Article

LogBD: A Log Anomaly Detection Method Based on Pre-trained Models and Domain Adaptation

Le Deng [1], Shuxian Liu [1,*], Huan Xu and[1] Wei Wang[1]

[1] College of Information Science and Engineering, Xinjiang University, Urumqi 830046, China; sc_dengle@stu.xju.edu.cn (L.D.); xhuan@stu.xju.edu.cn (H.X.); wangwei33@stu.xju.edu.cn (W.W.)

* Correspondence: liushuxian@xju.edu.cn

Abstract: The log data generated during the operation of a software system contains information about the system, and using logs for anomaly detection can detect system failures in a timely manner. Most of the existing log anomaly detection methods are specific to a particular system, have cold start problems, and are sensitive to updates in log format. In this paper, we propose a log anomaly detection method LogBD based on pre-training model and domain adaptation, which uses the pre-training model BERT to learn the semantic information of logs, and can solve the problems caused by multiple meaning words and log statement updates. An anomaly detection method LogBD based on distance to determine anomalies is constructed based on domain adaptation, using TCN networks to extract common features of different system logs and map them to the same hypersphere space. Finally, experiments are conducted on two publicly available datasets to evaluate the method, and the experimental results show that the method can better solve the log instability problem and has some improvement in the cross-system log anomaly detection effect.

Keywords: log anomaly detection; pre-trained models; domain adaptation

1. Introduction

With the needs of social development, computer software is widely used in various fields of life and the scale of software is increasing [1], while the technologies and configurations involved in software systems usually evolve and their hardware and software inevitably fail. In the wake of the Newcastle pneumonia outbreak, industries worldwide have accelerated their shift to online services, and online services in some key industries have profoundly affected the daily lives of people in various countries. A downtime of software systems that provide critical services will have a huge impact on society.

The increasing user base and gradually diversifying functional requirements have led to the expansion of the functional scope of various information systems, the gradual complexity of system architectures, and the existence of a large number of potential risks, resulting in frequent incidents. in 2015, Amazon Web Services (AWS) experienced a massive downtime that lasted more than 40 seconds. This resulted in unresponsiveness for several APPs such as Slack, Asana, Netflix, Pinterest, and multiple websites using AWS services. 2017, Baidu Search experienced an outage that caused search requests to fail to display results and took 42 minutes to recover. During the outage, hundreds of millions of searches were affected. The cause of the outage was due to a bug in a software update performed by Baidu, which caused the server to stop serving. in 2018, the AWS management console failed for nearly six hours, threatening its sales activities that lasted 36 hours [2]. in 2020, numerous services and major applications of Google, such as YouTube and Gmail, were inaccessible , and millions of users were affected to varying degrees with outage conditions. 2021, several key Fed-operated payment systems, which underpin millions of financial transactions, suffered service disruptions due to operational errors. This service disruption, in turn, was caused by a failure within the Federal Reserve's systems. Therefore, after a system or software is up and running, it is of paramount importance to maintain its proper functioning and to detect and deal with failures in a timely manner.

 doi:10.20944/preprints202305.1083.v1

Maintaining and ensuring the availability of the entire system is the responsibility of the O&M staff. In the face of the increasing demand for software reliability, early O&M was performed manually, but in the era of rapidly expanding Internet business and rising labor costs, traditional manual O&M has become unsustainable and cannot keep pace with the rapid development of the Internet [3]. Thus, automated O&M was born, which mainly uses scripts that can predefine rules and automatically trigger [4] to carry out routine, repetitive O&M tasks, thus reducing O&M costs and improving O&M efficiency. Automated O&M is an expert system based on industry and O&M scenario knowledge. With the rapid growth of Internet business scope and the diversification of service types, expert systems based on manually specified rules cannot meet the demand. the emergence of DevOps partially solves this problem, but it focuses more on the global perspective of value delivery and horizontal integration. AIOps is an advanced implementation of DevOps in O&M. the concept of AIOps was introduced in 2016 by Gartner [5], the goal of AIOps is to address the inefficiencies, poor accuracy, and lack of automation of traditional IT operations, which is based on three elements: data management, algorithms, and scenario-driven, and is formed by combining big data and machine learning technologies. Data analysis in AIOps is no longer limited to the scope of the data itself, but by mining specific patterns to predict possible events or diagnose the root cause of current system behavior. AIOps enables operators to act and solve problems more intelligently. A key focus of AIOps is fault discovery [6], and logs play an integral role in fault discovery.

Logs are one of the primary methods for recording operational status in IT domains such as operating systems, and are an important resource for identifying whether a system is in a healthy state, containing a wealth of system information. System developers and operations and maintenance personnel often use log data to view system status, find system failures, and locate root causes. The rich information hidden provides a good perspective for analyzing software system problems. Therefore, mining log information in large amounts of log data can help enhance system health, stability and availability, and help operations and maintenance personnel to find abnormalities in system operation status.

Most of the existing approaches learn normal patterns from a large number of normal logs, and the models examine the abnormal log sequences based on the learned normal patterns. Most of the methods use static word embedding methods, such as Word2Vec, Glove, while words may have different meanings in different contexts, and dynamic word embedding methods are needed. The above methods are also not studied for cross-system log anomaly detection, they are all for a specific system, and the models obtained from training do not work well on new systems. In this paper, we propose LogBD to build a cross-system log anomaly detection model to detect anomalies in both source and target systems, using the semantic and sequential relationships of log messages to achieve anomaly detection results using very few logs in the target system.

The main contributions of this paper are as follows:

1. Uses BERT [29] to extract the semantic features of log messages and feeds the log sequence vector with preserved semantic information into a neural network, which learns the normal patterns and semantic information of logs and is able to better detect abnormal log sequences.
2. Input the log sequence vector into the TCN (Temporal Convolutional Network) [35] network for data mapping, and use the hypersphere objective function to map the log sequences into the hypersphere space, the distance between normal log sequences and abnormal log sequences to the center of the sphere is different, based on the distance mapped to the center of the sphere to determine anomaly.
3. Uses domain adaptation learning to train the TCN network so that it can extract common features from log data from different systems and apply the "knowledge" learned from the source system data directly to the target system data, thus enabling the detection model to detect anomalies from multiple systems.

The remainder of this paper is organized as follows. The relevant knowledge and work is presented in Section 2, and the details of our work are presented in Section 3. Finally, the performance of the proposed model is evaluated in Section 4, and the work of this paper is summarized in Section 5.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 May 2023 doi:10.20944/preprints202305.1083.v1

2. Relate Work

Log anomaly detection Log anomaly detection generally has four steps, focusing on three parts: log parsing, feature selection, and anomaly detection. We address these three parts to introduce the related work.

2.1. Log Parsing

This part of log parsing is the task of parsing the raw log statements into log templates or log events. The body of a log statement usually consists of two parts: (1) Templates: static keywords describing system events, usually a piece of natural language, that are explicitly written in the code of the log statement. (2) Parameters: also called dynamic variables, which are the values of a variable during the program runtime. In logging exception detection operation, the log content is the most important. Log parsing is generally done to parse the log content, replace the variables and keep the constants to get the log template. As shown in Figure 1.

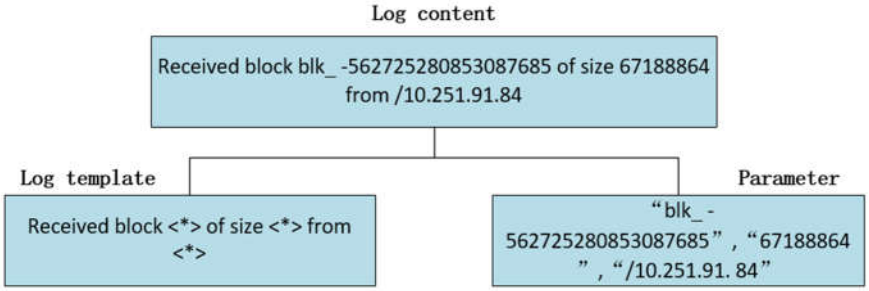

Figure 1. Result of log parsing.

