ITC 3/53
Information Technology and Control
Vol. 53 / No. 3/ 2024
pp. 813-832
DOI 10.5755/j01.itc.53.3.35704

Multi-Scale Temporal Convolutional Networks and Multi-Head Attention for Robust Log Anomaly Detection

Received 2023/11/23 | Accepted after revision 2024/04/29

HOW TO CITE: Zhang, Z., Li, W., Wang, Y., Wang, Z., Sheng, X., Zhou, T. (2024). Multi-Scale Temporal Convolutional Networks and Multi-Head Attention for Robust Log Anomaly Detection. *Information Technology and Control,* 53(3), 813-832. https://doi.org/10.5755/j01.itc.53.3.35704

Multi-Scale Temporal Convolutional Networks and Multi-Head Attention for Robust Log Anomaly Detection

Zhigang Zhang, Wei Li
Tianjin Electric Power Trading Center Co, Tianjin 300010, China

Yizhe Wang
Beijing Electric Power Trading Center Co, Beijing 100000, China

Zhe Wang
School of Computer and Software, Nanjing University of Information Science and Technology, Nanjing 210044, China

Xiang Sheng, Tianxiang Zhou
Beijing Electric Power Trading Center Co, Beijing 100000, China

Corresponding author: 202212210005@nuist.edu.cn

System logs are instrumental in understanding computer system behavior and ensuring system stability and reliability, making anomaly detection in system logs crucial. However, with the increasing scale and complexity of modern software systems, log data is growing exponentially, rendering traditional manual log inspection methods inefficient. Moreover, the evolution of log messages over time results in a lower accuracy rate for anomaly detection. To address these issues, this paper proposes a log anomaly detection method based on multi-scale temporal convolution networks and multi-head attention. This method utilizes temporal convolution networks to extract temporal information from log data and extracts hidden features of logs through different receptive fields of multi-scale convolution kernels. By integrating the multi-head attention mechanism, the sequential dependencies of logs can be better captured. We conducted repeated experiments on the authoritative public HDFS and BGL log datasets to evaluate their detection accuracy and robustness. The experiments demonstrate that MTCNLog outperforms existing anomaly detection methods and is robust to the continuous evolution of logs.

KEYWORDS: Anomaly detection, System log, Log analysis, Deep learning, Neural networks.

1. Introduction

Ensuring high availability and reliability is of utmost importance for large-scale systems that are heavily reliant on software [15]. These systems are designed to serve a broad spectrum of users, providing a wide array of services. Even the slightest malfunction can lead to user dissatisfaction, which could potentially translate into significant economic losses. Hence, it is imperative for these systems to maintain continuous operation, typically on a 24x7 basis. The implementation of precise and efficient detection methods can mitigate system failures that are triggered by anomalous events. Consequently, anomaly detection emerges as a pivotal component in guaranteeing the quality assurance of complex, software-intensive systems. It is through such mechanisms that the robustness and reliability of these systems can be upheld, thereby ensuring seamless service delivery to the end-users.

Software-intensive systems are known to document runtime information through the generation of console logs. The scale and complexity of these systems often result in the production of substantial volumes of logs. These logs, which serve as a comprehensive record of the system's operational status, are extensively utilized for anomaly detection. Take, for example, Alibaba's cloud computing system, which generates an astounding 30-50 GB of trace logs every hour, equivalent to approximately 120-200 million lines. Each entry in a log file is essentially a semi-structured text message, originating from a log statement. This message comprises predefined event templates along with several dynamic parameters, offering a detailed insight into the system's functioning [9]. A log message can be deconstructed into a log event, with each log event embodying the template or the constant component of the message. Each log entry is timestamped, marking the time of its occurrence. As such, a log file can be viewed as a log sequence, comprising a vast number of log entries organized in the order of system execution. Log anomalies can be categorized into two types: individual log anomalies and log sequence anomalies. Individual log anomalies document abnormal states and are typically perceived as outliers. However, given the varied abnormal behaviors across different systems, detecting individual log anomalies can be challenging without prior knowledge of the target system's abnormal behavior. Our focus, therefore, shifts to log sequence anomalies, which allow us to identify abnormalities arising from an abnormal execution order or incomplete execution patterns within the log sequence. The automatic analysis of logs, aimed at identifying anomalous system behaviors, is becoming increasingly crucial in managing modern complex systems. Consequently, log anomaly detection has emerged as a vibrant area of research in both academic and industrial settings.

Traditional methods of log anomaly detection are often deemed inefficient [19]. These methods typically hinge on domain knowledge and utilize techniques such as regular expression matching or keyword searches to pinpoint anomalies. While these methods are capable of detecting anomalies, they require substantial time and financial investment. In the context of software-intensive systems, the application of traditional log anomaly detection methods is impractical. The scale and complexity of these systems necessitate a more efficient and cost-effective approach to anomaly detection. This highlights the need for innovative solutions that can effectively identify anomalies in log sequences, thereby enhancing the reliability and performance of these systems.

In recent years, a multitude of data-driven approaches have been introduced to automate the detection of system anomalies. A variety of log sequence anomaly detection methods have been devised using traditional machine learning techniques. These include logistic regression [8], support vector machines and principal component analysis. These techniques extract valuable features from log sequences and train binary classifiers, with or without labels, to identify system anomalies. While these traditional machine learning-based methods have demonstrated high accuracy, they fall short in capturing the temporal information embedded within log sequences due to manual feature extraction. To address this limitation, deep learning-based methods such as DeepLog [7], LogAnomaly [17], PLELog [26], LogRobust [27] and HitAnomaly [14] have been extensively employed for log sequence anomaly detection. Most of these methods leverage recurrent neural networks (RNNs) to capture comprehensive bidirectional contextual information. Despite the commendable performance of previous methods, both machine learning and deep learning-based, in log sequence anomaly detection, there remain challenges in modeling and analyzing log data.

Semantic Information: Current methods predominantly employ sequential vectors or quantitative vectors to represent log sequences. However, these approaches tend to overlook the semantic information inherent in logs, which can result in an inadequate capture of log features. The semantic content of logs, which provides valuable insights into the system's behavior, is thus often underutilized in anomaly detection.

Dependency: The detection of anomalies in log sequences relies heavily on both global and local dependencies between logs. Previous methods have often utilized Long Short-Term Memory networks to capture the dependency information within log sequences. However, due to the sequential nature of LSTM, it falls short in effectively capturing global dependencies. Moreover, its capacity to capture local dependencies is also somewhat limited due to the absence of interactive relationships.

Log Template Updating: The intricacy of software functions and the regular updates to business processes lead to more frequent alterations to log templates, thereby significantly increasing the diversity of log templates. This could disrupt the correlation between logs in the log sequence learned by the model, making it challenging to adapt to the constantly changing situations encountered in real-world scenarios.

To address the aforementioned challenges, we introduce a novel approach, MTCNLog, for handling log sequences. First, we employed the popular log parsing approach IPLoM [18] to extract log templates, and performed post-processing operations to mitigate the adverse effects of redundant templates, thereby facilitating subsequent feature extraction. Subsequently, log templates are represented using weighted semantic vectors based on Word2Vec, facilitating the capture of semantic information hidden within the log templates. Upon obtaining the semantic vector sequences, MTCNLog employs an improved TCN [4] based on multiple scales and a multi-head attention mechanism to learn hidden patterns within the log sequences and predict whether the log sequence is anomalous.

