X_i\theta_{Q_j}, X_i\theta_{K_j}, X_i\theta_{V_j}\right), \tag{2}$$

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V; \tag{3}$$

θ_{Q_j}, θ_{K_j}, $\theta_{V_j} \in \mathbb{R}^{d_e \times d_k}$ and $\theta_O \in \mathbb{R}^{hd_k \times d_e}$ are learnable parameter matrices; d_k is the dimension of a self-attention head. Additionally, each sub-layer in the Transformer layer applies residual connections and layer normalization [11]. Therefore, the output of the self-attention layer is defined as:

$$\tilde{H}_i = \text{LayerNorm}(X_i + H_i). \tag{4}$$

The FFN is applied after the self-attention sub-layer. It consists of two linear transformations with a ReLU activation function in between:

$$\text{FFN}(\tilde{H}_i) = \text{ReLU}\left(\tilde{H}_i\theta_1 + b_1\right)\theta_2 + b_2, \tag{5}$$

where $\theta_1 \in \mathbb{R}^{d_e \times d_{ff}}$, $\theta_2 \in \mathbb{R}^{d_{ff} \times d_e}$, $b_1 \in \mathbb{R}^{d_{ff}}$, and $b_2 \in \mathbb{R}^{d_e}$ are learnable parameter matrices and biases; d_{ff} is the dimension of FFN. After the FFN sub-layer, another residual connection and layer normalization are applied. Therefore, the output of the Transformer layer is defined as the result of the second layer normalization:

$$\text{TransformerLayer}(X_i) = \text{LayerNorm}\left(\tilde{H}_i + \text{FFN}\left(\tilde{H}_i\right)\right). \tag{6}$$

Finally, we use a linear layer to reduce the output of the Transformer layer to a d_z-dimensional vector Z_i, which is the latent representation of the log sequence X_i:

$$\begin{aligned} Z_i &= \text{Encoder}(X_i) \\ &= \text{TransformerLayer}(X_i)\theta_3 + b_3, \end{aligned} \tag{7}$$

where $\theta_3 \in \mathbb{R}^{wd_e \times d_z}$ and $b_3 \in \mathbb{R}^{d_z}$ are learnable parameters of the linear layer.

The Transformer-based decoder has a network structure completely symmetrical to the encoder. It decodes the latent representation Z_i into a log sequence $\hat{X}_i \in \mathbb{R}^{w \times d_e}$. To make it easy, we denote the operation as:

$$\hat{X}_i = \text{Decoder}(Z_i). \tag{8}$$

To ensure that the autoencoder can learn latent representations of normal log sequences while reconstructing them correctly. We use mean squared error as the measure of reconstruction error, that is:

$$\text{RE}(X_i) = \|X_i - \hat{X}_i\|_2^2, \tag{9}$$

where $\|\cdot\|_2$ denotes the L_2 norm and $\hat{X}_i$ is the reconstruction of X_i by the autoencoder model. The reconstruction error is also used as an anomaly score to detect abnormal log sequences, as we will explain in Section IV-G. The corresponding objective function is defined as:

$$\mathcal{L}_{\text{RE}} = \frac{1}{n}\sum_{i=1}^{n}\text{RE}(X_i). \tag{10}$$

By minimizing (10), normal log sequences will have lower reconstruction errors because their latent representations are correctly learned by the encoder, while abnormal log sequences will have higher reconstruction errors.

In addition, we apply L_2 regularization to the latent representation Z [28], as follows:

$$\mathcal{L}_{\text{EM}} = \frac{1}{n}\sum_{i=1}^{n}\|Z_i\|_2^2, \tag{11}$$

where Z_i is the latent representation of X_i. This regularization term encourages the latent representations to have smaller magnitudes, preventing them from becoming too large and potentially overfitting the data.

Combining the two parts above, we define the objective function of the autoencoder part as:

$$\mathcal{L}_{\text{AE}} = \mathcal{L}_{\text{RE}} + \alpha\mathcal{L}_{\text{EM}}, \tag{12}$$

where α is a hyperparameter controlling the weight of $\mathcal{L}_{\text{EM}}$.

E. Domain-Adversarial Training

In this section, we describe the domain-adversarial training method that we use to learn domain-invariant representations (Z) of the system's normal log sequence.

We introduce an additional domain discriminator D and connect it to the Transformer-based encoder through a gradient reversal layer (GRL) [12]. The discriminator models the probability that the encoder generates the latent representation from the source system or the target system. The discriminator consists of two linear layers with a LeakyReLu [29] function. Formally, we can calculate the probability distribution p_i that the latent representation Z_i comes from $\mathcal{D}_s$ or $\mathcal{D}_t$ by applying a softmax function to the output of the discriminator:

$$p_i = \text{Softmax}\left(\text{LeakyReLU}\left(\text{GRL}(Z_i)\theta_4 + b_4\right)\theta_5 + b_5\right), \tag{13}$$

where θ_4, θ_5, b_4, and b_5 are learnable parameters of the discriminator.

During forward propagation, the GRL acts as an identity function, allowing the representations to flow through unchanged. However, during backward propagation, the GRL modifies the gradients that flow back from the discriminator to the encoder. It multiplies the gradients by a negative constant λ, effectively changing their direction. Fig. 4 shows the role of GRL in forward and back propagation. Consequently, the encoder and the discriminator optimize in opposite directions: the encoder aims to generate representations that obscure their domain source, while the discriminator seeks to accurately identify the domain source.

With the inclusion of the discriminator, the encoder now has two objectives. Firstly, it learns a latent representation of the system's normal log sequence. Secondly, it ensures that the representations for different systems are domain-invariant and indistinguishable by the discriminator. The objective of the Transformer-based decoder remains the same, which is to reconstruct the log sequence from the domain-invariant representation provided by the encoder. This adversarial training process encourages the encoder to learn representations that are insensitive to the domain shift between the source and target systems. It effectively aligns the feature distributions of different domains, producing domain-invariant representations that can be shared and transferred across systems.

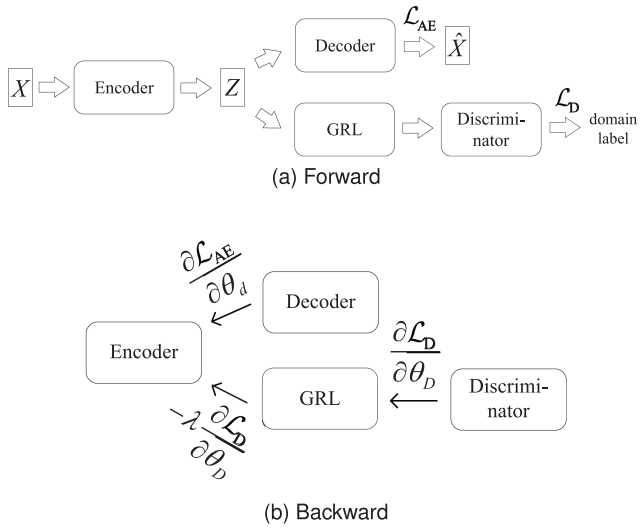


Fig. 4. The role of GRL in forward and back propagation.

The theoretical foundation of domain-invariant latent representations is supported by minimizing the $\mathcal{H}$-divergence [12], [30] between $p_S(Z)$ and $p_T(Z)$, which represent the latent feature distributions of the source and target domains, respectively. As noted by Ganin et al. [12], $\mathcal{H}$-divergence $d_{\mathcal{H}}(p_S(Z), p_T(Z))$ can be estimated as follows:

$$\begin{aligned} &d_{\mathcal{H}}\left(p_S(Z), p_T(Z)\right) \\ &= 2\sup_{\mathcal{H}}\left|\mathbb{P}_{Z\sim p_S(Z)}\left[h(Z)=1\right] - \mathbb{P}_{Z\sim p_T(Z)}\left[h(Z)=1\right]\right| \\ &= 2\sup_{\mathcal{H}}\left[\mathbb{P}_{Z\sim p_S(Z)}[h(Z)=0] + \mathbb{P}_{Z\sim p_T(Z)}\left[h(Z)=1\right] - 1\right] \\ &= 2\sup_{\mathcal{H}}\left[f(h) - 1\right], \end{aligned} \tag{14}$$

where $\mathcal{H}$ is the hypothesis class of functions $h : \mathcal{Z} \to \{0, 1\}$ and $f(h) = \mathbb{P}_{Z\sim p_S(Z)}[h(Z) = 0] + \mathbb{P}_{Z\sim p_T(Z)}[h(Z) = 1]$. Thus, $f(D)$ measures the discriminator's effectiveness in identifying the domain source of the latent representations.