Log parsing has been studied by many scholars, and log parsing algorithms are divided into three main categories: clustering-based log pattern parsing algorithms, frequent item mining-based log pattern parsing algorithms, and heuristic-based log pattern parsing algorithms. The main clustering-based log pattern parsing algorithm is the Spell method. the Spell [7] algorithm is a structured streaming event log parser based on LCS (longest common subsequence) to parse unstructured log messages into structured log types and parameters in an online streaming fashion. The main log pattern parsing algorithm based on frequent item mining is the FT-Tree [8] algorithm, which is a general log template generation method that uses an extended prefix tree structure to represent switch syslog message templates, where the subtypes of the detail fields are usually made up of the longest combinations of frequently occurring words, so extracting the templates is equivalent to identifying from the syslog messages the the longest combination of frequently occurring words from syslog messages. The main heuristic-based log pattern parsing algorithm is the Drain [9] algorithm, which is a fixed-depth tree-based approach that enables streaming and timely log parsing. Drain uses domain knowledge and regular expressions to preprocess new logs as they arrive. Drain then searches for log groups according to the encoding rules of the nodes inside the tree. If a suitable log group is found, the log message is matched to a log event stored in that log group. Otherwise, a new log group is created based on the log messages.

2.2. Feature Extraction

After processing the raw logs and before anomaly detection, log feature representation is also needed because log templates in the form of text data cannot be processed directly by computers, so feature representation is needed before model training and anomaly detection as a way to learn normal or abnormal patterns of logs. In previous studies, scholars selected features from system logs [10], including log templates, number of events, intervals, event indexes, and log variables, and then encoded them for use in neural networks by unique thermal coding or other methods. For example, event sequences are represented as event ID sequence vectors (IDs), i.e., sequences of log templates in chronological order, and are fed into an RNN to learn temporal correlations, treating unusual template sequences among them as anomalies; One-hot Encoding is one of the most common

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 May 2023 doi:10.20944/preprints202305.1083.v1

techniques for processing categorical data and thus is often applied to encoding event types or tokens. Most scholars use One-hot Encoding data directly as input to neural networks, but it can also be combined with other features (e.g., counting vectors) so that neural networks can recognize the input and learn different models for different log templates; One-hot Encoding may be too dimensional in some cases where the amount of data is too large, and some scholars use the NLP methods to replace it, such as Word2Vec and Glove, which are able to learn the semantic information of logs and better utilize the information contained in log data.

2.3. Anomaly Detection

The existing anomaly detection methods based on log data are mainly rule-based, graph-based and machine learning-based detection methods [10]. Rule-based log anomaly detection methods use the frequency of keywords appearing in logs, combine domain knowledge, and design relevant regular expressions to mine potential information in log data. Graph-based anomaly detection uses the contextual relationships of log sequences to model and construct a request execution logic graph. Currently, machine learning-based methods include traditional machine learning methods and deep learning methods, which mainly use neural network models to infer whether logs are abnormal by judging log sequence relationships.

Among the traditional machine learning methods, Xu W et al construct a log anomaly detection method based on principal component analysis [11], which constructs log feature vectors from the state ratio vectors and event count vectors during system execution, and combines PCA with term weighting techniques in the field of data retrieval, which can discover patterns between event count vectors; Chen et al propose a decision tree based detection method [12], which uses decision trees to handle discrete and continuous attributes to perform anomaly detection by analyzing server request sequences; MENG W B proposed LogClass [13], which uses a data-driven approach to detect and classify anomalies in logs. First, the raw log data is preprocessed to generate a bag-of-words model description approach. Then, PU learning methods and support vector machines are used to train anomaly detection and classification models. Finally, anomaly detection is performed on logs; LOU J G et al. proposed invariant mining (IM) to mine information in log data [14], which mines invariants from console logs by a strong search algorithm, and detects system anomalies if the appearance of a new log sequence breaks an invariant during the life cycle of system execution; Vaarandi R et al. proposed logcluster [15], a log clustering-based problem identification method that uses hierarchical clustering techniques to aggregate similar log sequences into clusters for automatic problem identification.

As the size of log data increases, more human and material resources are required for feature engineering, and the cost of traditional machine learning methods increases greatly, many scholars apply deep learning methods to log anomaly detection. Du et al. proposed DeepLog [16], a deep learning model based on long and short-term memory network (LSTM), which is the first to use deep learning for detecting log data sequences that constructs log sequences as time series to achieve anomaly detection of log template sequences and parameter values; Meng et al. design a vectorized representation of templates inspired by word embbeding, template2Vec, and propose LogAnomaly [17], an anomaly detection framework for unstructured log streams; ZHANG XU et al. propose the LogRobust model [18], which uses the semantic encoding method FastText to extract log semantic information and represent it as a semantic vector for the unstable characteristics of logs, and learns semantic information and detects anomalies through the BiLSTM model, which can categorize log events as semantic approximations when logs are updated, solving the instability of log events caused by log updates, etc; Xia B et al. used generative adversarial networks based on LSTM of system logs using permutation event modeling to detect anomalies, called LogGAN [19], to detect log-level anomalies based on patterns. The permutation event modeling mitigates the strong sequential features in LSTM and solves the problem of misordering caused by the late arrival of logs. Generating adversarial networks mitigates the effect of imbalance between normal and abnormal data and improves anomaly detection; Chen R et al. argued that it is impractical to build log anomaly detectors to apply to all types of software systems [20], but there are similar parts between different log types

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 May 2023 doi:10.20944/preprints202305.1083.v1

of different software systems, and migration learning is used to transfer knowledge on the source system to the target system; Haixuan Guo et al. proposed a BERT-based log anomaly detection method LogBERT [21], which learns the patterns of normal log sequences by two self-supervised training tasks and is able to detect anomalies that deviate from normal log sequences. Zhang C K et al. proposed an unsupervised log sequence anomaly detection network LSADNET based on local information extraction and global sparse transformer model [22]. Multilayer convolution is applied to capture local correlations between adjacent logs and the Transformer is used to learn global dependencies between distant logs.

The shortcomings of existing studies are as follows:

1. Log format is constantly changing, and most methods are based on the assumption that log templates do not change. Word2Vec method used in most current studies to represent log templates. Word2Vec is a method to convert words into vectors, and using shallow neural network training, words can be converted into vector form. However, Word2Vec is a static word embedding method, and the word vectors of corresponding words are fixed without considering the contextual background, extracting no semantic information, unable to represent the similarity between different templates with the same or similar semantics, and unable to effectively solve the problem of multiple meaning words.
2. When the anomaly detection model is applied to a new system, it takes some time to collect enough logs to train the model, and there is a cold start problem. In a cross-system setup, the log templates of the two may be completely different and semantically similar. It is not feasible to use the knowledge of one system to fit the anomaly detection model of the other system.

In this paper, we propose the log anomaly detection method LogBD, which uses a pre-trained model BERT to extract log semantic information and obtain semantic vectors as features that can better learn the information contained in the logs; a TCN network is trained as a feature extractor using a domain adaptation method to extract common features of logs from different systems and build a cross-system log anomaly detection model to detect the source and target anomalies in the system.

3. Methodology

3.1. Overview

Logs record information about the state of software systems when they are running, and have become an important resource to ensure the reliability and continuity of software systems. However, it is quite difficult to discover system anomalies from massive log data, and based on deep learning and domain adaptation, we propose a domain adaptation-based log anomaly detection method, LogBD, which is outlined in Figure 2.