The proposed method encompasses multiple modules, including a multi-scale convolution module, dilated convolution, residual blocks, and a multi-head attention module. Due to the inherent limitations of a single Temporal Convolutional Network in extracting information across different time scales, the multi-scale feature extraction module in this study employs multiple convolutional kernels of varying sizes to capture diverse local information. This approach effectively circumvents the need for deep stacking of the model. Moreover, the TCN, when combined with a multi-head attention mechanism, can effectively integrate the contextual information of log statements. The attention mechanism assigns weights to more significant events, thereby facilitating more accurate and rapid detection. Past methods have commonly employed Long Short-Term Memory (LSTM) networks for detecting anomalies in log data. LSTM captures sequence information through recursive formulas. However, as information in LSTM can only flow from one time step to the next, such sequential networks struggle to effectively uncover dependencies between distant log entries in the log sequences. In addition, LSTM is unable to learn the intrinsic connection features of log sequences from a global perspective. In log anomaly detection tasks, both local correlations between adjacent log entries and long-range dependencies between distant log entries can greatly impact the final anomaly judgments. Therefore, methods relying solely on LSTM still have some limitations in log anomaly detection tasks. Compared to LSTM, our MTCNLog model exhibits superior capability in extracting long-term dependency features. Our method is particularly suited for handling system log data, which displays distinct temporal patterns and long-term dependencies, making it more effective than the LSTM model. The main contributions are summarized as follows:

1 By incorporating the post-processing step into IPLoM, we have effectively addressed the issue of insufficient accuracy in existing log templates. This enhancement not only facilitates subsequent feature extraction but also lays a solid foundation for the efficacy of log anomaly detection.

2 We propose a deep learning model for log anomaly detection that fully exploits the strengths of multi-scale feature extraction modules, temporal convolutional networks, and multi-headed attention mechanisms for handling time-series data. The multi-scale feature extraction module enhances the model's learning capacity via a multi-reso-

lution receptive field. TCN can effectively distill hidden information and temporal relationships within features while eliminating redundant traits. Multi-headed attention captures complex global dependencies of log anomalies and spotlight critical information. By automatically learning salient features and uncovering latent sequential dependencies, the model achieves performant and flexible anomaly detection.

3 We have conducted numerous comparative and robustness experiments on two authoritative log datasets, HDFS and BGL, to validate the performance of the MTCNLog model in system log anomaly detection. The experimental results demonstrate that MTCNLog exhibits commendable accuracy and robustness in system log anomaly detection.

The rest structure of this paper is outlined as follows: the related work is introduced in Section 2. Section 3 describes the proposed log anomaly detection method MTCNLog, including the overall detection framework, log processing, and anomaly detection model. Finally, in Section 4, we conducted experiments to evaluate the performance of the proposed model, and Section 5 summarizes our work in this paper.

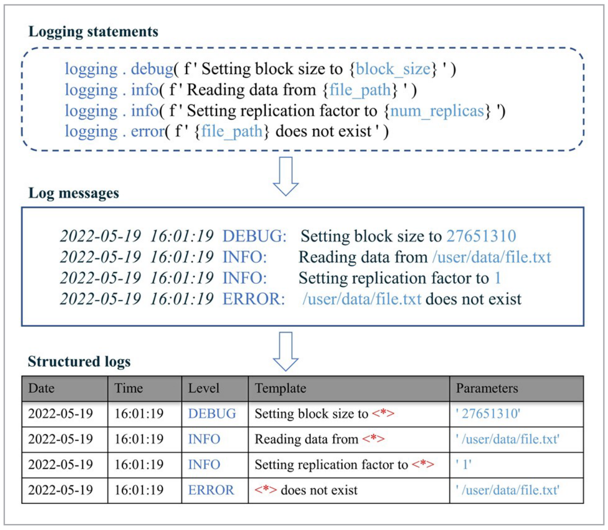
2. Related Work

The process of log anomaly detection involves three crucial stages: log parsing, feature extraction, and anomaly detection. Each stage has its specific methods and tools, and there exists a close relationship between them. In the following, we will review the related work for each stage.

2.1. Log Parsing

Log parsing is typically the initial step in log anomaly detection frameworks [1]. It acts as a preprocessing tool, and its accuracy has a significant influence on the outcomes of log anomaly detection. Log parsing involves the conversion of semi-structured log data into structured data [2]. Logs are unstructured and comprise both fixed and variable components. The goal of log parsing is to segregate the constant and variable parts in the raw logs, thereby extracting uniformly formatted log templates. An illustration of log parsing is depicted in Figure 1.

Figure 1
Overview of log parsing



```
logging . debug( f' Setting block size to {block_size} ' )
logging . info( f' Reading data from {file_path} ' )
logging . info( f' Setting replication factor to {num_replicas} ')
logging . error( f' {file_path} does not exist ' )
```

```
2022-05-19  16:01:19  DEBUG:  Setting block size to 27651310
2022-05-19  16:01:19  INFO:   Reading data from /user/data/file.txt
2022-05-19  16:01:19  INFO:   Setting replication factor to 1
2022-05-19  16:01:19  ERROR:  /user/data/file.txt does not exist
```

Date	Time	Level	Template	Parameters
2022-05-19	16:01:19	DEBUG	Setting block size to <*>	'27651310'
2022-05-19	16:01:19	INFO	Reading data from <*>	'/user/data/file.txt'
2022-05-19	16:01:19	INFO	Setting replication factor to <*>	'1'
2022-05-19	16:01:19	ERROR	<*> does not exist	'/user/data/file.txt'

Log parsing solutions primarily fall into three categories: clustering-based approaches, frequent pattern mining-based approaches, and heuristic approaches. Some log parsers leverage traditional clustering algorithms. For instance, LogMine [10] is a representative clustering-based solution that utilizes a single-pass clustering algorithm to scan all log messages, thereby generating dense clusters.

Certain log parsers make use of frequent pattern mining, a well-established technique in data mining, to uncover patterns that go beyond a set support threshold. LogCluster uses hash tables to pinpoint words that occur frequently. It extracts all frequent words from each log message to form or update potential groups.

A number of log parsers utilize a variety of heuristic algorithms for log template extraction. Drain [11] is a representative heuristic approach that groups original log messages using a parsing tree of fixed depth, with each group converging at leaf nodes. Predefined filtering rules guide the search for suitable leaf nodes. Spell is another widely used log parsing method that is based on the longest common subsequence. It processes log messages in a streaming manner into structured templates and variables. IPLoM parses logs in three steps: it scans and partitions logs by length, divides partitions using token frequency and bijection, and extracts templates where unique tokens are constants and others are variables. In this study, IPLoM was selected as the log parsing method, transforming semi-structured log data into structured log events and generating log templates. To enhance the accuracy of log parsing, post-processing operations were also applied.

2.2. Feature Extraction

The subsequent stage in the log anomaly detection process involves feature extraction from the logs, which necessitates representing the logs in a format that complies with the requirements of the anomaly detection model.