We adopt cross-entropy loss as the objective function for domain prediction of latent representation, which is defined as:

$$\mathcal{L}_{\text{D}} = -\frac{1}{n}\sum_{i=1}^{n}\left[q_i\log(p_i) + (1-q_i)\log\left(1-p_i\right)\right], \tag{15}$$

where p_i is calculated by (13) and q_i is the domain label of the i-th log sequence X_i, indicating whether it comes from $\mathcal{D}_s$ or $\mathcal{D}_t$ (if $X_i \in \mathcal{D}_s$, then $q_i = 0$, if $X_i \in \mathcal{D}_t$, then $q_i = 1$). The discriminator is trained to minimize 15, thereby maximizing $f(D)$ and enhancing the classification accuracy of the latent representation. GRL inverts these gradients for the encoder, compelling it to reduce $f(D)$ by learning representations that confuse the discriminator, thus effectively reducing $d_{\mathcal{H}}(p_S(Z), p_T(Z))$.

At this point, we define the mathematical concept of domain-invariant latent representation as follows: A latent representation $Z \in \mathbb{R}^{d_z}$ is learned by the encoder (Encoder : $\mathbb{R}^{w \times d_e} \rightarrow \mathbb{R}^{d_z}$), satisfying two properties:

- *Task Relevance:* Z preserves the structural and semantic patterns of normal log sequences necessary for reconstruction and anomaly detection by minimizing the reconstruction error ($\mathcal{L}_{\text{AE}}$).
- *Domain Invariance:* The distributions of Z for the source system $\mathcal{D}_s$ and target system $\mathcal{D}_t$, denoted as $p_S(Z)$ and $p_T(Z)$, respectively, are aligned such that the $\mathcal{H}$-divergence between them ($d_{\mathcal{H}}(p_S(Z), p_T(Z))$) is minimized.

F. Overall Objective Function

During the training phase, we aim to minimize the objective function $\mathcal{L}_{\text{AE}}$ on the training set $\mathcal{D}$, thereby optimizing the encoder parameters θ_e and decoder parameters θ_d. This ensures that the autoencoder can learn the latent representation and reconstruction of normal log sequences in both $\mathcal{D}_s$ and $\mathcal{D}_t$.

To obtain domain-invariant representations, we optimize the domain discriminator parameters θ_D by minimizing the cross-entropy loss ($\mathcal{L}_{\text{D}}$), which is equivalent to maximizing the negative cross-entropy loss ($-\mathcal{L}_{\text{D}}$). Similarly, we optimize the encoder parameters θ_e by maximizing the same cross-entropy loss as the discriminator, which is equivalent to minimizing the negative cross-entropy loss. Therefore, the overall objective function can be defined as follows:

$$\mathcal{L} = \min_{\theta_e, \theta_d} \max_{\theta_D} \left[\mathcal{L}_{\text{AE}}(\theta_e, \theta_d) - \beta \mathcal{L}_{\text{D}}(\theta_e, \theta_D) \right], \tag{16}$$

where θ_e, θ_d and θ_D are the parameters of the encoder, the decoder and the domain discriminator, respectively. β is a hyperparameter that controls the weight of $\mathcal{L}_D$.

G. Anomaly Detection

Once the Transformer-based autoencoder is trained, it can be used for anomaly detection. The basic assumption is that normal log sequences can be effectively compressed into low-dimensional latent representations and reconstructed with minimal error. In contrast, anomalous log sequences, which deviate significantly from the learned normal patterns, result in larger reconstruction errors [31]. Therefore, we classify the log sequence X_i as anomalous if its reconstruction error exceeds a certain threshold δ; otherwise, we consider it normal. The formula is:

$$\text{Pred}(X_i) = \begin{cases} \text{Normal}, & \text{RE}(X_i) \leq \delta; \\ \text{Anomaly}, & \text{RE}(X_i) > \delta. \end{cases} \tag{17}$$

where $\text{RE}(X_i)$ is calculated by 9 and the threshold δ is obtained by Algorithm 1 on an additional validation set to ensure the optimal F1-score. When there are known abnormal samples in the validation set, the threshold can be selected by maximizing the F1-score.

Algorithm 1: Threshold Selection for Anomaly Detection.

Input: $recon\_losses$: List of reconstruction losses for log entries, $labels$: Ground truth labels (0 for normal, 1 for anomaly), n: Number of threshold candidates to evaluate
Output: δ: Optimal threshold

Compute $mean_N \leftarrow \text{Mean}(\{loss \mid loss \in recon\_losses \wedge label = 0\})$
Compute $mean_A \leftarrow \text{Mean}(\{loss \mid loss \in recon\_losses \wedge label = 1\})$
Initialize $step\_size \leftarrow (mean_A - mean_N)/n$
Initialize $max\_f1 \leftarrow -1$, $\delta \leftarrow mean_N$
for $i \leftarrow 0$ to $n-1$ **do**
  $current\_threshold \leftarrow mean_N + i \times step\_size$
  $predictions \leftarrow \{0 \text{ if } loss \leq current\_threshold \text{ else } 1 \mid loss \in recon\_losses\}$
  Compute $f1 \leftarrow \text{F1-score}(predictions, labels)$
  if $f1 > max\_f1$ **then**
    $max\_f1 \leftarrow f1$
    $\delta \leftarrow current\_threshold$
  end if
end for
return δ

TABLE II
STATISTICS OF THE DATASETS

Datasets	# Size	# Log entries	# Anomalies
Thunderbird	1.6G	10,000,000	353,794
Spirit	1G	7,983,345	768,142
BGL	709M	4,747,963	348,460

V. Evaluation

A. Experimental Setup

1) Datasets: We evaluate our approach on three public log datasets, Thunderbird, Spirit and BGL. Table II shows the statistics of the datasets.

- *Thunderbird:* This dataset is collected from the Thunderbird supercomputer system at Sandia National Labs (SNL) [32]. Thunderbird is a large dataset containing over 200 million log entries and nearly 30 GB in size [10]. Consistent with prior work [6], [9], we use the first 10,000,000 log entries from the original Thunderbird dataset to form our dataset. Among these entries, 353,794 are labeled as anomalies.
- *Spirit:* The Spirit dataset is also collected from a supercomputer at SNL and contains more than 272 million log entries [33]. For Spirit, we use the 1 GB of logs from the Spirit dataset same as prior work [6], which contains 7,983,345 log entries with 768,142 anomalies.
- *BGL:* It comprises 4,747,963 logs collected from the BlueGene/L supercomputer system at Lawrence Livermore National Labs (LLNL) [32], of which 348,460 log entries are anomalous.

For the above datasets, we use a sliding window of size 20 to slice the logs into log sequences ($w = 20$), following [9]. Then,

TABLE III
DATA PARTITIONING DETAILS WHEN DATASET SERVES AS THE SOURCE SYSTEM

		Thunderbird	Spirit	BGL
Training	# Sequences	100,000	100,000	100,000
Validation	# Sequences # Anomalies	10,000 2,574	10,000 2,177	10,000 948
Test	# Sequences # Anomalies	2,375,185 611,443	1,869,679 407,116	1,058,369 100,338

TABLE IV
DATA PARTITIONING DETAILS WHEN DATASET SERVES AS THE TARGET SYSTEM

		Thunderbird	Spirit	BGL
Training	# Sequences	10,000	10,000	10,000
Validation	# Sequences # Anomalies	1,000 247	1,000 207	1,000 87
Test	# Sequences # Anomalies	2,483,185 613,770	1,977,679 409,086	1,166,369 101,199

we split the obtained log sequences into three non-overlapping parts: training set, validation set (used to determine the reconstruction error threshold δ), and test set.