The first step is log parsing, which extracts log contents from unstructured raw logs, and then removes numbers and punctuation marks from logs by regular matching. (b) After comparing some log parsing methods, we use Drain for log parsing, which efficiently converts pre-processed log constants into log templates;

The second step is feature extraction, after log parsing the log templates are fed into the pre-trained model BERT, a bi-directional encoder based on Transformer [31] proposed by Google, through which semantic vectors containing semantic information of the log templates are obtained. After representing the log templates as numeric vectors that the neural network can use, the BERT model uses a sliding window method segmentation of the original log sequence into datasets;

The third step is anomaly detection, where the log template sequence after the digital vector representation is fed to a deep learning model for training to learn the patterns of normal log sequences and generate an anomaly detection model based on the distance of the log sequence mapping to the center of the hypersphere sphere.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 May 2023 doi:10.20944/preprints202305.1083.v1

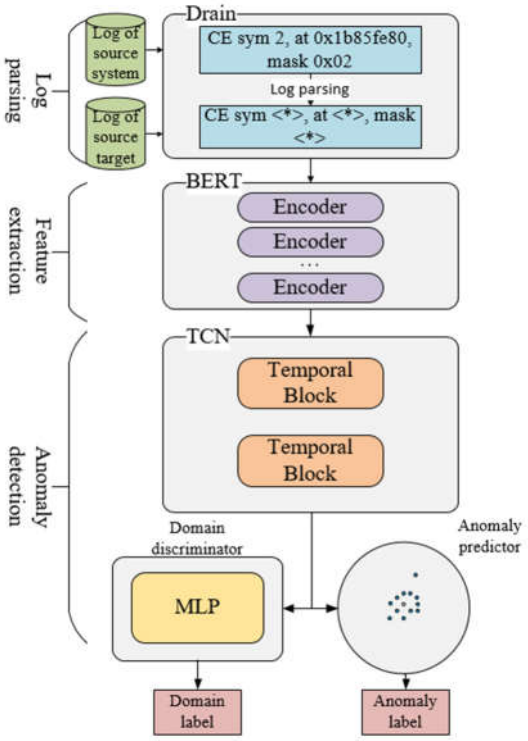

Figure 2. The framework of LogBD.

3.2. Log Parsing

The data required for a general neural network model is structured data, and the raw logs are usually a large amount of repetitive unstructured data. The step of log parsing is to parse the raw logs into log templates. Traditional log parsing methods rely on regular expressions, which require manual writing, as well as the need to maintain template updates, and cannot be effectively processed for the large amount of log data generated by modern systems. To improve the efficiency and accuracy of anomaly detection, this paper uses Drain to extract relevant information from the logs. "081109 203518 143 INFO dfs.DataNode$DataXceiver: Received block blk 560063894682806537 of size 67108864 from /10.251.194.129 " is a raw log from the HDFS log dataset [23], "081109 203518" is the timestamp of the log data, "143" is the process number of the log data, and "INFO" is the hazard level of the log data, " dfs.DataNode$DataXceiver" is the component that generated the log data, "Received block blk 560063894682806537 of size 67108864 from /10.251.194.129" is the content of this log data, which records the operation status of this system. The original log contains some irrelevant information, such as IP address, timestamp and fast ID, which is removed by predefined regular expressions based on domain knowledge, and the content of the log statement is obtained after removing the irrelevant information. Receiving block blk <*> of size <*> from <*>".

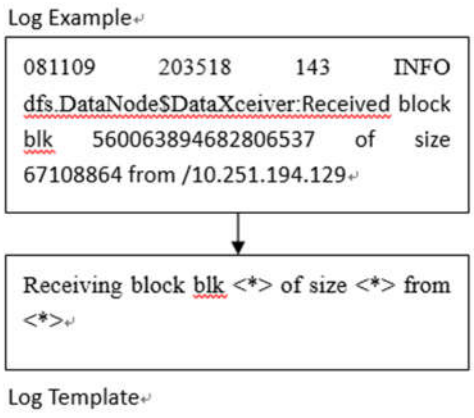

Figure 3. Example of log parsing.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 May 2023 doi:10.20944/preprints202305.1083.v1

3.3. Feature Extraction

The log templates are textual data, while the data processed by machine learning and deep learning algorithms are usually fixed-length numerical inputs, and the log templates in textual form must be converted into vectors for computer processing, and this method chooses the word embedding method to obtain the semantic vectors.

During the development of natural language processing, word embeddings have undergone an evolution from static to dynamic [24]. Static word embeddings have evolved in two broad stages, and in the former stage mainly sparse and high-dimensional vectors are used to represent words, such as One-hot encoding. Each word of a sentence corresponds to a high-dimensional vector, and all bits in the vector are "0" except for a certain bit which is "1". This cannot extract semantic similarity, and the vector dimension of this method is very large, and there are many zero values, which wastes storage space. In the latter stage, to solve these problems, neural networks are trained using large amounts of text data to generate low-dimensional vectors, and Word2Vec [25] and Glove [26] are among the classical approaches. Glove optimizes Word2Vec by considering not only the adjacent local information but also using the co-occurrence matrix to count the global information for better results. Static word embeddings produce vector representations and words in a one-to-one correspondence, which cannot solve the problem of multiple meaning words in log utterances, resulting in incomplete extracted semantic information. dynamic word embedding methods have emerged, including ELMo [27] (Embeddings from Language Models), GPT [28] (Gererate Pre- Training), BERT [29] (Bidirectional Encoder Representations from Transformers), etc. ELMo extracts the forward and reverse sequence information based on the input text information using a bidirectional LSTM [30] model, and the extracted information is used after stitching. The word vector is generated dynamically based on the whole contextual information. GPT uses a one-way Transformer as a feature extractor for extracting the above information without considering the reverse contextual information. In contrast, BERT changes the LSTM to Transformer based on ELMo to extract bi-directional context information at the same time, changing from word-level embedding to sentence-level embedding and preserving more semantic information. Firstly, the tokenizer of nltk package is used to remove the punctuation from the log template, and the * in the template is removed together, leaving the token of the log template to recompose the sentence input to BERT, and the embedding layer outputs the vector representation of the sentence by Token embedding, Position embedding and Segment embedding. The vector representation is then input to the encoding layer, where the Encoder module of the two Transformers [31] in the encoding layer processes the vector representation, and each word is computed with the attention of the words before and after it to obtain a more accurate contextual representation. After the Encoder module to achieve the semantic vector encoding of the log sequence, the final pooling layer performs an average pooling operation on the output of all tokens to obtain a fixed size vector as the representation of the whole input sequence. The feature representation process is shown in Figure 4.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 May 2023 doi:10.20944/preprints202305.1083.v1

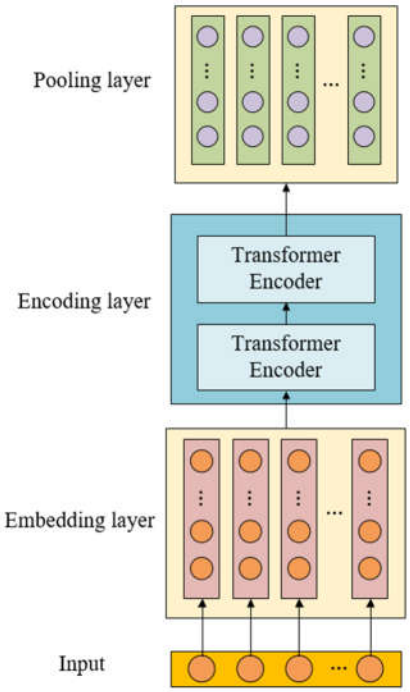

Figure 4. Get vector representation by BERT.

3.3. Anomaly Detection

Log anomaly detection is an important step to ensure stable and sustainable operation of software systems, and many machine learning based log anomaly detection methods are proposed to detect anomalous log sequences, due to limited anomalous logs, many methods detect anomalous log sequences based on log normal patterns by learning log normal patterns from a large amount of normal log data, but most methods are applied to a system and when deploying to a new system, a large number of logs need to be collected in the new system to retrain the model [20]. For the cold start problem, solutions have also been proposed, such as LogTransfer, which assumes that anomalous log sequences in different systems may have similar patterns and log message words from different systems usually overlap. Also, there are some similarities in the normal workflow of different systems, and migration learning can be used to construct a cross-system anomaly detection model to detect anomalies in the source and target systems, but LogTransfer requires a large amount of normal and anomaly log data from the source and target systems to construct the anomaly detection model, and it is actually difficult to collect enough log data in a new system [32].