Log representation techniques take semi-structured raw log data or parsed log data as input and generate representations at different levels of abstraction [20]. Specifically, it is necessary to extract relevant features from log templates to input them into deep learning models for detection. The advancement of Natural Language Processing (NLP) techniques has provided methodologies for feature representation on log templates in this context. Currently, there are three main representation formats: sequential vectors, quantitative vectors, and semantic vectors. Sequential vectors reflect the order of log events in a window. For example, DeepLog indexes each log event and subsequently creates sequential vectors for corresponding log windows. Quantitative vectors, similar to log count vectors, are used to capture the occurrences of each log event in a log window. Semantic vectors, on the other hand, are used to capture the meaning of each log event in a log window. Semantic vectors are generated by applying NLP techniques such as word embedding or topic modeling on log templates. LogAnomaly utilizes sequential vectors and quantitative vectors to detect anomalies.

In log anomaly detection, semantic vectors from language models are used to capture log event semantics. These vectors provide a dual-layered representation: token-level and event-level. A log message is essentially a token string, each represented as embeddings via a pre-trained language model like Word2Vec. Logsy [19], for example, labels preprocessed log messages and generates token embeddings from the log template. These representations, along with token position encoding, are input into a transformer-based architecture, refining token-level representations through training. Event-level embeddings, vector representations encoding individual log templates, are obtained by aggregating token-level embeddings using various pooling methods. LogRobust employs a pre-trained FastText model to compute semantic vectors for log events. Moreover, certain language models are capable of directly generating such representations. For example, Swisslog [16] uses a pre-trained BERT as the sentence encoder to directly produce sequence-level embeddings for log templates. In our research, we have adopted a weighted vector representation method that combines Word2Vec and TF-IWF to extract significant features from log templates.

2.3. Anomaly Detection

The task of detecting anomalies in logs can be viewed as a binary classification problem, where the goal is to classify a log sequence as either normal or anomalous. Methods for log anomaly detection that have been proposed recently can be broadly divided into two categories: those based on data mining techniques and those that utilize deep learning approaches.

Log anomaly detection methods, based on data mining, are broadly classified into supervised and unsupervised learning techniques. Supervised techniques use labeled log data to train models, learning fixed patterns corresponding to different log labels. Common models include support vector machines, finite state automatons [6], decision trees [5], logistic regression [20], among others. These methods typically achieve higher detection accuracy but require labor-intensive and costly manual labeling. Their performance varies across different datasets and applications. A key limitation is their inability to effectively model sequential dependencies and semantic contexts among log events. For instance, PCA and clustering treat each log message independently, ignoring their occurrence order. Pattern mining learns correlation but lacks generalization. To overcome these limitations, recent research focuses on deep learning techniques, capable of capturing complex sequential and structural log data information.

Indeed, deep learning has garnered considerable attention over the past decade due to its superior capabilities in model representation and performance. Numerous studies have leveraged deep learning for the detection of anomalies in log sequences.

Brown et al. [3] introduced an RNN model that incorporates an attention mechanism for the detection of anomalous patterns in system logs. Zhang et al., on the other hand, proposed a unique anomaly detection model for the automatic analysis of console logs. This model leverages the Long Short-Term Memory network to capture the sequential characteristics of log sequences, marking the first application of the LSTM model in the field of log anomaly detection. This has led to numerous studies exploring the use of LSTM models and their variants for log anomaly detection. OC4Seq utilizes a multi-scale RNN framework that takes into account the imbalanced nature of log data. This allows for the capture of different levels of sequence patterns by embedding discrete event sequences into a latent space, facilitating relatively straightforward anomaly detection. Compared to LSTM and RNN, the Temporal Convolutional Network is a novel method for time series prediction that can capture long-term dependencies. He et al. [12] utilized a TCN model, trained on normal sequences, to forecast trends across a series of time steps. They then fitted the errors from these predictions to a Gaussian distribution to determine if the sequence was anomalous.

Despite the impressive achievements of many methods in log data anomaly detection, the increasing quantity and complexity of log data in software-intensive systems have made it more challenging to detect anomalies in real log sequences. In this study, we primarily utilized the TCN model, multi-head attention mechanism, and multi-scale convolution architecture to develop a deep learning-based system log anomaly detection model. This model can effectively handle longer log sequences, automatically learn the importance of various log event sequences, discover hidden dependencies in the sequences, and improve the accuracy of anomaly detection.

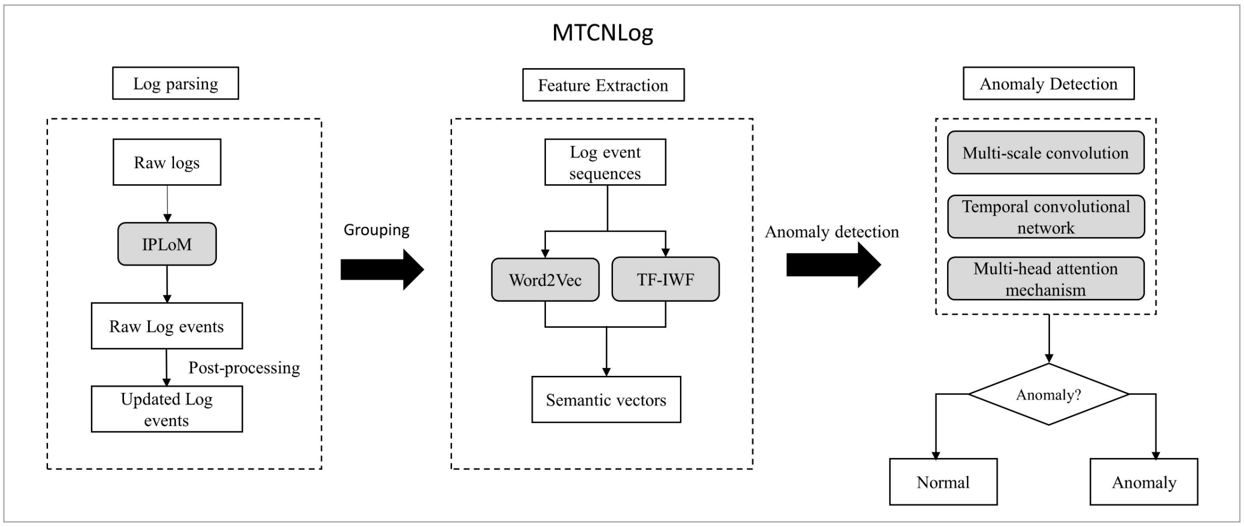
3. Methodology

In this section, we provide a detailed description of the proposed MTCNLog method. We begin by introducing the framework of MTCNLog. After presenting the framework, we propose an improved step for IPLoM, which parses raw logs into log templates. Following this, we employ language models in conjunction with TF-IWF weighting to represent log templates as semantic vectors. Finally, we utilize a TCN model that integrates a multi-head attention mechanism and a multi-scale feature extraction module to learn the long-term dependencies of log sequences and carry out log anomaly detection. This approach allows us to effectively capture the intricate patterns within log data and enhance the accuracy of anomaly detection.

3.1. MTCNLog Framework

Our MTCNLog framework aims to leverage the semantic relationships in log sequences for better log processing. Moreover, by automatically learning the correlation of various log event sequences, the framework uncovers hidden dependencies within the sequences, thereby improving the accuracy of log anomaly detection. The MTCNLog framework is illustrated in Figure 2.

Figure 2
The framework of MTCNLog



3.2. Log Parser (IPLoM)

Effective log analysis begins with log parsing, transforming unstructured raw log messages into structured log events. Raw logs contain runtime information like timestamps and verbosity levels. For instance, in the raw log "Jun 22 04:11:42 com-bo pam_unix 17037 session closed for user news", the log header is "Jun 22 04:11:42 combo supam_unix 17037", and the content is "session closed for user news". Log parsers aim to convert raw messages into structured log templates by preserving constant words and replacing variable words with wildcards (e.g., "<*>"). This process enhances the efficiency and effectiveness of log analysis.