To emulate the real-world imbalance between source and target system log samples, we ensure a larger number of normal log sequences for training on the source system than on the target system. Specifically, when using the aforementioned datasets as source or target systems, we randomly select 100,000 or 10,000 normal log sequences respectively for the training set. Table III and Table IV provide detailed partitioning information. When evaluating cross-system anomaly detection performance, we combine training samples from both the source and target systems to form the final training set $\mathcal{D}$, in alignment with the methodology described in the baseline method LogTAD [9]. Additionally, we assess single-system anomaly detection performance using only source system training samples, as shown in Table III.

2) Baselines: We compare LogDLR with the following unsupervised log anomaly detection methods:

- *PCA [34]:* PCA constructs feature vectors from the parsed data, and then uses principal component analysis (PCA) to mine the feature vectors and label them as normal or abnormal.
- *LogCluster [35]:* LogCluster first converts log sequences into weighted vectors, then clusters them by similarity and selects a representative log sequence for each cluster, and finally detects anomalies based on it.
- *DeepLog [3]:* DeepLog is a deep learning-based method that uses LSTM to learn log patterns from normal system executions and detect anomalies when deviations from the patterns occur.
- *LogTAD [9]:* LogTAD is a state-of-the-art unsupervised cross-system anomaly detection framework based on domain-adversarial adaptation, which uses an LSTM-based generator to make normal log data of different systems close to the center in a hypersphere and then uses the distance from the center to detect anomalies.

Among the four baseline methods, only LogTAD is specifically designed for cross-system anomaly detection. PCA, LogCluster, and DeepLog are non-cross-system methods and do not incorporate cross-system adaptive mechanisms. To ensure fair comparisons, the non-cross-system methods (PCA, LogCluster, and DeepLog) are trained on a mixture of source and target system data when evaluating anomaly detection performance in cross-system scenarios, as described in Section V-A1. When evaluating anomaly detection performance in single-system scenarios, since only the source system dataset is used, the cross-system methods (LogTAD and the proposed LogDLR) need to remove the cross-system adaptive mechanism (domain-adversarial training). Specifically, for LogTAD, only the objective function $\mathcal{L}_{\text{en}}$[9] of the LSTM-based generator is optimized. For LogDLR, only the objective function $\mathcal{L}_{\text{AE}}$ of the Transformer-based autoencoder is optimized. Notably, the respective anomaly detection principles of LogTAD and LogDLR remain unchanged after removing the cross-system adaptive mechanism.

In terms of implementation, for PCA, LogCluster, and DeepLog, we use the open-source code available on Github.[3,4] For LogTAD, we use the code provided by the original paper authors on GitHub.[5] For all baseline methods, we report their best results.

3) Metrics: We use the Precision, Recall, and F1-score metrics to evaluate the performance of LogDLR. These metrics can be calculated as follows:

$$Precision = \frac{TP}{TP + FP}, \tag{18}$$

$$Recall = \frac{TP}{TP + FN}, \tag{19}$$

$$F1 - score = \frac{2 \times Precision \times Recall}{Precision + Recall}, \tag{20}$$

where TP (True Positive) is the number of abnormal log sequences that the model correctly detects, FP (False Positive) is the number of normal log sequences that are mistakenly classified as anomalies and FN (False Negative) is the number of abnormal log sequences that the model fails to identify.

4) Implementation Details: We implement our method using Python 3.9 and Pytorch 1.12 on a Linux server with an NVIDIA A100-PCIE-40 GB GPU. Our method's Transformer-based autoencoder has $l = 4$ Transformer layers in both the encoder and the decoder, and the number of attention heads h is 16. The dimensions of the semantic vector d_e and the latent representation d_z are 384 and 512, respectively. We set the hyperparameters α and β in the objective function to 0.1 and 1.0, respectively.

To reduce the domain discriminator's noise early in training and improve the robustness of domain-adversarial training, we adopt the warm-up strategy from [12], gradually increasing the parameter λ from 0 to 1 instead of fixing it.

We use the Adam [36] optimizer and a cosine learning rate scheduler with a warm-up period for training. The initial learning

[3]https://github.com/logpai/loglizer
[4]https://github.com/donglee-afar/logdeep
[5]https://github.com/hanxiao0607/LogTAD

TABLE V
EVALUATION RESULTS OF THUNDERBIRD AS SOURCE SYSTEM

Methods	Thunderbird (Source only)			Thunderbird→Spirit			Thunderbird→BGL		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
PCA	0.705	0.903	0.792	0.451	0.965	0.615	0.553	0.791	0.651
LogCluster	0.872	0.702	0.778	0.866	0.679	0.762	0.498	0.960	0.656
DeepLog	0.830	0.941	0.882	0.820	0.847	0.833	0.634	0.957	0.763
LogTAD	0.870	0.978	0.921	0.884	**0.971**	0.926	0.887	0.924	0.905
LogDLR	**0.938**	**1.000**	**0.968**	**0.976**	0.962	**0.969**	**0.929**	**0.968**	**0.948**

* The left side of the "→" represents the source system, and the right side represents the target system. The reported results for LogDLR are the averages of 10 independent runs.

TABLE VI
EVALUATION RESULTS OF SPIRIT AS SOURCE SYSTEM

Methods	Spirit (Source only)			Spirit→Thunderbird			Spirit→BGL		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
PCA	0.552	**0.986**	0.708	0.491	0.958	0.649	0.353	0.699	0.469
LogCluster	0.913	0.703	0.795	0.797	0.575	0.668	0.585	0.901	0.709
DeepLog	0.867	0.938	0.901	0.688	0.876	0.771	0.657	0.884	0.754
LogTAD	0.902	0.956	0.928	0.777	0.915	0.841	0.846	0.917	0.880
LogDLR	**0.988**	0.976	**0.982**	**0.850**	**1.000**	**0.919**	**0.936**	**0.955**	**0.945**

TABLE VII
EVALUATION RESULTS OF BGL AS SOURCE SYSTEM

Methods	BGL (Source only)			BGL→Thunderbird			BGL→Spirit		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
PCA	0.641	0.709	0.674	0.440	0.972	0.605	0.430	0.870	0.576
LogCluster	0.868	0.900	0.883	0.703	0.655	0.678	0.902	0.642	0.749
DeepLog	0.848	0.948	0.896	0.715	0.780	0.746	0.755	0.911	0.826
LogTAD	0.952	0.946	0.950	0.724	0.910	0.806	0.882	**0.957**	0.918
LogDLR	**0.964**	**0.998**	**0.980**	**0.758**	**0.992**	**0.859**	**0.931**	0.951	**0.941**

rate is 0.0001, and the batch size is 32. These choices are based on best practices in training Transformer-based models and have been empirically validated to ensure stable convergence.

B. Performance Evaluation

The experiment utilizes the Thunderbird, Spirit, and BGL datasets to evaluate the performance of LogDLR in both single-system and cross-system anomaly detection. In single-system scenarios, only a single source system dataset is used for training, denoted as "Source Only" in the results. In cross-system scenarios, one dataset is designated as the source system and another as the target system, with both datasets mixed for training. Results are reported using "→" to represent cross-system evaluation; for example, "Thunderbird → Spirit" indicates the evaluation results of using Thunderbird as the source system on the target system Spirit. The experiment includes three single-system scenarios and six cross-system scenarios.

Table V, Table VI, and Table VII present the comparison results of LogDLR and baseline methods on the source and target systems. In single-system scenarios, when Thunderbird, Spirit, and BGL are used as source systems, LogDLR achieves the highest F1-score, with an average F1-score of 0.977, which is 4.4% higher than LogTAD (0.933). Among the deep learning-based baseline methods, LogTAD's performance is second only to LogDLR and superior to DeepLog. The traditional machine learning-based methods, PCA and LogCluster, perform worse than DeepLog and LogTAD. These results demonstrate that LogDLR can achieve effective unsupervised anomaly detection in single-system scenarios using only the Transformer-based autoencoder.