To address the limitations of previous studies, we propose a domain-adaptive cross-system log anomaly detection method based on LogBD. LogBD establishes an adversarial training framework by adversarial Domain-Adversarial Training of Neural Networks (DANN) [33], and simultaneously makes anomalies in logs and attribution domains, and finally achieve the effect of detecting anomalies with confusing data domains to achieve cross-domain (system) log anomaly detection, which requires only a small amount of target system log data to achieve better performance. The adversarial domain adaptive network is a deep learning model for domain adaptation, which achieves feature alignment between source domain (system) and target domain (system) by adversarial training, learns feature representations that can effectively classify both source and target domains, and improves the generalization performance of the model. The network architecture of the DANN model is shown in Figure 5, and contains three parts: feature extractor, anomaly classifier and domain discriminator.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 May 2023 doi:10.20944/preprints202305.1083.v1

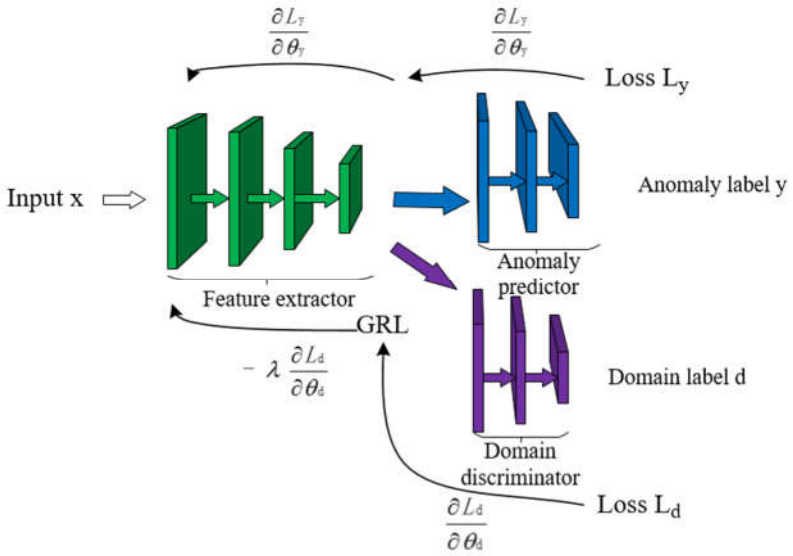

Figure 5. The framework of DANN.

3.3.1. Feature Extractor

LogBD trains the anomaly detection model using only normal log data, and D is a dataset consisting of two parts: normal log messages $D^S=\{L_i\}_{i=1}^{Max_S}$ of the source system and a small number of normal log messages $D^T=\{L_i\}_{i=1}^{Max_T}$ of the target system. After generating the log template vector representation by the pre-training model BERT, each log template is converted to a semantic vector representation, and a log sequence is represented by $X=\{x_n\}_1^n$, where x_n denotes the vector representation of the nth log message.

Inspired by the Deep SVDD method, LogBD's feature extractor maps log sequences to hypersphere space [34], which needs to be processed by neural networks, and deep neural networks are better at extracting the depth information contained in log data than traditional machine learning methods, and since log data carry temporal information, many scholars treat them as time-ordered sequences and use RNNs to learn log the temporal relationships between them. Although RNNs establish connections in temporal order with some memorability, they are prone to gradient disappearance or gradient explosion. On the basis of RNN, scholar Schmidhuber proposed LSTM network, and compared with RNN, LSTM introduces a new temporal chain on the basis of RNN. Long-range dependencies can be better captured using the LSTM model because it possesses a gating mechanism to increase or decrease the information of cell states and ultimately decide which information to use. However, LSTM has some problems, i.e., it cannot operate in parallel, only one step can be processed in one operation, and the latter step must wait for the previous step to be processed before performing the operation, and keeping the intermediate results is very memory consuming. Therefore, this method uses the TCN [35] network with better performance and less consumption as the feature extractor G_f. As shown in Figure 6, a vector representation of log sequences is input to the TCN for encoding and processing to obtain the log sequence mapping v. The i th sequence from the source system is mapped as v_i^S and the i th sequence from the target system is mapped as v_i^T.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 May 2023 doi:10.20944/preprints202305.1083.v1

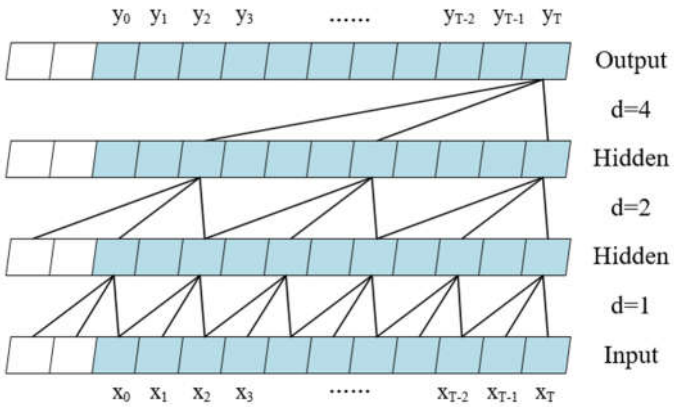

Figure 6. Feature extractor processing log sequence representation.

For the feature extractor G_f, to have all log sequences mapped into a hypersphere space, first set the center of the hypersphere to which all log sequences of the source and target systems are mapped to the mean value of all log sequences in the training dataset, The formula is as follows:

$$c = Mean\left(\sum_{i=1}^{Max_\epsilon} v_i^\epsilon\right), \epsilon \in \{S, T\} \quad (1)$$

where S denotes Source domain, the source system; T denotes Target domain, the target system.

Since each log sequence is represented by the feature extractor as $v=G_f(x;\theta_f)$, the distance of the log sequence mapping to the center of the hypersphere sphere is v_i^ϵ-c. In order to make the normal log sequence mapping close to the center of the hypersphere c, the distances of the log sequence mapping to the center of the sphere all need to be close to the minimum, setting the objective function of the feature extractor as follows:

$$\mathcal{L}_{ex} = \sum_{\epsilon\in\{S,T\}} \sum_{i=1}^{Max_\epsilon} \|v_i^\epsilon - c\|^2 + \frac{\lambda}{2}\sum_{l=1}^{L} \|W^l\|_F^2 \quad (2)$$

By minimizing this formula, all normal log sequences can be kept close to the center. The ex in the formula represents the feature extractor, and the first term in the formula makes all log sequences mapped as close to the center of the sphere as possible, and the second term is regularized.

The loss function of the feature extractor is calculated by the mean square error (MSE), and the feature extractor is optimized to keep the log sequences close to the center of the sphere by minimizing this loss function, which is calculated as follows:

$$loss(v_i, c) = \frac{1}{n}\sum_{i=1}^{n}(v_i - c)^2 \quad (3)$$

3.3.2. Adversarial Learning

Although LogBD uses a TCN network to map log sequences into the hypersphere space, log sequence mappings from different systems can still be located in different regions of the hypersphere space, rather than clustered around a common hypersphere sphere center c, as shown in Figure 7.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 May 2023 doi:10.20944/preprints202305.1083.v1

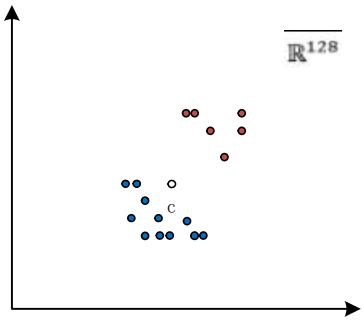

Figure 7. Different systems data clustered in different regions.

Therefore, this paper proposes a cross-system data mapping method for adversarial training, which brings log sequence mappings from different systems close to each other. A domain discriminator G_d is set to distinguish whether the log sequence mapping comes from the source or the target system, and the domain discriminator uses a multilayer perceptron containing two 64-dimensional fully connected neural networks. A logistic function is used for prediction, as follows:

$$G_d(v^\epsilon) = \sigma(w^T v^\epsilon + b) \tag{4}$$

where $\sigma()$ denotes the logistic function and w and b are trainable parameters.

The loss function of the domain discriminator is calculated by the cross-entropy loss function, which is as follows:

$$loss(\hat{d}, d) = -\hat{d}[d] + \log\left(\sum_i \exp(\hat{d}[i])\right) \tag{5}$$

where d is the true domain label of the log and $\hat{d}$ is the domain prediction label of the domain discriminator output.