Transforming dynamic, unstructured logs into a structured format can improve anomaly detection accuracy. Traditional log parsing, performed manually by developers using regular expressions, relies heavily on expert knowledge and experience but lacks active learning capabilities. Recently, more efficient log parsers like IPLoM have been proposed. The IPLoM algorithm is a log data clustering algorithm that operates by iteratively partitioning a set of log messages. While IPLoM is currently a competent log parsing tool, parsing errors can still occur. If IPLoM is used directly to construct log templates for anomaly detection, achieving optimal detection results can be challenging. This is because IPLoM sometimes parses many additional log events, which impedes the performance of the anomaly detection model detection. As shown in Table 1, the HDFS log templates parsed by the IPLoM method resulted in 42 different types of HDFS log templates. However, some log templates were found to be repetitive. For instance, the four log event templates in {E9,E10,E11} are similar, and E9 may encompass the other two templates. Directly converting these templates into semantic vectors could significantly impact the reliability of log anomaly detection. In this study, to enhance the accuracy of log parsing, we made certain improvements to the IPLoM method. We incorporated a post-processing operation into the IPLoM method. After IPLoM extracts the templates, we further optimize the log parsing effect to achieve a higher level of detection.

To enhance the accuracy of parsing, it is crucial to merge redundant templates that are similar and eliminate invalid templates through the post-processing operations of IPLoM. Specifically, IPLoM will examine the log event set one by one to identify which event templates originate from the same event. Since the variables have been replaced by "<*>" in the previous processing, we first replace consecutive wildcards "<*>" with a single wildcard "<*>". Next, we select some representative log entries that match each log template and check for duplication. Based on the frequency of the log templates, if duplication is found, the final event template is extracted using the longest common subsequence algorithm. The post-process-

Table 1
Log templates obtained by parsing HDFS logs through the IPLoM

Event Id	Log Template
E1	Receiving block<*>src:<*>dest:<*>
E2	BLOCK* NameSystem.allocateBlock: <*> <*>
...	...
E9	writeBlock <*> received exception <*>
E10	writeBlock <*> received exception java.io.IOException
E11	writeBlock <*> received exception java.io.InterruptedIOException
...	...
E21	Exception in receiveBlock for block <*> <*>
E22	Exception in receiveBlock for block <*> java.io.IOException:
...	...
E42	Deleting block <*> file <*>

Table 2
Updated HDFS log templates

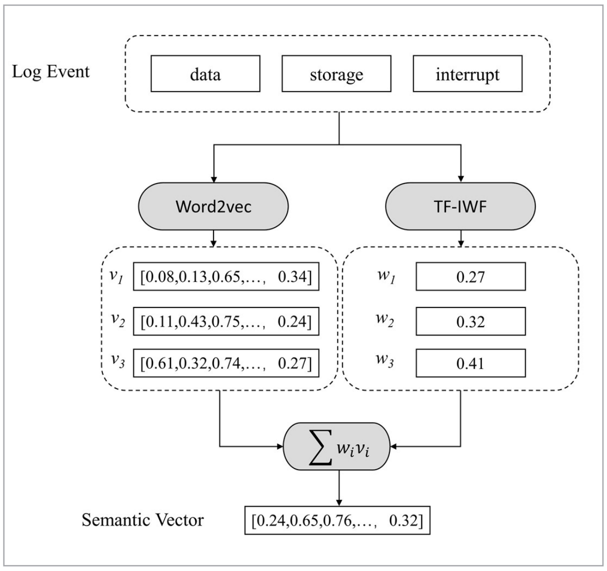
EventId	Log Template
E1	Receiving block<*>src:<*>dest:<*>
...	...
E6	Changing block file offset of block<*>from<*>to<*>meta file offset to<*>
...	...
E18	Unexpected error trying to delete block<*>BlockInfo not found in volumeMap<*>
...	...
E30	Deleting block<*>file<*>

ing results of the log events are displayed in Table 2. This approach ensures a more accurate and reliable log parsing process, thereby improving the overall effectiveness of log anomaly detection.

3.3. Log Represent

The second stage of the log analysis process involves feature extraction. After log parsing, MTCNLog uses a combination of Word2Vec and TF-IWF to convert each log event into a semantic vector, as shown in Figure 3. Word2Vec, an unsupervised language model, learns semantic knowledge from extensive text corpora. Understanding log semantics is vital for log anomaly detection, with word comprehension closely tied to preceding and succeeding semantics. Word2Vec abstracts the next word's occurrence as a dependency on previous words, expressing this dependency and the sentence's deep meaning in vector form. Methods like vector weighted average yield the entire sentence's vector form, enhancing sentence expressiveness. Thus, we use Word2Vec to vectorize logs, accurately representing log events and boosting subsequent analysis effectiveness.

Figure 3
The process of semantic vectorization of log event



After obtaining word embedding vectors, each log event is treated as a natural language sentence. Given the semantic features of log events, words contribute differently to overall semantics. A simple average aggregation method may not accurately reflect these features. MTCNLog uses TF-IWF weighted aggregation, a common weighting technique in information retrieval and exploration, to represent the relationship between each word and log event. TF-IWF, an enhancement of the term frequency-inverse document frequency algorithm, measures word importance in a sentence. The TF term frequency matrix measures a word's association with a given text, while IWF measures the word's importance. We thus use the Term Frequency (TF) to measure the importance of word, where $TF(token) = \frac{\#token}{\#event}$, #token is the number of target word in a log event, #event is the number of all words in a log event. If the word "Exception" appears in all log events, it becomes too common and cannot distinguish these log events, so its weight should be reduced. Therefore, it is also necessary to utilize inverse document frequency (IWF) as a metric, where $\mathrm{IWF}(\mathrm{token}) = \log\left(\frac{\#\mathrm{L}}{\#\mathrm{T}}\right)$, #L represents the total number of tokens for all log events, and #T refers to the total number of times the token appears in all log events. For each word, its TF-IWF weight w is calculated through TF×IWF. Finally, by summing the word vectors in the list with their corresponding TF-IWF weights, we can obtain a semantic vector representing a specific log event, as shown in Equation (1).

$$V = \frac{1}{N}\sum_{i=1}^{N} w_i \cdot v_i . \quad (1)$$

Indeed, MTCNLog effectively utilizes the semantic features of log events. The use of semantic vectors allows it to identify log events with similar semantics and distinguish between different log events.