In cross-system scenarios, LogDLR achieves the highest F1-score on the target system, with an average F1-score of 0.930, which is 5.1% higher than LogTAD (0.879). This indicates that the Transformer-based autoencoder combined with adversarial training effectively extracts domain-invariant latent representations, overcoming differences in log distribution between source and target systems for cross-system anomaly detection. In contrast, the non-cross-system baseline methods, PCA, LogCluster, and DeepLog, exhibit lower F1-scores on the target system compared to the cross-system methods, LogTAD and LogDLR, due to the absence of cross-system adaptation mechanisms, leading to performance degradation when encountering data distribution shifts across systems.

Fig. 5 presents the evaluation results for different window sizes w in the Thunderbird → BGL scenario, with all other parameters set to default. The window size refers to the number of consecutive log entries used for feature extraction and anomaly

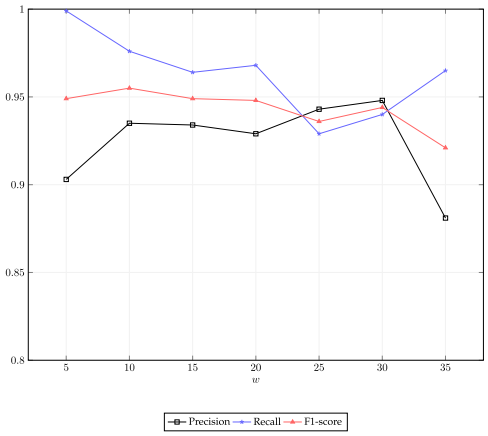


Fig. 5. Results of different window sizes w in the Thunderbird $\rightarrow$ BGL scenario. The x-axis is the window size w, and the y-axis is the precision, recall, and F1-score.

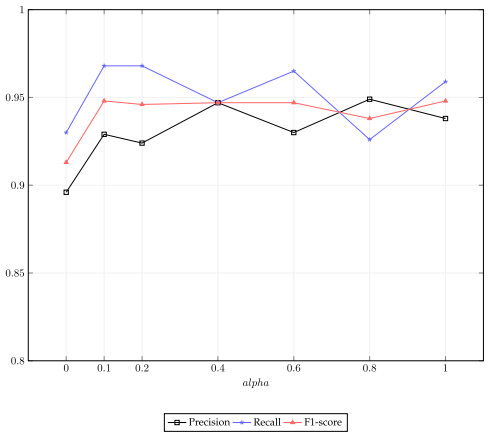


Fig. 6. Results of different parameters α in the Thunderbird $\rightarrow$ BGL scenario. The x-axis is the α and the y-axis is the precision, recall, and F1-score.

detection sequences. When w is between 10 and 30, all three metrics maintain high values. However, if w is too small or too large, the model's precision decreases.

Fig. 6 demonstrates the effect of the parameter α on the performance of LogDLR in the Thunderbird $\rightarrow$ BGL scenario, with all other parameters set to default. The parameter α controls the weight of the L_2 regularization term on the latent representation, encouraging compact and well-regularized feature representations. When $\alpha = 0$, the F1-score is low, indicating that the absence of regularization may lead to fitting noisy or system-specific features, resulting in degraded performance. As α increases, the F1-score remains high and relatively stable, emphasizing the importance of regularization in controlling the latent space and improving generalization. This demonstrates that adding regularization to the autoencoder enhances the model's ability to generalize across systems.

Fig. 7 illustrates the effect of the parameter β on the performance of LogDLR in the BGL $\rightarrow$ Spirit scenario, where β controls the weight of the domain-adversarial loss, with all other parameters set to default. When $\beta = 0$, no domain-adversarial loss is applied, resulting in low precision and F1-score. When β is between 0.6 and 1.2, the best balance between precision,

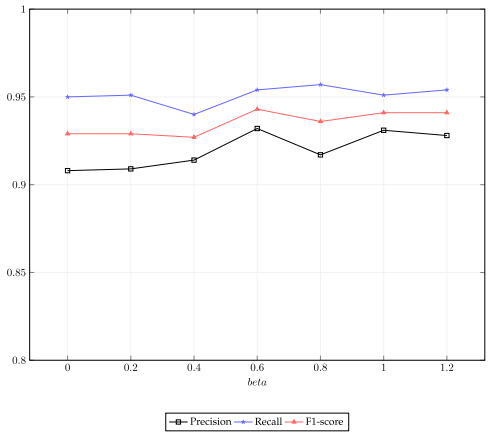


Fig. 7. Results of different parameters β in the BGL $\rightarrow$ Spirit scenario. The x-axis is the β, and the y-axis is the precision, recall, and F1-score.

TABLE VIII
PERFORMANCE OF DIFFERENT TRANSFORMER LAYERS l IN THE BGL $\rightarrow$ SPIRIT SCENARIO

# Layers	Precision	Recall	F1-score	Time per Batch (ms)
1	0.724	0.950	0.822	11
2	0.782	0.894	0.834	18
4	0.931	0.951	0.941	33
6	0.979	0.955	0.967	52
8	0.938	0.969	0.953	64

* Time per batch represents the average duration required to process a single batch during training, with a batch size of 32.

recall, and F1-score is achieved, indicating that a moderate domain-adversarial loss weight is well-suited for extracting domain-invariant features while maintaining anomaly detection performance.

Table VIII presents the evaluation results of autoencoder performance with different Transformer layer numbers l in the BGL $\rightarrow$ Spirit scenario, including the average training time per batch. The number of Transformer layers (l) is a crucial factor affecting model performance, as it determines the model's ability to capture long-distance dependencies and complex patterns in log sequences. However, an increase in the number of layers also leads to a significant rise in computational overhead, necessitating a balance between performance and computational cost. When $l = 1$, the model's feature extraction capability is limited due to insufficient depth, resulting in the lowest F1-score among all configurations. As the number of layers increases to 4, the model shows significant improvements in all metrics, reaching an F1-score of 0.941. This indicates that a moderate increase in the number of layers enhances anomaly detection capabilities. When $l = 6$, the model achieves peak performance with an F1-score of 0.967, verifying that a deeper architecture can effectively improve the ability to capture complex log patterns. However, when l increases to 8, the F1-score decreases slightly, possibly due to overfitting caused by too many layers. Additionally, as the number of layers increases, the calculation time per batch also rises significantly, from 11ms for $l = 1$ to 64ms for $l = 8$. Overall, these results indicate that the Transformer-based

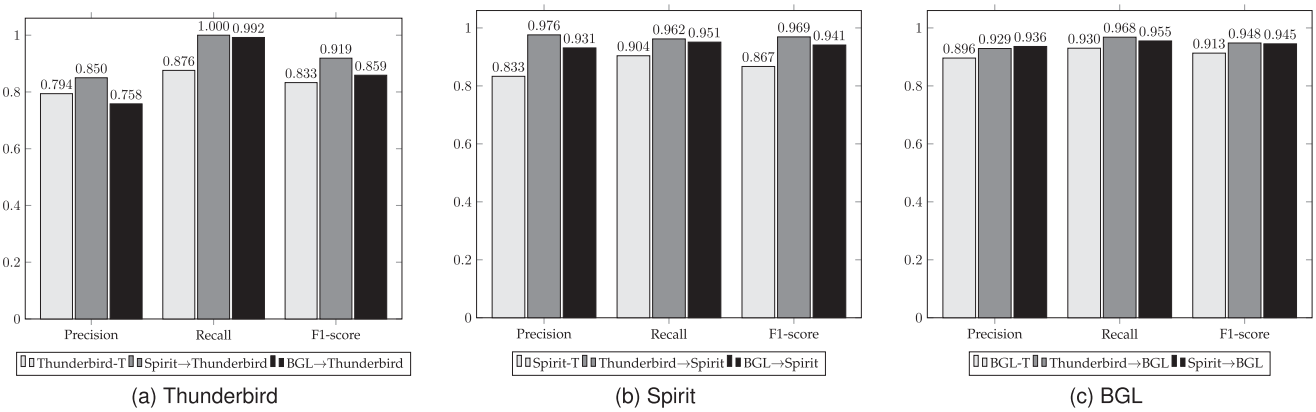


Fig. 8. Impact of source system data on target system detection performance.

autoencoder achieves the best balance of performance and computational cost with a moderate number of layers. This trade-off is particularly important for practical applications, especially in scenarios where large-scale log data needs to be processed, and computing overhead must be reasonably controlled to meet real-time and resource constraints.