The TCN network is trained as a feature extractor, i.e., $v^\epsilon = G_f(X^\epsilon)$, so that it gains the ability to extract features common to both the source and target systems, making the discriminator unable to distinguish the source system data from the target system data. The adversarial training of the feature extractor and the domain discriminator is achieved by the gradient reversal layer (GRL)[33], where the GRL multiplies the error passed back to the feature extractor by a negative number (-λ) from the domain discriminator to achieve the opposite of the training objective and achieve the adversarial effect. The objective function of adversarial training is as follows:

$$\mathcal{L}_{adv} = \min_{G_f} \max_{G_d} \begin{pmatrix} \mathbb{E}_{X^S \sim P_{source}}\left[log G_d(G_f(X^S))\right] + \\ \mathbb{E}_{X^T \sim P_{target}}\left[log\left(1 - G_d\left(G_f(X^T)\right)\right)\right] \end{pmatrix} \tag{6}$$

where X^S and X^T denote the source log sequence and the target log sequence, respectively.

By minmax optimization training, the log sequence mapping generated by the TCN network is able to mislead the domain discriminator, i.e., the distributions of source and target log sequences are confused. Finally, the objective function training framework by the following equation:

$$\mathcal{L} = \mathcal{L}_{ex} + \lambda \mathcal{L}_{adc} \tag{7}$$

λ is a hyperparameter that is used to balance these two components, and its value is set to 0.1.

During the LogBD training process, an optimizer is needed to update the model parameters. In this paper, we use the Adam optimizer to optimize the model, which maintains the average of the gradient of each parameter, and the average of the gradient squared, and uses these estimates to calculate the adaptive learning rate for each parameter.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 May 2023 doi:10.20944/preprints202305.1083.v1

3.3.3. Anomaly Detection

The anomaly classifier determines whether a log sequence is anomalous based on the distance mapped to the center of the sphere. As shown in Figure 8, the blue normal data points are scattered haphazardly in the space. By training the feature extractor TCN network, the data points are mapped to the hypersphere space, which has a hypersphere centered at c. The TCN network is trained to make the normal data points close to the hypersphere sphere center, and then the anomaly detection is performed by calculating the anomaly distance threshold using some normal and anomaly data.

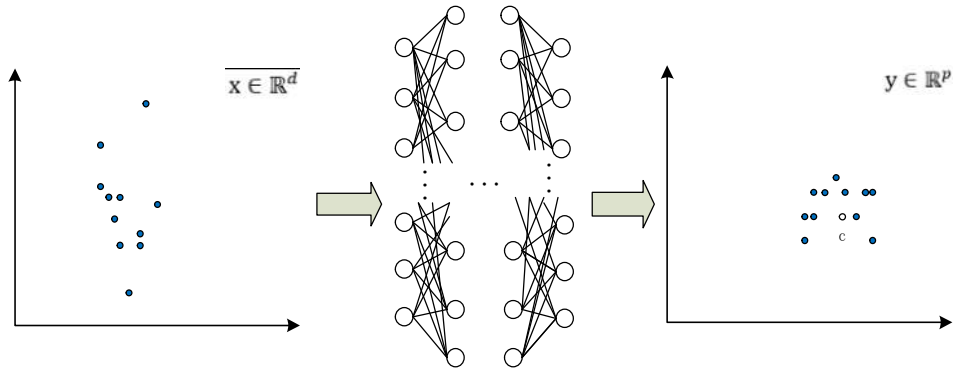

Figure 8. Principle of hypersphere anomaly detection.

After the training phase, the feature extractor gains the ability to map both the source and target system data to the vicinity of the hypersphere sphere center c, making the anomalous samples of both systems at a large distance from the center of c. To detect anomalies, a threshold γ needs to be set as the determination boundary to distinguish between normal and anomalous sequences. The validation set is used to find the optimal thresholds γ^S and γ^T in the source and target systems, respectively. Samples with a distance greater than γ from the center will be marked as abnormal sequences, and the validation set is composed of log sequences from any day.

Firstly, the average value of normal log sequences and abnormal log sequences in the validation set after mapping positions are calculated and noted as $Mean^{normal}$ and $Mean^{abnormal}$, and the threshold is set to the average value of normal sequence positions $Mean^{normal}$, and the abnormal labels of log sequences are given by comparing the distance of log sequences mapped to the center of the hypersphere sphere and the magnitude of the threshold, and then by the log If the AUC is greater than the current AUC value, the best threshold value is changed to the current threshold setting and the best AUC is also changed to the current AUC value, otherwise the best threshold value and the best AUC do not change, and the threshold value increases toward $Mean^{abnormal}$ at the end of judgment to expand the judgment range of normal log sequence. The above operation is cycled 100 times and the best threshold value γ is finally obtained.

4. Experiment and Analysis

Firstly, LogBD and log anomaly detection methods PCA, LogCluster, DeepLog, LogAnomaly, LogBERT and LogTransfer are tested on two datasets and the results are analyzed. Then, the influence of different log vector representation methods on anomaly detection performance is analyzed, and the influence of domain adaptation on anomaly detection performance is analyzed.

4.1. Experiment Setting

4.1.1. Dataset

The experimental dataset comes from a large log dataset loghub[23] released by the intelligent operation and maintenance team of the Chinese University of Hong Kong. The total size of the maintained log data set is more than 77 GB, which is specially collected for log analysis and research. Some of the logs are production data released in previous studies, while others are real data collected

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 May 2023 doi:10.20944/preprints202305.1083.v1

from laboratory systems. The logs in the data set come from various types of systems, including HDFS, BGL, Thunderbird, OpenSSH, etc. This paper uses BGL and Thunderbird data sets for experiments:

- BlueGene/L (BGL) dataset, which comes from the BlueGene/L supercomputer system of Lawrence Livermore National Labs, has 4,747,963 logs in the BGL dataset, including 4,399,494 normal logs and 348,469 abnormal logs.
- The Thunderbird (TB) data set is collected from the Thunderbird supercomputer system of Sandia National Labs. There are 3,737,209 logs in the TB data set, including 3,015,004 normal logs and 722,205 abnormal logs. The statistics of the two data sets are shown in Table 1.

Table 1. Information of Dataset.

Dataset	Total	Normal	Abnormal
BGL	4,747,963	4,399,494	348,469
TB	3,737,209	3,015,004	722,205

Simple outlier detection does not need to group logs, and a single abnormal event is considered to be an independent anomaly. However, deep learning models are used to reveal abnormal models of multiple log events, such as changes in event sequences or time log correlations. Therefore, it is necessary to organize log events into groups and then analyze them individually or interrelated. There are two common grouping methods. One is the sliding time window, that is, a log will appear in multiple windows. The other is a fixed time window, that is, a log can only appear in one group. In order to ensure the similarity of the experimental environment, a sliding window of 20 is used to cut the log file into a short sequence with a step size of 4.

After sliding window segmentation, the BGL dataset is divided into 1,175,002 log sequences. When BGL is used as the source system, there are 60,000 normal log sequences for training, 1,014,762 normal log sequences for testing, and 100,240 abnormal log sequences for testing. When BGL is used as the target system, 1000 normal log sequences are used for training, 1,073,762 normal log sequences are used for testing, and 100,240 abnormal log sequences are used for testing. The Thunderbird dataset is divided into 1,126,852 log sequences. When Thunderbird is used as the target system, there are 1000 normal log sequences for training, 803,783 normal log sequences for testing, and 322,069 abnormal log sequences for testing. When Thunderbird is used as the source system, there are 60,000 normal logs for training, 744,783 normal log sequences for testing, and 322,069 abnormal log sequences for testing. As shown in Table 2.

Table 2. The number of log sequences after sliding window division.

Dataset	Normal log sequence	Abnormal log sequence
BGL	1,074,762	100,240
TB	804,783	322,069

In the training phase, 60,000 normal sequences from the source system and 1000 normal sequences from the target system are mixed as the model training set.