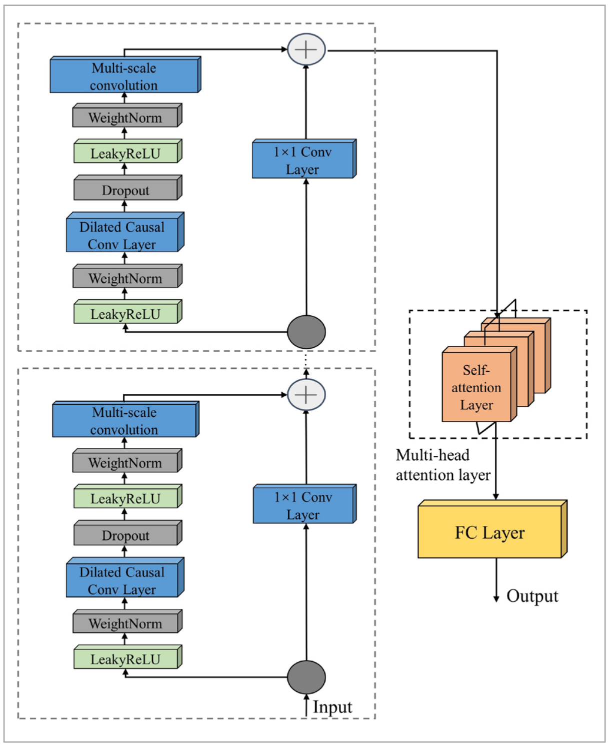
3.4. Log Anomaly Detection

Logs are a unique type of natural language, the semantics and temporal aspects of which warrant thorough exploration and analysis. Therefore, we segment log events into log event sequences. These sequences are then transformed into vectors of semantic vectors, derived from feature extraction, representing the log sequences. These log sequence vectors are subsequently input into the anomaly detection model. We have developed an anomaly detection model, MTCN-Log, which performs log anomaly detection based on an enhanced TCN that combines a multi-scale feature extraction module and a multi-head self-attention mechanism. This approach allows us to effectively capture the intricate patterns within log data and enhance the accuracy of anomaly detection.

3.4.1. Overall Structure of Detection Model

The anomaly detection model in MTCNLog is based on a pure multi-scale convolutional structure and Temporal Convolutional Network model, and introduces a multi-head attention mechanism into the anomaly detection model to enhance its ability to accurately detect anomalies. This model leverages the sparse connections of the convolutional structure to effectively extract features without requiring a large amount of computational resources. Its design combines the advantages of a multi-scale convolutional structure, TCN model, and multi-head attention mechanism. The multi-scale convolutional structure enables the model to capture the relationships between hierarchical log events, the TCN model can effectively capture the temporal dependencies in log data, and the multi-head attention can pay attention to dependencies at different positions in the log sequence simultaneously. This comprehensive modeling of interactions between logs is conducive to learning the semantic information of logs. This mechanism computes potential dependencies between logs and learns from them, enabling the model to capture intricate relationships within log data. By focusing on different aspects of the log simultaneously, this model can effectively detect anomalies that might otherwise be overlooked. The overall structure of the model is depicted in Figure 4.

Figure 4
The composition of the anomaly detection model



3.4.2. TCN

In the fields of computer vision and natural language processing, Convolutional Neural Networks have demonstrated excellent performance in feature learning. Subsequently, methods such as Long Short-Term Memory networks have emerged for learning from time series data. However, the effectiveness of CNN is limited by its receptive field.

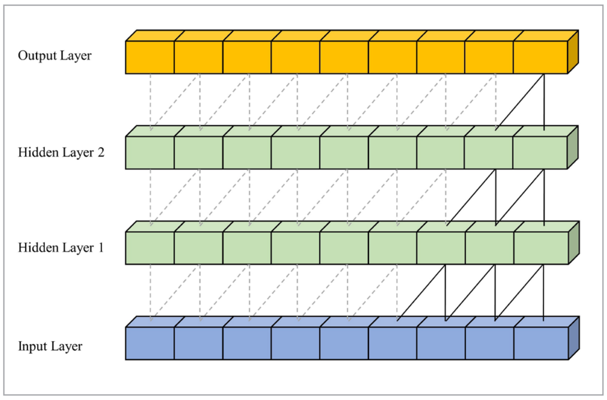
Therefore, Bai et al. [4] proposed a special type of convolutional neural network—Temporal Convolutional Network for sequence modeling tasks. TCN improves upon the basic CNN model and exhibits superior performance in handling time series problems. TCN can effectively analyze the relationships between data, with more stable gradients and higher computational efficiency. TCN is mainly composed of dilated convolution, causal convolution, and residual connections.

3.4.3. Casual Convolution

The causal convolution, as depicted in Figure 5, is a crucial component of the Temporal Convolutional Network structure. A system is deemed causal if its output is solely dependent on current and past inputs, and is independent of future inputs. In a similar vein, a convolution is considered causal if the output at a given time is influenced only by the current and previous inputs, and not by future inputs. Unlike the typical bidirectional structure, the causal convolution has a unidirectional structure, where the current output is determined by the current and past inputs. This characteristic of causal convolution addresses the issues of time leakage and inconsistent input-output lengths in sequence feature extraction. Specifically, for an input X = (x_0, x_1, x_2,...,x_t,...,x_T), the corresponding output yt is related to the current input xt and a period (x_{t-1}, x_{t-2},...,x_{t-k}) without the introduction of future inputs (x_{t+1}, x_{t+2},...,x_{t+T}). This ensures that the model's predictions are based solely on relevant and chronologically appropriate data, enhancing the accuracy and reliability of the anomaly detection process.

Figure 5

Structure of the causal convolution



3.4.4. Dilated Convolution

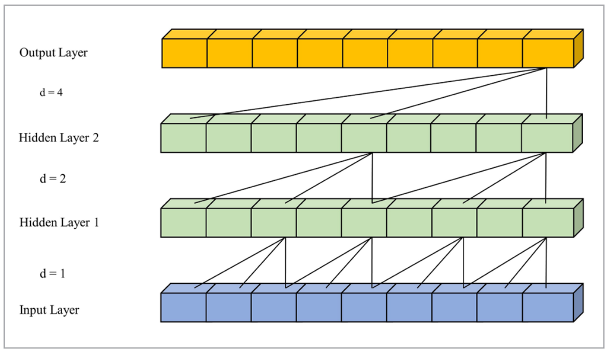
Traditional CNNs use standard convolution kernels, with small ones offering low complexity but potentially missing data variations, and large ones providing a large receptive field but high computational load. Temporal Convolutional Networks address this by using dilated convolution, expanding the receptive field without significantly increasing complexity. The dilation factor in dilated convolution influences the input interval's sampling range during convolution. The combination of causal convolution and dilated convolution, as depicted in Figure 6, can enhance the receptive field of the convolution layer and obtain more information based on strict time constraints. For the sequence input X = (x_0, x_1, x_2,...,x_t ,...,x_T) and the filter W = (w_0, w_1, w_2,...,w_n), the dilated convolution at x_t ($1 \le t \le T$) with dilation rate equal to d is defined as:

$$(W^{*}X)_{(x_t)} = \sum_{n=1}^{N} w_n x_t - (N-n)d\,. \tag{2}$$

Due to the linear increase of the dilation factor with network depth, the receptive field of the output layer can be effectively expanded. As a result, the entire model can extract information from longer.