C. Ablation Studies

- *RQ1:* How much does LogDLR improve the performance of the target system?

We investigate the performance improvement of LogDLR on the target system by removing the source system data and evaluating LogDLR only on the target system data of Thunderbird, Spirit, and BGL, as shown in Table IV. We denote these results as "*-T". Fig. 8 compares LogDLR with "*-T" on the target system. LogDLR achieves a higher F1-score than "*-T" on all target systems. The biggest improvement is in the Thunderbird → Spirit scenario, where LogDLR reaches an F1-score of 0.969, 10.2% higher than "Spirit-T" (0.867). This indicates that LogDLR's Transformer-based autoencoder effectively leverages the source system knowledge and enhances the detection performance on the target system by extracting domain-invariant latent representations of log sequences.

Furthermore, we observe that LogDLR has different impacts on the same target system depending on the source system. For example, the F1-score in the Thunderbird → BGL scenario is higher than that in the Spirit → BGL scenario. This may be due to the variability of log semantics across different systems, where systems with more semantic similarity can share more knowledge and achieve better transfer performance. This experiment demonstrates the effectiveness of LogDLR in improving the anomaly detection performance of the target system.

- *RQ2:* How effective is domain-adversarial training in cross-system detection?

We compare the performance of LogDLR with and without domain-adversarial training, by removing its domain discriminator and the GRL attached to the autoencoder. Fig. 9 shows the comparison results. LogDLR without domain-adversarial training has a lower F1-score on the target system than the original LogDLR. This suggests that domain-adversarial training helps LogDLR's Transformer-based autoencoder to learn domain-invariant latent representations more effectively, which enhances knowledge transfer between the source and target systems. This experiment demonstrates the effectiveness of domain-adversarial training in cross-system anomaly detection.

D. Visualization

In order to demonstrate the inner workings of LogDLR, we visualize the latent representations and reconstruction errors of log sequences in three different cross-system anomaly detection scenarios (Thunderbird → Spirit, Spirit → BGL, and BGL → Thunderbird). We randomly select 1,000 normal sequences each from the training sets of the source and target systems, and map their latent representations to a two-dimensional space using the t-SNE [37] algorithm. Fig. 10 shows the visualization results of the latent representations of logs before and after training in the three scenarios by LogDLR. Before training, the latent representations of the log sequences from the source and target systems are scattered, but after training, they overlap more. This shows that LogDLR extracts domain-invariant representations, where the latent representations of log sequences from both the source and target systems become more aligned and less affected by system-specific characteristics. The domain-invariant representation obtained by LogDLR is crucial for bridging the gap between the source and target systems. By aligning the distributions of latent representations, LogDLR enables meaningful comparisons and knowledge transfer between the two systems, despite their initial differences.

In addition, we randomly select 1,000 normal sequences and 1,000 abnormal sequences each from the test sets of the source and target systems, and visualize their reconstruction errors and corresponding detection thresholds. As shown in Fig. 11, the model generally has lower reconstruction errors for normal sequences and higher reconstruction errors for abnormal sequences. This indicates that LogDLR can effectively distinguish between normal and abnormal log sequences based on their reconstruction errors. By setting appropriate detection thresholds, LogDLR can perform unified anomaly detection on the source and target systems efficiently.

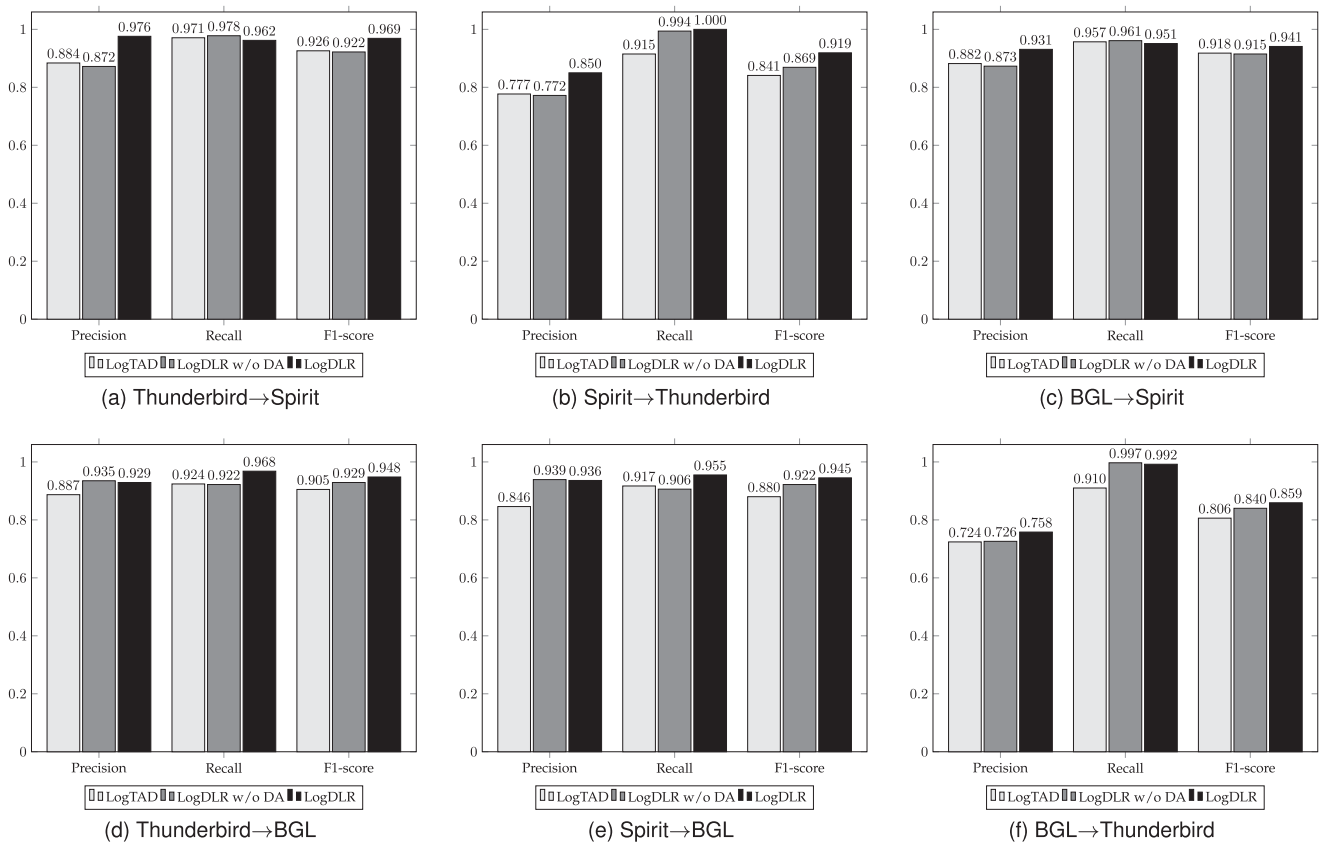


Fig. 9. Impact of domain-adversarial training on target system detection performance. "LogDLR w/o DA" refers to LogDLR without domain-adversarial training.