4.1.2. Evaluation Metrics

In order to measure the effectiveness of LogBD in anomaly detection, F1 value and AUC are used as evaluation indicators. The F1 value combines the Precision and Recall of the model. The higher the F1 value, the better the performance of the classification model. Precision represents the percentage of real anomalies in all detected anomalies, Recall represents the percentage of anomalies detected in the data set, and the F1 value is calculated as follows:

$$F_1 = 2\frac{Precision * Recall}{Precision + Recall} \tag{8}$$

The formulas of Precision and Recall are as follows :

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 May 2023 doi:10.20944/preprints202305.1083.v1

$$Precision = \frac{TP}{TP + FP} \tag{9}$$

$$Recall = \frac{TP}{TP + FN} \tag{10}$$

The AUC is the area under the ROC curve. The closer the area is to 1, the better the model effect is. It is mainly used to measure the performance of the binary classification model. The AUC calculation formula is as follows:

$$AUC = \frac{\sum_{ins_i \in positiveclass} rank_{ins_i} - \frac{M * (M+1)}{2}}{M * N} \tag{11}$$

Where $rank_{ins_i}$ is the ordinal number of the i th sample, M and N are the number of positive and negative samples, respectively, and $\Sigma_{ins_i \in positiveclass}$ is the sum of the ordinal numbers of all positive samples.

The value range of F1 value is 0 to 1. The higher the value, the better the performance of the classifier and the better the model effect, which can comprehensively describe the detection performance of the model. The value interval of AUC is 0.5 and 1. The higher the value, the better the performance of the model. AUC is not sensitive to abnormal threshold. When the distribution of abnormal log and normal log is unbalanced, AUC can measure the abnormal threshold well.

4.1.3. Baselines

LogBD is compared with the following log anomaly detection methods to evaluate the performance of LogBD:

- PCA, principal component analysis based on dimensionality reduction anomaly detection method, combined with PCA and term weighting technology in the field of data retrieval, constructed anomaly detection model ;
- logCluster, based on similarity clustering method, can automatically find similar log patterns and aggregate them into clusters to achieve automatic identification of anomalies.
- DeepLog, an anomaly detection model based on log template, uses LSTM to model log data, automatically learns abnormal behavior patterns, and compares predicted logs with real logs for anomaly detection.
- LogAnomaly integrates the log template by analyzing the synonyms in the log, encodes the log template into a semantic vector, and inputs it into the LSTM model for anomaly detection.
- LogBERT, a BERT-based method LogBERT, constructs two self-supervised training tasks to learn the pattern of normal log sequences and detect anomalies that deviate from the normal model.
- LogTransfer uses transfer learning to achieve cross-system anomaly detection, uses labeled normal and abnormal data of two systems to train the model, and shares a fully connected network that classifies logs.

4.1.4. Implementation

The model designed in this paper is built on the cloud server, The configuration used in the experiment is CPU Intel (R) Xeon (R) Platinum 8255C, GPU RTX 2080Ti, memory 43GB, Pytorch 1.7.1.

4.2. Results and Analysis

Since PCA, LogCluster, DeepLog, LogAnomaly and LogBERT are unsupervised models and are not designed for cross-system detection, these models are evaluated in two cases, that is, the training data set uses or does not use the samples of the target system, represented by W/O. Firstly, the Thunderbird dataset is used as the source system, and the BGL dataset is used as the target system. This situation is abbreviated as TB-BGL. The F1 value and AUC value of 0.880 are obtained on the

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 May 2023 doi:10.20944/preprints202305.1083.v1

source system Thunderbird, and the F1 value and AUC value of 0.938 and 0.973 are obtained on the target system BGL. Then the BGL data set is used as the source system and the Thunderbird data set is used as the target system. This case is abbreviated as BGL-TB. The F1 value of 0.933 and the AUC value of 0.978 are obtained on the source system BGL, and the F1 value of 0.841 and the AUC value of 0.854 are obtained on the target system Thunderbird.

As shown in Table 3 and Table 4, the table shows the experimental results of the six methods except LogTransfer in the case of TB-BGL and BGL-TB. W/O represents the training set using or not using the target system. Firstly, for PCA, LogCluster, DeepLog, LogAnomaly and LogBERT, they do not use domain adaptation. For Thunderbird as the source system, BGL as the target system (TB→BGL) or BGL as the source system, Thunderbird as the target system (BGL→TB), these five methods can take good F1 values and AUC values on the source system even if they do not use the sample training of the target system. When the training data set uses samples from the target system, they can obtain better F1 value and AUC value on the target system, but worse on the source system, indicating that these five methods do not have cross-system adaptive ability. Using a mixed log sequence of the source system and the target system will only make the training data distribution diverse, and the detection model confuses the distribution of the samples, resulting in poor model detection. Compared with these five methods, LogBD achieves better results in any scenario.

Table 3. The experimental results of six methods in the case of TB-BGL.

Method	Source		Target	
	F1	AUC	F1	AUC
PCA W/O	0.689/0.760	0.698/0.779	0.577/0.229	0.773/0.658
LogClusterW/O	0.708/0.724	0.688/0.716	0.597/0.223	0.686/0.500
DeepLog W/O	0.757/0.790	0.701/0.777	0.627/0.223	0.843/0.500
LogAnomaly W/O	0.793/0.813	0.754/0.791	0.697/0.287	0.859/0.647
LogBERT W/O	0.821/0.847	0.805/0.857	0.732/0.349	0.851/0.682
LogBD(ours)	**0.880**	**0.880**	**0.938**	**0.973**

Table 4. The experimental results of six methods in the case of BGL-TB.

Method	Source		Target	
	F1	AUC	F1	AUC
PCA W/O	0.322/0.642	0.587/0.816	0.731/0.558	0.776/0.504
LogClusterW/O	0.530/0.713	0.746/0.829	0.677/0.559	0.716/0.504
DeepLog W/O	0.834/0.878	0.824/0.867	0.720/0.656	0.779/0.600
LogAnomaly W/O	0.859/0.905	0.901/0.936	0.784/0.643	0.809/0.691
LogBERT W/O	0.882/0.926	0.915/0.954	0.809/0.709	0.828/0.733
LogBD(ours)	**0.933**	**0.978**	**0.841**	**0.854**

On the source system, the anomaly detection performance of LogBD is still better than these five, which fully shows that LogBD captures the pain points of log anomaly detection, that is, the accuracy of log template analysis, the use of log semantic information, and the method of anomaly detection. It can also be found that the performance of deep learning methods is better than that of machine learning methods, indicating that the machine learning model only uses the log template count vector as the input feature, without considering the log content itself, but it can still detect the abnormal information in the log to a certain extent, but it can not achieve good accuracy and coverage of anomaly detection. For example, PCA is based on the log template index for anomaly detection, only retains the main features of the original data, loses a lot of key information, and is difficult to learn features from the sparse count matrix. LogCluster is based on clustering for log anomaly detection, but it can not play a good role in the face of complex log structure, nor can it fully learn the features

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 May 2023 doi:10.20944/preprints202305.1083.v1

in the log, and the detection effect is not good. DeepLog regards the log sequence as a digital sequence and replaces the log template with a number. It not only uses the log parameter features but also integrates the log sequence features. However, it does not extract the semantic information in the log template, and it is easy to treat the log sequence that has not appeared in the training data as an exception, resulting in lower accuracy and more false alarms. Compared with the machine learning methods PCA and LogCluster, it has a more obvious improvement. The LogAnomaly method uses the semantic and grammatical information of the log template, and proposes Template2Vec for the synonyms in the log. It uses the word vector weighted average to obtain the vector representation, which has made some improvements on the basis of DeepLog. However, it does not consider the problem of polysemous words, only considers the representation of a single word vector, does not consider the context information, and the learned feature information is not comprehensive enough. LogBERT uses BERT to capture the pattern of normal log sequence, and uses two self-supervised task training models, mask log template prediction and hypersphere minimization. LogBERT uses the hypersphere objective function as LogBD, but the performance is not as good as LogBD, because LogBD uses domain adaptation to obtain more data with the same characteristics for training.