Figure 6

Structure of the dilated causal convolution

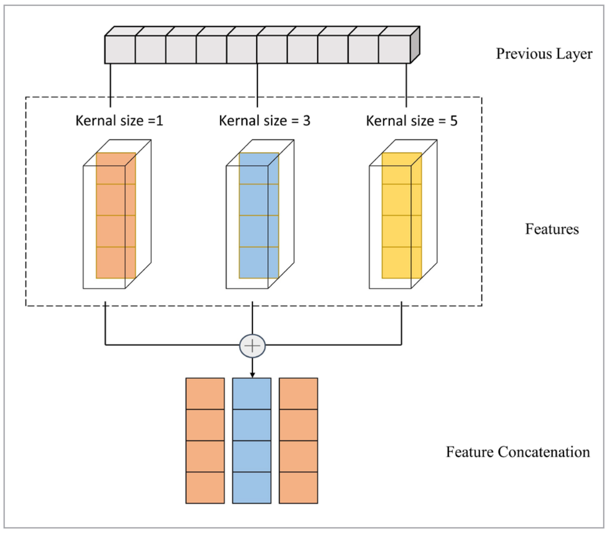


3.4.5. Multi-scale Feature Extraction

It is well established that simply increasing the depth of convolutional neural networks is the most straightforward approach to improve their performance. However, deeper models also run the risk of overfitting and lead to substantially higher computational costs. To mitigate this issue, the proposed MTCNLog incorporates a multi-scale feature extraction module based on a fully convolutional structure. Through sparse connections, this module enables extracting local features at different granularities from the log sequences. Specifically, it comprises three parallel causal convolution layers with kernels of varying sizes - the large-scale kernels aim at capturing long-term dependencies between logs while the small-scale kernels target short-term relationships. By focusing the distinct convolution kernels onto temporal patterns across different scales, the features can be deeply fused to output more enriched representations. Without markedly increasing complexity, such a configuration expands the number of feature channels and facilitates more comprehensive processing of multi-timescale sequences for extracting highly abstract characteristics. Compared to a single causal convolution kernel, the multi-scale module significantly empowers the model with the capacity to gather features and patterns across different ranges of the log events. The extraction flexibility of a solitary kernel is confined by its receptive field size. By contrast, the proposed module fully exploits multi-grained information in the sequence via the joint use of varying receptive fields, thereby enhancing the overall expression ability and generalization of the model. Consequently, the final model can learn a greater diversity of temporal features from the log data and achieve improved anomaly detection performance. The architecture of the multi-scale feature extraction module is illustrated in Figure 7.

Figure 7

Structure of the Multi-scale feature extraction module

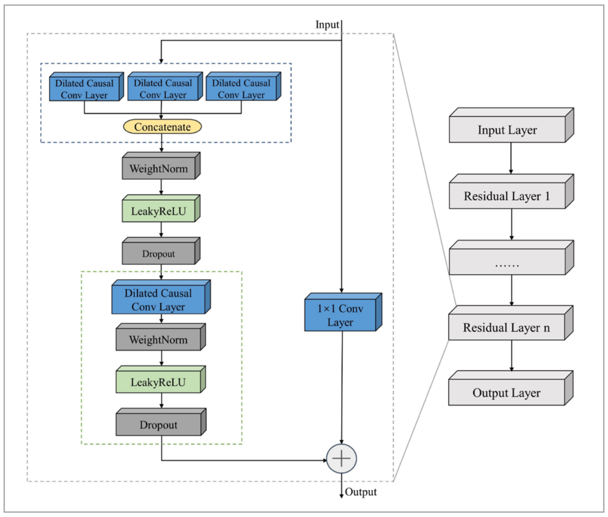


3.4.6. Residual Connection

TCN adopts residual structures [13] that can mitigate the vanishing and exploding gradient issues when training deep neural networks. Specifically, residuals enable preserving the training dynamics of models during iterative optimization. In TCN, two residual blocks are chained to construct the overall architecture. As illustrated in Figure 8, each constituent block contains a causal dilated convolution layer, weight normalization, activation functions, and dropout, connected in a channel-wise manner. The residual connection facilitates gradient propagation through the depth of the network. By doing so, TCN is able to incorporate the temporal convolutions in a very deep topology for effectively capturing long-term dependencies in sequential data.

Figure 8

Structure of the residual connection block



By utilizing diluted convolutions and Multi-scale feature extraction structures, MTCNLog can attain a broader receptive field compared to vanilla TCNs, enabling it to capture longer-range dependencies in log sequences. Meanwhile, stacking multiple layers is imperative for MTCNLog to achieve compelling detection accuracy through learning highly abstract input representations over depth. Addressing performance degradation from vanishing or exploding gradients, TCN constructs a deep feature extractor using chained residual blocks. Each block contains a causal dilated convolution channel, weight normalization, activation functions, and dropout. The multi-scale feature extraction module and dilated causal convolution use causal dilated convolution as a fundamental unit. A supplementary 1×1 convolution branch matches dimensions for element-wise addition. Weight normalization regularizes the parameter space, enhancing training speed, while dropout mitigates overfitting associated with deep topologies.

The classic TCN model uses the ReLU activation function for data transformation. However, outliers can cause neuron "death" and abnormal network parameter updates. Negative values, forced to 0, significantly impact sequential feature extraction in log sequences. The ReLU function is depicted in Figure 9(a), with its mathematical expression in Equation (3).

$$ReLU(x) = \{\begin{matrix} x, & x > 0 \\ 0, & x \leqslant 0 \end{matrix}. \tag{3}$$

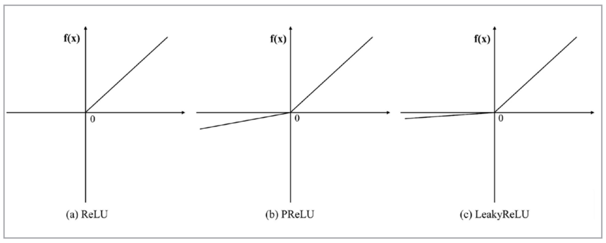
To address this issue, we propose the parametric rectified linear unit and Leaky ReLU as alternatives to the ReLU function. These activation functions have parameters in the negative region, preventing the forcing of all negative values to zero and effectively avoiding "dead" neurons. The function graphs of PReLU and LeakyReLU are shown in Figure 9(b) and Figure 9(c), and their mathematical expressions are given by Equations (4)-(5), respectively.

$$\mathrm{PReLU}(x_i) = \{\begin{matrix} x_i, & x > 0 \\ \lambda x_i, & x \leqslant 0 \end{matrix}, \tag{4}$$

$$\mathrm{Leaky} \cdot \mathrm{ReLU}(x_i) = \{\begin{matrix} x_i, & x > 0 \\ \mathrm{a}x_i, & x \leqslant 0 \end{matrix}. \tag{5}$$

Figure 9

The function graph of ReLU, PreLU and LeakyReLU



The difference between the two lies in whether the parameters are learnable. Among them, λ is a learnable parameter, while a is a fixed value. Although each PReLU only needs to learn a small number of parameters, each residual structure after the improvement contains 3 activation functions. If we improve the activation function to PReLU, after multiple layers of network transformation, a large number of parameters need to be learned. Therefore, we choose LeakyReLU as the activation function in the residual structure.