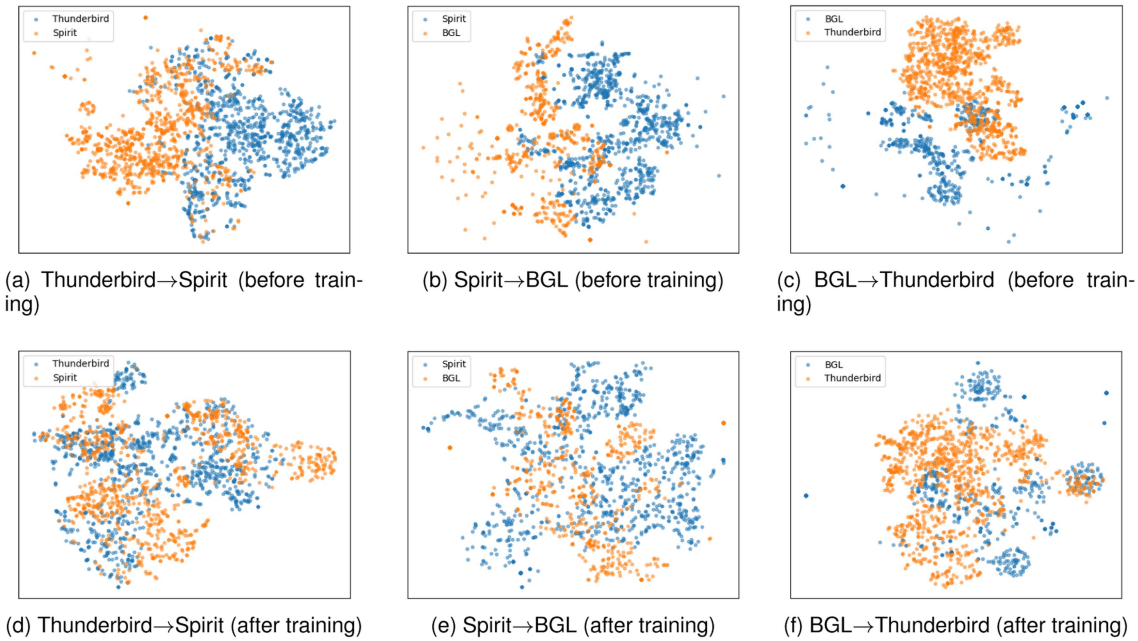


Fig. 10. Visualization of latent representations of log sequences before and after training. (a)–(c) represent Thunderbird → Spirit, Spirit → BGL, and BGL → Thunderbird scenarios before training, respectively. (d)–(f) represent the same scenarios after training.

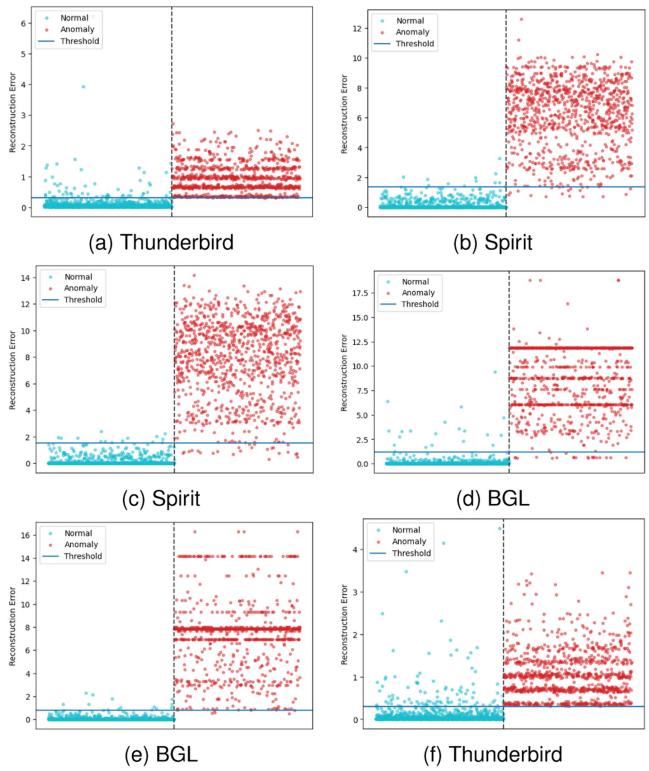


Fig. 11. Visualization of reconstruction errors of log sequences in different scenarios. (a), (c), and (e) represent the results on the source system (Thunderbird, Spirit, and BGL) in the Thunderbird → Spirit, Spirit → BGL, and BGL → Thunderbird scenarios, respectively. (b), (d), and (f) represent the results on the target system (Spirit, BGL, and Thunderbird) in the same scenarios.

VI. Threats to Validity

While this paper demonstrates the effectiveness of LogDLR in cross-system log anomaly detection, we acknowledge several threats that may affect the research results. First, the datasets used (Thunderbird, Spirit, BGL) are derived from public log datasets, which may not fully represent the log format and structural heterogeneity of all real systems. Therefore, we plan to further verify the generalization ability of LogDLR in various real systems in the future.

Second, LogDLR uses normal log data for unsupervised training; however, in practical applications, a small number of unlabeled abnormal samples may be mixed into the training dataset. This situation may affect the model's learning effect, particularly in extracting domain-invariant representations. Abnormal samples may be regarded as part of normal behavior by the model, leading to deviations in domain-invariant representations and ultimately affecting anomaly detection accuracy.

Finally, although the threshold selection method proposed in Algorithm 1 can ensure an optimal balance between precision and recall, its potential limitation is the reliance on a small number of labeled validation datasets. The performance of the selected threshold is affected by the quality and representativeness of the labeled validation data. To mitigate these limitations, future work will explore other threshold selection strategies, such as statistical methods based on percentiles.

VII. Related Work

Log anomaly detection methods can be categorized into traditional machine learning and deep learning approaches.

Early anomaly detection relied on models like Decision Trees [38], SVMs [39], and PCA [34], which used log count vectors as inputs. Unsupervised techniques like PCA [34] and LogCluster [35] grouped similar log entries to identify outliers. However, these methods often struggle with the complexity and high dimensionality of log data.

Deep learning approaches offer greater flexibility in modeling complex patterns. CNN-based models [40], [41] effectively capture local dependencies, while LSTM-based methods such as DeepLog [3], LogTransfer [2], LogTAD [9], Pylogsentiment [42], DeepSyslog [7], and SwissLog [8] model sequential patterns. Despite their success, LSTMs have limitations in capturing long-range dependencies and global information of log sequences.

Recent works demonstrate the superiority of Transformer-based models [11]. For instance, HitAnomaly [43] and HilBERT [44] utilize a hierarchical Transformer structure to model log template sequences, while LogBERT [45] adopts BERT to capture the patterns of normal log sequences through self-supervised training tasks. NeuralLog [6] employs a Transformer-based binary classification model to identify anomalies effectively. However, NeuralLog relies on a supervised learning paradigm, requiring labeled data for training. In contrast, LogDLR operates in an unsupervised manner, leveraging reconstruction errors and domain-invariant representations for anomaly detection, making it more practical in scenarios where labeled anomaly data is scarce or unavailable. Unlike NeuralLog, which focuses on single-system anomaly detection, LogDLR integrates a Transformer-based autoencoder with domain-adversarial training to achieve cross-system anomaly detection.

A. Unsupervised Anomaly Detection

Unsupervised methods presume that system logs generated during normal operation typically exhibit stable patterns [10]. When a failure or anomaly occurs, these patterns may be disrupted by events such as error logs, out-of-order log sequences, or premature termination of log sequences [46]. By learning the patterns of normal operation, these methods can automatically trigger an alarm when deviations from normal conditions are detected.

Traditional machine learning-based methods identify anomalies through unsupervised data mining techniques, such as PCA [34] and LogCluster [35]. Another class of methods predicts the next log template based on prior log sequences, as exemplified by DeepLog [3]. These methods predict the next log template within the input window and compare it with the actual template. If the actual template is not among the top k most likely templates predicted by the model, it is classified as abnormal. However, DeepLog heavily relies on the completeness of the sequence, resulting in degraded performance when log data is missing or nonlinear patterns are present [47].

TABLE IX
COMPARISON OF CROSS-SYSTEM LOG ANOMALY DETECTION METHODS

Methods	Feature Extraction	Cross-system Method	Detection Model	Detection Method	Unsupervised
LogTransfer	Word Embedding	Transfer Learning	LSTM-based	Binary Classifier	No
LogTAD	Word Embedding	Domain-Adversarial Training	LSTM-based	SVDD	Yes
LogDLR (Ours)	Sentence Embedding	Domain-Adversarial Training	Transformer-based	Autoencoder	Yes

A different category of methods is based on support vector data description (SVDD) [48], such as LogTAD [9]. The fundamental idea is to construct a hypersphere with the smallest possible radius to enclose most normal data points, with data points outside the hypersphere classified as abnormal.