For LogTransfer, LogTransfer is a supervised transfer learning method that uses normal and abnormal labeled data of the source system and the target system to train a cross-system anomaly detection model. LogTransfer achieves good performance when sufficient tag data is available. In this experiment, we tested how many labeled abnormal samples are needed to train LogTransfer to make it have similar performance as LogBD. When using 100 abnormal sequences from the source system and 10 abnormal sequences from the target system to train LogTransfer, LogTransfer can achieve the best performance on the source system. The detection results in the two scenarios are shown in Table 5 and Table 6.

Table 5. The case of TB-BGL.

Method	Source		Target	
	F1	AUC	F1	AUC
LogTransfer	**0.981**	**0.975**	0.788	0.833
LogBD(ours)	0.880	0.880	**0.938**	**0.973**

Table 6. The case of BGL-TB.

Method	Source		Target	
	F1	AUC	F1	AUC
LogTransfer	**0.971**	0.972	0.792	0.828
LogBD(ours)	0.933	**0.978**	**0.841**	**0.854**

For the TB → BGL scenario, training LogTransfer with 10 abnormal sequences of the target system is not enough to be better than LogBD. It is better than LogBD on the source system, but worse on the target system. For the BGL-TB scenario, the performance is comparable on the source system and lower than LogBD on the target system. Therefore, LogBD can provide good performance using only normal data when it is difficult to obtain labeled abnormal samples.

Unlike previous methods, LogBD uses BERT to extract the semantic information of log messages, and uses semantic vectors to represent log templates, rather than the log templates id, Word2Vec and Glove used in previous methods. This paper compares the model performance using four log template representation methods. The detection results generated by different log template representation methods in two scenarios are shown in Table 7 and Table 8. It is found that the performance is greatly improved when BERT is used for log template representation. This may be because : the method based on the log template id is to number the log template and represent it by number. This method regards the log template as a number, and does not consider the semantic information contained in the log template. Word2Vec and Glove 's word vectors are fixed by looking up the dictionary and taking the corresponding word vectors. They cannot dynamically adjust the

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 May 2023 doi:10.20944/preprints202305.1083.v1

word vectors according to different context contexts, and lose the integrity of the log template semantics. According to the context of the input sentence, BERT returns sentence-level word vectors through model calculation in the model network. Due to the different context of the input context, the returned word vectors are different, so as to distinguish polysemous words. Compared with the first three, BERT can learn the deep semantics of sentences and capture the similarity between different log statements.

Table 7. Detection results generated by different log template representation methods in TB-BGL scene.

Method	Source		Target	
	F1	AUC	F1	AUC
Log Template ID	0.725	0.733	0.784	0.776
Word2Vec	0.788	0.797	0.845	0.909
Glove	0.832	0.857	0.897	0.918
BERT(ours)	**0.880**	**0.880**	**0.938**	**0.973**

Table 8. Detection results generated by different log template representation methods in BGL-TB scene.

Method	Source		Target	
	F1	AUC	F1	AUC
Log Template ID	0.781	0.799	0.679	0.691
Word2Vec	0.813	0.854	0.728	0.774
Glove	0.885	0.862	0.792	0.828
BERT(ours)	**0.933**	**0.978**	**0.841**	**0.854**

The cross-system log anomaly detection model LogBD proposed in this paper uses the domain adaptation method in transfer learning and achieves excellent performance. In order to prove the effectiveness of the domain adaptation method, a set of comparative experiments are carried out to compare the performance of the model without using the domain adaptation method and using the domain adaptation method. Observe the impact of domain adaptation methods on model performance.

The detection results using or not using the domain adaptation method in the two scenarios are shown in Tables 9 and 10. The without indicates that the domain adaptation method is not used, and the with indicates that the domain adaptation method is used. Through observation, it can be seen that the domain adaptive method can greatly improve the performance of the anomaly detection model, so that the anomaly detection model can learn the similarity between the two system log data, thereby detecting the anomalies of the two systems.

Table 9. TB-BGL scenario uses or does not use domain adaptation method.

Domain Adaptation	Source		Target	
	F1	AUC	F1	AUC
Without	0.794	0.807	0.467	0.418
With(ours)	**0.880**	**0.880**	**0.938**	**0.973**

Table 10. BGL-TB scenario uses or does not use domain adaptation method.

Domain Adaptation	Source		Target	
	F1	AUC	F1	AUC
Without	0.647	0.872	0.592	0.628
With(ours)	**0.933**	**0.978**	**0.841**	**0.854**

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 May 2023 doi:10.20944/preprints202305.1083.v1

This experiment also further evaluated the performance of LogBD when training with different numbers of target system normal log sequences. The experimental results are shown in Figure 9.

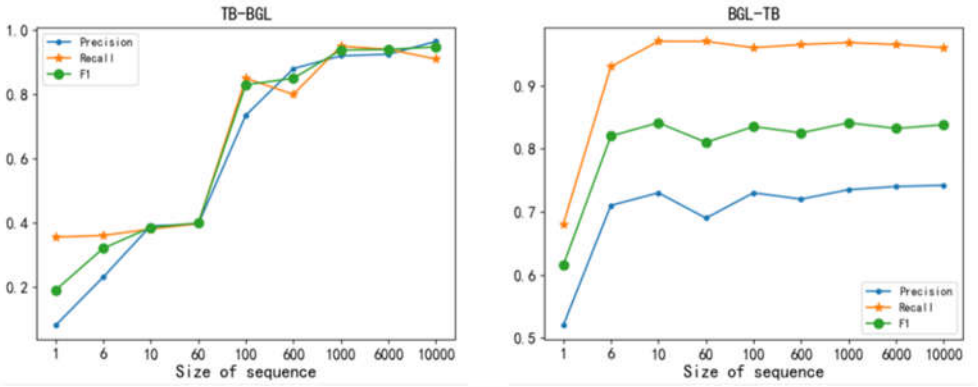

Figure 9. Left is TB-BGL scene, right is BGL-TB scene.

It can be observed that LogBD can achieve high anomaly detection performance for the target system by using a small number of normal sequences in the target system. In the TB → BGL scenario, using about 100 normal log sequences in BGL can achieve good performance. For the BGL → TB scenario, only using 10 normal log sequences in Thunderbird is enough to obtain good performance. When the number of log sequences increases, the performance will continue to improve. In general, even if the new online system deployment time is short, it is easy to obtain normal log data from the target system, so LogBD has strong feasibility and accuracy in detecting anomalies in the new system.

5. Conclusions

Aiming at the problems existing in the current log anomaly detection methods, this paper proposes a cross-system log anomaly detection method LogBD based on domain adaptation. This method establishes an adversarial training framework based on the adversarial domain adaptive network, uses the pre-training model BERT to extract the semantic information of the log statement, uses the TCN network to learn the order relationship of the log sequence, and maps the log sequence from the two systems to the hypersphere space, so that they are close to the center of a hypersphere sphere. Based on the distance between the log sequence and the center of the sphere, the anomaly is judged, and the distribution of the domain is confused. At the same time, a good anomaly detection effect is obtained. Finally, the related experiments of LogBD are designed and carried out. The experiments show that the method has achieved good results in cross-system log anomaly detection.

In this paper, the log data is divided into log sequences through the sliding window mechanism, only the log template sequence is used for anomaly detection, and the parameter information in the log message is not used. The next research direction is to establish new anomaly detection schemes for some important parameters, such as delay anomaly for log interval. Although the log anomaly detection model in this paper considers cross-system anomaly detection, it can only realize anomaly detection on two systems, and can not be deployed on multiple different systems. It is the next research direction to continue to study multi-system log anomaly detection in the field of domain adaptation.

Author Contributions: Conceptualization, L.D. and S.L.; methodology, L.D.; software, L.D.; validation, L.D., H.X. and W.W.; formal analysis, L.D.; investigation, L.D.; resources, L.D.; data curation, L.D.; writing—original draft preparation, L.D.; writing—review and editing, L.D.; visualization, L.D.; supervision, L.D.; project administration, S.L.; funding acquisition, S.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Nature Science Foundation of China, grant number 61762085 and the Autonomous Region Science and Technology Program of Xinjiang, grant 2020A02001-1.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are openly available at ref. [23].