3.4.7. Multi-head Attention Mechanism

Attention models optimize parameters by leveraging data correlations, enhancing model accuracy. Initially applied in natural language processing, attention matches queries with key-value pairs to generate weighted outputs. The query, keys, values, and output are all d-dimensional vectors. The weight for each value is determined by the dot product between the query and the corresponding key. However, large differences in dot products can lead to vanishing gradients when applying softmax normalization. To mitigate this, a scaling factor is introduced before dot products for rescaling. The innovation of attention mechanisms is multi-head attention, conducting multiple independent attention calculations. Each head focuses on a different feature subspace, allowing the model to integrate information from different granularities. To address the issue of sequential models like TCN losing precedence information over long sequences, multi-head attention is used to mine long-range correlations in data more effectively. Specifically, the model constructs a multi-head computational unit based on scaled dot product attention, as illustrated in Figure 10(a). The calculations are as follows:

$$Attention(Q,K,V) = Softmax\left(\frac{QK^T}{\sqrt{d_K}}\right)V, \tag{6}$$

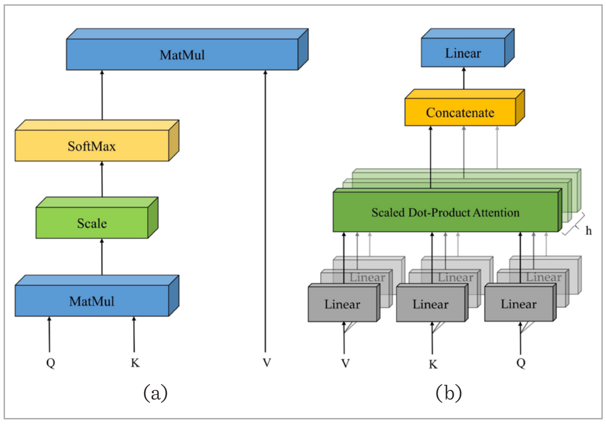
where *Q, K,* and *V* denote the query, key and value matrices separately; d_K refers to the dimension of *K*. After computing query-key similarities as weights, they are divided by d_K to prevent overlarge dot products. As shown in Figure 10(b), attention from multiple heads is generated in parallel and concatenated as the final output, formulated as:

Multibead(Q,K,V)=Concat(head_1,...,$\mathrm{head}_b$$W^0$,

where W_i^Q, W_i^K, W_i^V are all projection matrices. Q, K, and *V* first undergo linear transformations and then undergo point-wise multi-head attention h-scrunching, which can be defined as multi-head attention. Finally, the results of h scaled dot-product attentions

Figure 10

Details of the self-attention mechanism



are concatenated and another linear transformation is applied to obtain the output of multi-head attention.

The integration of multi-head attention, temporal convolutional networks, and a multi-scale feature extractor forms a robust anomaly detection model. This model enhances anomaly detection by highlighting key parts of log sequences and revealing hidden inter-log dependencies. Multi-head attention facilitates learning cross-log relevance, TCN use layered causal dilated convolutions to collect temporal contextual signals, and the multi-scale module identifies correlated patterns across different ranges. These components work together to effectively extract diagnostic features sensitive to abnormalities, covering both local and global log interactions.

4. Experiment

In this section, we will first outline the experimental dataset and evaluation metrics. Following this, we will compare the performance of MTCNLog on large-scale system log data with existing methods. We will also examine the impact of key parameters in the model on its performance. Finally, we will verify the effectiveness of each module of MTCNLog and its robustness to unseen logs.

4.1. Dataset

To assess the performance of MTCNLog, we utilized the HDFS and BGL log datasets, which are two widely recognized datasets for log anomaly detection.

The HDFS dataset is a public benchmark dataset frequently used for log-based anomaly detection. It comprises 11,175,629 log messages generated from over 200 Amazon EC2 nodes, of which 288,250 messages are abnormal, accounting for less than 2.6%.

The BlueGene/L supercomputer system at Lawrence Livermore National Laboratory (LLNL) gave rise to BGL, which comprises 4,747,963 log messages, including 348,460 logs identified as anomalies. The specific information is presented in Table 3.

Table 3

Summary of the datasets

System	Time	logs	Anomalis	Templates
HDFS	2 days	11,175,629	16,838	30
BGL	215 days	4,747,963	348,460	378

4.2. Baselines

We compared our approach with six distinct methods, including unsupervised ones such as PCA and Deeplog, semi-supervised LogAnomaly, and supervised methods like LogRobust and LightLog.

PCA [31]: Principal Component Analysis is used for statistical analysis on log sequences, extracting an event count matrix. The count matrix is then decomposed using singular value decomposition by PCA to obtain ordered features.

LightLog [23]: Lightlog is a lightweight TCN deployed on edge devices for the supervised classification of temporal log data. It alleviates the substantial computational load of the downstream processing pipeline during both the training and detection phases.

Deeplog [7]: DeepLog, a trailblazing framework for system log anomaly detection, uses advanced deep learning techniques, specifically recurrent neural networks, to model the sequential dependencies in logs and identify anomalies.

LogRobust [27]: LogRobust vectorizes words, computes their TF-IDF, and derives the log template's semantic vector by summing the weights with the TF-IDF.

LogAnomaly [17]: LogAnomaly identifies anomalies by using a log template count vector and learning normal log sequence patterns.

4.3. Evaluation Metrics

To evaluate the effectiveness of the proposed model in anomaly detection, we use Precision, Recall, and F1-score as evaluation metrics. These metrics are defined as follows:

Precision is the percentage of anomalies that are correctly detected among all the detected anomalies by the model. The formula is as follows:

$$Precision = \frac{TP}{TP+FP}. \tag{7}$$

Recall refers to the percentage of anomalies that are correctly detected by the model among all the anomalies. The formula is as follows:

$$Recall = \frac{TP}{TP+FN}. \tag{8}$$

The F1-score is equivalent to the weighted average of the comprehensive evaluation metrics Precision and Recall. The formula is as follows:

$$F1-score = \frac{2\times Precision\times Recall}{Precision+Recall}. \tag{9}$$

4.4. Experiment Setting

This experiment was developed using Python 3.7, and the deep learning framework PyTorch 1.10.2 was used to construct the model. It was run on a Windows 10 64bit environment with an Intel(R) Core(TM) i5-9400F processor and 16GB memory, and the GeForce RTX 2060 GPU was used to accelerate model training.

In our experiments, we fine-tuned the parameters of our deep neural network model on each dataset. Some parameters remained constant across datasets, while others needed unique adjustments. To avoid overfitting, we trained the predictive model on the training set and used early stopping on the validation set. We recorded the parameter configurations that produced the best results. Specifically, we set the length of the log sequence m = 40, the size of the sliding window was 100. We set the maximum training epoch = 150, the learning rate lr = 0.001, the number of LR reduction iteration rounds lr_step = (30, 40), the LR reduction rate lr_decay_ratio = 0.1, dropout was 0.5, the batch size batch_size (HDFS) = 24, and the batch size batch_size (BGL) = 36. Other hyperparameters were set as follows: TCN input channels were 300, output channels were 200, and the number of TCN residual units was 3. Multi-scale convolution kernel sizes were [1, 3, 5], multi-head attention heads were 8.

4.5. Results Analysis

In our approach, we employ an enhanced IPLoM for log template extraction. However, there are numerous alternative methods for log template extraction available. Therefore, it's essential to validate the effectiveness of our method through experiments and assess the impact of the log parser on anomaly detection performance. In these experiments, we applied representative log parsing methods currently available, including Drain, Spell, Logram, IPLoM and Lenma. The experimental results are presented in Table 4.

The aforementioned log parsers exhibit high-performance metrics on both the HDFS and BGL datasets, suggesting that the impact of the log parser on the performance of deep learning-based log anomaly detection is relatively minor. While Drain and Spell have the highest Precision metrics on HDFS and BGL, respectively, our method's Precision is slightly lower. However, our method achieves optimal levels for both Recall and F1-score, demonstrating its effectiveness. When considering overall performance, our method outperforms the others, further validating its efficacy.