Another approach involves using autoencoders to learn representations of normal log sequences, such as AutoLog [49]. A trained autoencoder accurately encodes normal log patterns, while abnormal instances produce significantly higher reconstruction errors [31], [50], [51]. These reconstruction errors serve as a critical metric for identifying anomalies. However, AutoLog [49] employs an MLP-based autoencoder, which is limited in capturing temporal and semantic features, particularly in heterogeneous log data. LogDLR overcomes these limitations by utilizing a Transformer-based autoencoder.

B. Cross-System Anomaly Detection

Cross-system anomaly detection reduces the need to develop separate models for different systems, significantly lowering maintenance costs [2].

LogTransfer [2] is the first method to propose transfer learning for transferring learned anomaly patterns from a source system to a target system using a shared fully connected network. However, it requires labeled data for both systems, limiting its applicability in real-world scenarios. To tackle this, LogTAD [9] introduces an unsupervised framework using an LSTM-based generator to align the distributions of normal log data from different systems via domain-adversarial adaptation. Anomalies are identified as points deviating from a hypersphere's center.

However, both LogTransfer and LogTAD utilize GloVe [14] and Word2Vec [15] for log semantics extraction, which are less effective than pre-trained models and sentence embeddings [7], [52], [53]. Both methods rely on LSTM-based anomaly detection models, but LSTM has been shown to be less effective than Transformer in capturing long-distance dependencies and complex log semantics. This limitation hinders their ability to adapt to diverse log distributions [6]. Table IX provides a detailed comparison of LogDLR with LogTransfer and LogTAD. LogDLR employs a Transformer-based autoencoder combined with domain-adversarial training to learn domain-invariant latent representations of normal log patterns, facilitating unsupervised cross-system anomaly detection. Additionally, LogDLR leverages pre-trained sentence embeddings (SBERT) to semantically represent heterogeneous log data more effectively than traditional word embedding methods used by LogTransfer and LogTAD. These features enable LogDLR to achieve a higher F1-score compared to LogTAD in unsupervised cross-system anomaly detection scenarios.

VIII. CONCLUSION

In this work, we propose a novel unsupervised cross-system log anomaly detection approach, which we call LogDLR. LogDLR aims to extract domain-invariant latent representations from log sequences of different systems, which capture the essential features of the log sequences while ignoring the format and syntax differences. We treat logs as natural language text and use a universal sentence embedding model to extract semantic vectors from log entries, enabling LogDLR to handle heterogeneous logs effectively. To obtain the domain-invariant latent representations of log sequences, we combine a Transformer-based autoencoder with domain-adversarial training. After learning domain-invariant representations of normal log sequences, we can perform unified anomaly detection on both source and target systems. Comprehensive evaluations on three public log datasets from different systems show that LogDLR outperforms the state-of-the-art method for unsupervised cross-system log anomaly detection.

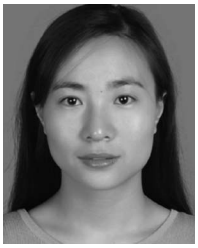
REFERENCES

[1] J. Zhu, P. He, Q. Fu, H. Zhang, M. R. Lyu, and D. Zhang, "Learning to log: Helping developers make informed logging decisions," in *Proc. 37th IEEE Int. Conf. Softw. Eng.*, 2015, pp. 415–425.
[2] R. Chen et al., "LogTransfer: Cross-system log anomaly detection for software systems with transfer learning," in *Proc. 31st IEEE Int. Symp. Softw. Rel. Eng.*, 2020, pp. 37–47.
[3] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1285–1298.
[4] W. Meng et al., "LogAnomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, 2019, pp. 4739–4745.
[5] X. Zhang et al., "Robust log-based anomaly detection on unstable log data," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Foundations Softw. Eng.*, 2019, pp. 807–817.
[6] V.-H. Le and H. Zhang, "Log-based anomaly detection without log parsing," in *Proc. 36th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2021, pp. 492–504.
[7] J. Zhou, Y. Qian, Q. Zou, P. Liu, and J. Xiang, "DeepSyslog: Deep anomaly detection on syslog using sentence embedding and metadata," *IEEE Trans. Inf. Forensics Secur.*, vol. 17, pp. 3051–3061, 2022.
[8] X. Li, P. Chen, L. Jing, Z. He, and G. Yu, "SwissLog: Robust anomaly detection and localization for interleaved unstructured logs," *IEEE Trans. Dependable Secure Comput.*, vol. 20, pp. 2762–2780, 2023.
[9] X. Han and S. Yuan, "Unsupervised cross-system log anomaly detection via domain adaptation," in *Proc. 30th ACM Int. Conf. Inf. Knowl. Manage.*, 2021, pp. 3068–3072.
[10] V.-H. Le and H. Zhang, "Log-based anomaly detection with deep learning: How far are we?," in *Proc. 44th Int. Conf. Softw. Eng.*, 2022, pp. 1356–1367.
[11] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1–11.
[12] Y. Ganin and V. Lempitsky, "Unsupervised domain adaptation by backpropagation," in *Proc. 32nd Int. Conf. Mach. Learn.*, 2015, pp. 1180–1189.
[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[14] J. Pennington, R. Socher, and C. Manning, "GloVe: Global vectors for word representation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2014, pp. 1532–1543.
[15] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 1–9.
[16] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Computat. Linguistics*, 2019, pp. 4171–4186.
[17] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using siamese BERT-networks," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2019, pp. 3982–3992.
[18] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, "Adversarial discriminative domain adaptation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 7167–7176.
[19] Y. Ganin et al., "Domain-adversarial training of neural networks," *J. Mach. Learn. Res.*, vol. 17, pp. 1–35, 2016.
[20] Y. Zhang, T. Liu, M. Long, and M. Jordan, "Bridging theory and algorithm for domain adaptation," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 7404–7413.
[21] J. Na, H. Jung, H. J. Chang, and W. Hwang, "FixBi: Bridging domain spaces for unsupervised domain adaptation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 1094–1103.
[22] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *Proc. IEEE Int. Conf. Web Serv.*, 2017, pp. 33–40.
[23] M. Du and F. Li, "Spell: Streaming parsing of system event logs," in *Proc. 16th IEEE Int. Conf. Data Mining*, 2016, pp. 859–864.
[24] J. Zhu et al., "Tools and benchmarks for automated log parsing," in *Proc. 41st Int. Conf. Softw. Eng.*, 2019, pp. 121–130.
[25] A. Vervaet, "MoniLog: An automated log-based anomaly detection system for cloud computing infrastructures," in *Proc. 37th Int. Conf. Data Eng.*, 2021, pp. 2739–2743.
[26] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "Towards automated log parsing for large-scale log data analysis," *IEEE Trans. Dependable Secure Comput.*, vol. 15, pp. 931–944, 2018.
[27] T. Zhang, H. Qiu, G. Castellano, M. Rifai, C. S. Chen, and F. Pianese, "System log parsing: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 35, pp. 8596–8614, 2023.
[28] P. Ghosh, M. S. M. Sajjadi, A. Vergari, M. Black, and B. Scholkopf, "From variational to deterministic autoencoders," in *Proc. Int. Conf. Learn. Representations*, 2020, pp. 1–25.
[29] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. 30th Int. Conf. Mach. Learn.*, 2013, pp. 1–6.
[30] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. W. Vaughan, "A theory of learning from different domains," *Mach. Learn.*, vol. 79, pp. 151–175, 2010.
[31] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini, "Anomaly detection using autoencoders in high performance computing systems," in *Proc. 31st Innov. Appl. Artif. Intell. Conf.*, 2019, pp. 9428–9433.
[32] J. Zhu, S. He, P. He, J. Liu, and M. R. Lyu, "Loghub: A large collection of system log datasets for ai-driven log analytics," in *Proc. 34th Int. Symp. Softw. Rel. Eng.*, 2023, pp. 355–366.
[33] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *Proc. 37th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2007, pp. 575–584.
[34] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proc. ACM SIGOPS 22nd Symp. Operating Syst. Princ.*, 2009, pp. 117–132.
[35] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *Proc. 38th Int. Conf. Softw. Eng. Companion*, 2016, pp. 102–111.
[36] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Representations*, 2015, pp. 1–15.
[37] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, pp. 2579–2605, 2008.
[38] M. Chen, A. Zheng, J. Lloyd, M. Jordan, and E. Brewer, "Failure diagnosis using decision trees," in *Proc. Int. Conf. Autonomic Comput.*, 2004, pp. 36–43.
[39] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, "Failure prediction in IBM BlueGene/L event logs," in *Proc. 7th IEEE Int. Conf. Data Mining*, 2007, pp. 583–588.
[40] S. Lu, X. Wei, Y. Li, and L. Wang, "Detecting anomaly in Big Data system logs using convolutional neural network," in *Proc. 16th Int. Conf. Dependable Autonomic Secure Comput*, 2018, pp. 151–158.
[41] B. Li, S. Ma, R. Deng, K.-K. R. Choo, and J. Yang, "Federated anomaly detection on system logs for the Internet of Things: A customizable and communication-efficient approach," *IEEE Trans. Netw. Service Manage.*, vol. 19, no. 2, pp. 1705–1716, Jun. 2022.
[42] H. Studiawan, F. Sohel, and C. Payne, "Anomaly detection in operating system logs with deep learning-based sentiment analysis," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 5, pp. 2136–2148, Sep./Oct. 2021.
[43] S. Huang et al., "HitAnomaly: Hierarchical transformers for anomaly detection in system log," *IEEE Trans. Netw. Service Manage.*, vol. 17, no. 4, pp. 2064–2076, Dec. 2020.
[44] S. Huang, Y. Liu, C. Fung, H. Wang, H. Yang, and Z. Luan, "Improving log-based anomaly detection by pre-training hierarchical transformers," *IEEE Trans. Comput.*, vol. 72, no. 9, pp. 2656–2667, Sep. 2023.
[45] H. Guo, S. Yuan, and X. Wu, "LogBERT: Log anomaly detection via BERT," in *Proc. Int. Joint Conf. Neural Netw.*, 2021, pp. 1–8.
[46] Z. Chen, J. Liu, W. Gu, Y. Su, and M. R. Lyu, "Experience report: Deep learning-based system log analysis for anomaly detection," 2021, *arXiv:2107.05908*.
[47] J. Li, H. He, S. Chen, D. Jin, and J. Yang, "LogGraph: Log event graph learning aided robust fine-grained anomaly diagnosis," *IEEE Trans. Dependable Secure Comput.*, vol. 21, no. 4, pp. 1876–1889, Jul./Aug. 2024.
[48] L. Ruff et al., "Deep one-class classification," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 4393–4402.
[49] M. Catillo, A. Pecchia, and U. Villano, "AutoLog: Anomaly detection by deep autoencoding of system logs," *Expert Syst. Appl.*, vol. 191, pp. 1–21, 2022.
[50] J. An and S. Cho, "Variational autoencoder based anomaly detection using reconstruction probability," *Special Lecture IE*, vol. 2, no. 1, pp. 1–18, 2015.
[51] D. Gong et al., "Memorizing normality to detect anomaly: Memory-augmented deep autoencoder for unsupervised anomaly detection," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 1705–1714.
[52] W. Wang, S. Lu, J. Luo, and C. Wu, "DeepUserLog: Deep anomaly detection on user log using semantic analysis and key-value data," in *Proc. IEEE 34th Int. Symp. Softw. Rel. Eng.*, 2023, pp. 172–182.
[53] X. Xie, S. Jian, C. Huang, F. Yu, and Y. Deng, "LogRep: Log-based anomaly detection by representing both semantic and numeric information in raw messages," in *Proc. IEEE 34th Int. Symp. Softw. Rel. Eng.*, 2023, pp. 194–206.