Acknowledgments: We thank NSFC for funding this research. We thank the anonymous reviewers for their contribution to this paper.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

The appendix is an optional section that can contain details and data supplemental to the main text—for example, explanations of experimental details that would disrupt the flow of the main text but nonetheless remain crucial to understanding and reproducing the research shown; figures of replicates for experiments of which representative data is shown in the main text can be added here if brief, or as Supplementary data. Mathematical proofs of results not central to the paper can be added as an appendix.

Appendix B

All appendix sections must be cited in the main text. In the appendices, Figures, Tables, etc. should be labeled starting with "A"—e.g., Figure A1, Figure A2, etc.

References

References must be numbered in order of appearance in the text (including citations in tables and legends) and listed individually at the end of the manuscript. We recommend preparing the references with a bibliography software package, such as EndNote, ReferenceManager or Zotero to avoid typing mistakes and duplicated references. Include the digital object identifier (DOI) for all references where available.

Citations and references in the Supplementary Materials are permitted provided that they also appear in the reference list here.

In the text, reference numbers should be placed in square brackets [ ] and placed before the punctuation; for example [1], [1–3] or [1,3]. For embedded citations in the text with pagination, use both parentheses and brackets to indicate the reference number and page numbers; for example [5] (p. 10), or [6] (pp. 101–105).

1. Chen, Z.; Liu, J.; Gu, W.; et al. Experience report: Deep learning-based system log analysis for anomaly detection[J]. arXiv preprint arXiv:2107.05908, 2021.
2. The Complete History of AWS Outages. Available online: https://awsmaniac.com/aws-outages/ (accessed on 2 October 2022).
3. Wang, Q.; Zhang, X.; Wang, X.; et al. Log Sequence Anomaly Detection Method Based on Contrastive Adversarial Training and Dual Feature Extraction[J]. Entropy, 2021, 24(1): 69.
4. Liao, H.J.; Lin, C.H.R.; Lin, Y.C.; et al. Intrusion detection system: A comprehensive review[J]. Journal of Network and Computer Applications, 2013, 36(1): 16-24.
5. Garnter. Market Guide for AIOps Platforms. Available online: http://www.garnter.com/doc/3892967/market-guide-aiops-platforms.
6. Zhaoxue, J.; Tong, L.; Zhenguo, Z.; et al. A survey on log research of aiops: methods and trends[J]. Mobile Networks and Applications, 2021, 26(6): 2353-2364.
7. Du, M.; Li, F.; Spell: Online streaming parsing of large unstructured system logs[J]. IEEE Transactions on Knowledge and Data Engineering, 2018, 31(11): 2213-2227.
8. Zhang, S.; Meng, W.; Bu, J.; et al. Syslog processing for switch failure diagnosis and prediction in datacenter networks[C]. 2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS), 2017: 1-10.
9. He, P.; Zhu, J.; Zheng, Z.; et al. Drain: An online log parsing approach with fixed depth tree[C]. 2017 IEEE International Conference on Web Services (ICWS), 2017: 33-40.
10. Landauer, M.; Onder, S.; Skopik, F.; et al. Deep Learning for Anomaly Detection in Log Data: A Survey[J]. arXiv preprint arXiv:2207.03820, 2022.
11. Xu, W.; Huang, L.; Fox, A.; et al. Detecting large-scale system problems by mining console logs[C]. The ACM SIGOPS 22nd Symposium on Operating Systems Principles(SOSP), 2009: 117-132.
12. Chen, M.; Zheng, A.X.; Lloyd, J.; et al. Failure diagnosis using decision trees[C]. 1st International Conference on Autonomic Computing(ICAC), 2004: 36-43.
13. Meng, W.; Liu, Y.; Zhang, S.; et al. Logclass: Anomalous log identification and classification with partial labels[J]. IEEE Transactions on Network and Service Management, 2021, 18(2): 1870-1884.

14. Lou, J.G.; Fu, Q.; Yang, S.; et al. Mining Invariants from Console Logs for System Problem Detection[C]. 2010 USENIX Annual Technical Conference(ATC), 2010: 1-14.
15. Vaarandi, R.; Pihelgas, M.; Logcluster-a data clustering and pattern mining algorithm for event logs[C]. 2015 11th International Conference on Network and Service Management(CNSM), 2015: 1-7.
16. Du, M.; Li, F.; Zheng, G.; et al. Deeplog: Anomaly detection and diagnosis from system logs through deep learning[C]. The 2017 ACM SIGSAC Conference on Computer and Communications Security(ACM CCS), 2017: 1285-1298.
17. Meng, W.; Liu, Y.; Zhu, Y.; et al. LogAnomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs[C]. The Twenty-Eighth International Joint Conference on Artificial Intelligence(IJCAI), 2019, 19(7): 4739-4745.
18. Zhang, X.; Xu, Y.; Lin, Q.; et al. Robust log-based anomaly detection on unstable log data[C]. The 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering(ESEC/FSE), 2019: 807-817.
19. Xia, B.; Bai, Y.; Yin, J.; et al. Loggan: a log-level generative adversarial network for anomaly detection using permutation event modeling[J]. Information Systems Frontiers, 2021, 23: 285-298.
20. Chen, R.; Zhang, S.; Li, D.; et al. Logtransfer: Cross-system log anomaly detection for software systems with transfer learning[C]. 31st International Symposium on Software Reliability Engineering(ISSRE), 2020: 37-47.
21. Guo, H.; Yuan, S.; Wu, X.; Logbert: Log anomaly detection via bert[C]. 2021 International Joint Conference on Neural Networks(IJCNN), 2021: 1-8.
22. Zhang, C.; Wang, X.; Zhang, H.; et al. Log sequence anomaly detection based on local information extraction and globally sparse transformer model[J]. IEEE Transactions on Network and Service Management, 2021, 18(4): 4119-4133.
23. He, S.; Zhu, J.; He, P.; et al. Loghub: a large collection of system log datasets towards automated log analytics[J]. arXiv preprint arXiv:2008.06448, 2020.
24. Wang, Y.; Hou, Y.; Che, W.; et al. From static to dynamic word representations: a survey[J]. International Journal of Machine Learning and Cybernetics, 2020, 11: 1611-1630.
25. Mikolov, T.; Chen, K.; Corrado, G.; et al. Efficient estimation of word representations in vector space[J]. arXiv preprint arXiv:1301.3781, 2013.
26. Pennington, J.; Socher, R.; Manning, C.D.; Glove: Global vectors for word representation[C]. The 2014 Conference on Empirical Methods in Natural Language Processing(EMNLP), 2014: 1532-1543.
27. Sarzynska-Wawer, J.; Wawer, A.; Pawlak, A.; et al. Detecting formal thought disorder by deep contextualized word representations[J]. Psychiatry Research, 2021, 304: 114135.
28. Radford, A.; Narasimhan, K.; Salimans, T.; et al. Improving language understanding by generative pre-training[J]. 2018.
29. Devlin, J.; Chang, M.W.; Lee, K.; et al. Bert: Pre-training of deep bidirectional transformers for language understanding[J]. arXiv preprint arXiv:1810.04805, 2018.
30. Graves, A.; Schmidhuber, J.; Framewise phoneme classification with bidirectional LSTM and other neural network architectures[J]. Neural networks, 2005, 18(5-6): 602-610.
31. Vaswani, A.; Shazeer, N.; Parmar, N.; et al. Attention is all you need[J]. Advances in neural information processing systems, 2017, 30.
32. Han, X.; Yuan, S.; Unsupervised cross-system log anomaly detection via domain adaptation[C]. The 30th ACM International Conference on Information & Knowledge Management(CIKM), 2021: 3068-3072.
33. Ganin, Y.; Ustinova, E.; Ajakan, H.; et al. Domain-adversarial training of neural networks[J]. The journal of machine learning research, 2016, 17(1): 2096-2030.
34. Ruff, L.; Vandermeulen, R.; Goernitz, N.; et al. Deep one-class classification[C]. 10th International Conference on Machine Learning(ICML), 2018: 4393-4402.
35. Lea; Colin; et al. "Temporal convolutional networks: A unified approach to action segmentation." Computer Vision–ECCV 2016 Workshops: Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part III 14. Springer International Publishing, 2016.