Junwei Zhou (Member, IEEE) received the PhD degree from the Department of Electrical Engineering, City University of Hong Kong. Since then, he has been a postdoctoral researcher with Pennsylvania State University, USA, a visiting scholar with the Polytechnic University of Turin, Italy, and a researcher with the City University of Hong Kong, Hong Kong, China. He is a professor with the School of Computer and Artificial Intelligence, Wuhan University of Technology, China. His research interests include computer vision, information security, and distributed source coding. He has served as a PC Member for top conferences, such as EMNLP 2023-2024, Coling 2022, ACL 2022-2024, and ISSRE 2022-2024.

Shaowen Ying received the BS degree in vehicle engineering from the Wuhan University of Technology, where he is currently working toward the MS degree in computer science and technology. His current research interests include log analysis, system reliability, and software engineering.



Shulan Wang received the PhD degree in information and communication engineering from Shenzhen University, Shenzhen, China, in 2016. She is an associate professor of Internet of Things Engineering with the College of Big Data and Internet, Shenzhen Technology University, Shenzhen. Her research interests include cryptography, cloud computing, and data security.

Dongdong Zhao received the PhD degree from the University of Science and Technology of China (USTC), in 2016. He is currently an associate professor with the School of Computer Science and Artificial Intelligence, Wuhan University of Technology. His research interests include dependable computing, information security, privacy protection and biometrics.

Jianwen Xiang (Member, IEEE) received the PhD degree in computer software and theory from Wuhan University and in information science from the Japan Advanced Institute of Science and Technology. Since then, he has been a special research fellow with the National Institute of Advanced Industrial Science and Technology, Japan, the Director of the Central Research Laboratory, NEC Corporation, Japan, and a Postdoctoral Researcher with the Japan Advanced Institute of Science and Technology. He is currently a professor with the School of Computer Science and Technology, Wuhan University of Technology, China, and a visiting professor with Kyushu University, Japan. His research interests include dependable computing, information security, formal methods, and knowledge management. He has served as a Reviewer for international journals, including *IEEE Transactions On Reliability, Reliability Engineering and System Safety*, *ACM Transactions on Software Engineering and Methodology*, *Journal of Systems and Software*, and IEICE Transactions.

Kaitai Liang (Member, IEEE) received the PhD degree in computer science from the Department of Computer Science, City University of Hong Kong. He worked as a postdoctoral researcher with Aalto University and held a permanent assistant professor position in secure systems with the University of Surrey. With over 11 years experiences on cybersecurity R&D, his main focus is on the design and implementation of cryptographic protocols to security. He has led as a PI in several European funded projects, e.g., TANGO, ASSURED, IRIS, and delivered real-world impact via these projects with academic and industrial partners (e.g., IBM, Fujitsu, NEC, EIT Manufacturing, BIBA, ATOS, UBITECH). He has also maintained a tight research relationship with Europe, Asia-pacific and northern America academic communities. He has published a series of research works ($>$ 140 publications, $>$ 4,500 citations), applying information security and crypto tools to address cybersecurity challenges. These publications have appeared in high-tier international information security journals and A* conferences, e.g., USENIX Security, NDSS, ESORICS (where he was honored with the Best Research Paper Award in 2015), *IEEE Transactions on Information Forensics and Security*, *IEEE Transactions on Dependable and Secure Computing*, IEEE IoT.

Peng Liu (Member, IEEE) received the BS and MS degrees from the University of Science and Technology of China, and the PhD degree from George Mason University, in 1999. He is the Raymond G. Tronzo, M.D. professor of cybersecurity, the Founding Director of the Center for CyberSecurity, Information Privacy, and Trust, and the Founding Director of the Cyber Security Laboratory, Penn State University. He has co-led the effort to make Penn State the NSA-Certified National Center of Excellence in Information Assurance Education and Research. He has advised or co-advised more than 35 Ph.D. dissertations to completion. He has published numerous papers on top conferences and journals. His research has been sponsored by NSF, ARO, AFOSR, DARPA, DHS, DOE, AFRL, NSA, TTC, CISCO, and HP. His research interests include all areas of computer security. He was a recipient of the DOE Early Career Principle Investigator Award. He has served as the Program (Co-)Chair or the General (Co-)Chair for over ten international conferences (e.g., Asia CCS 2010) and workshops (e.g., MTD 2016). He will serve as the PC Co-Chair for IEEE/IFIP DSN 2022. He has chaired the Steering Committee of SECURECOMM from 2008 to 2014. He has served on over 100 program committees and reviewed papers for numerous journals. He is the Editor-in-Chief of the Journal of Computer Security. He was an associate editor for *IEEE Transactions on Dependable And Secure Computing*.