Furthermore, we evaluated MTCNLog against five other anomaly detection techniques: PCA, LightLog, Deeplog, LogRobust, and LogAnomaly. The assessment was conducted on a variety of datasets to gauge the model's performance in log anomaly detection. For unsupervised techniques like PCA and DeepLog, only normal log sequences from the training dataset were utilized to construct the anomaly detection model. As depicted in Figure 11, it is evident from the Recall that, compared to traditional PCA, the recall rate on the HDFS dataset has increased by 30.65%, and the recall rate on BGL has increased by 28.82%. This suggests that the MTCNLog model exhibits a high recall rate on both the HDFS and BGL datasets, thereby enhancing its ability to accurately detect anomalies. The precision of the model further underscores its effectiveness in detecting abnormal log data. As can be seen from the F1-score, our method achieved an impressive F1-score of 98.65% and 98.34% on the HDFS and BGL datasets, respectively, surpassing all other methods.

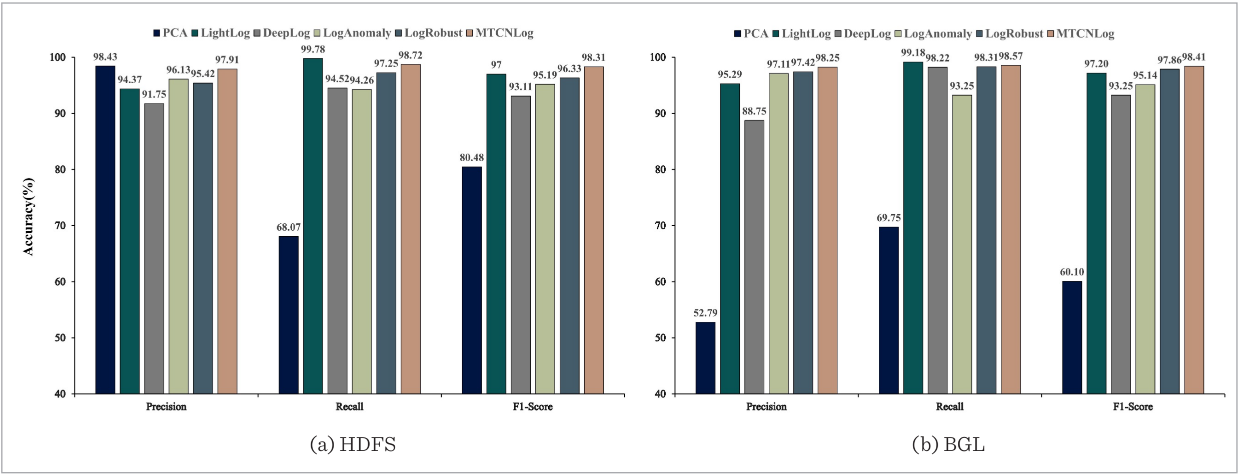
Table 4

Impact of Log Parser on Anomaly Detection Performance

Parser	HDFS(%)			BGL(%)		
	P	R	F1	P	R	F1
Drain	**97.94**	97.12	97.53	97.68	98.31	97.99
Spell	96.76	95.11	95.93	**98.31**	98.45	98.38
IPLoM	95.31	94.22	94.76	96.71	97.35	97.03
Logram	96.32	96.78	96.55	98.31	95.42	96.84
Lenma	96.87	94.10	95.46	98.11	97.87	97.99
Ours	97.91	**98.72**	**98.31**	98.25	**98.57**	**98.41**

Figure 11

Comparison of different methods. (a) HDFS dataset. (b) BGL dataset

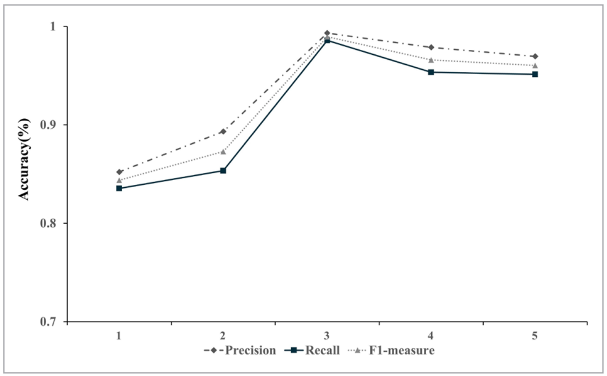


Indeed, both PCA and DeepLog employ the log template's index for log representation, which may overlook the semantic information embedded in the templates. Furthermore, during the anomaly detection process, DeepLog independently models and identifies anomalies in execution path types and parameter types, potentially neglecting the interrelationships among various anomalies. Notably, compared to LogRobust, which is known for its robust detection performance, our method's F1-score increased by 1.4% and 0.99%, respectively. We also found that supervised methods for anomaly prediction significantly surpassed unsupervised methods in terms of precision, recall, and F1-score, leading us to adopt this approach for optimal anomaly detection rates. Interestingly, the unsupervised DeepLog also demonstrated commendable performance. However, further exploration is required to significantly enhance the detection performance of unsupervised anomaly detection methods to meet practical needs. MTCNLog integrates Word2vec and TF-IWF to better extract semantic information from logs, improving anomaly detection accuracy. Temporal convolutional networks equipped with multi-scale feature extraction modules and multi-headed attention perform log anomaly detection. High recall and F1-score demonstrate the efficacy of this model for detecting anomalies in log sequences. Our model exhibits robust performance across datasets, precisely identifying abnormalities in both the HDFS and BGL logs. The amalgamation of semantic-aware embeddings and multi-faceted temporal modeling enables precise, stable detection of anomalies regardless of log source.

To assess the effect of the number of branches in the multi-scale feature extraction structure, we began with a single causal convolution branch (L=1) and incrementally increased the number of convolution branches L to 1, 2, 3, 4, and 5. Figure 12 illustrates the experimental results of MTCNLog with varying branch numbers. As depicted in Figure 12, MTCN-Log maintains relatively consistent accuracy across different branch numbers, but the F1-score exhibits significant fluctuations. Despite the increase in branch number, MTCNLog's anomaly detection efficiency does not significantly decline and remains at a high level. However, excessively small or large branch numbers result in marginally lower recall rates and F1-scores, causing the model to miss some anomalies. In conclusion, MTCNLog exhibits optimal performance when the branch number is 3.

Figure 12

Effect of Multi-scale structure branch number on Precision, Recall, and F1-score

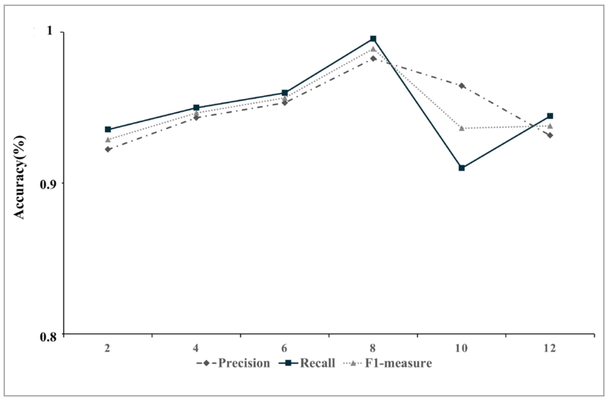


Furthermore, our experiment examined the impact of the number of branches in the multi-scale feature extraction structure and the number of multi-head attention layers on the model's performance. We conducted tests on the HDFS dataset, altering one parameter value while maintaining the default values for the others. To observe its influence on MTCNLog, we took two as the step size and set the number of heads to values within [2,12]. Figure 13 shows its influence on MTCN-Log, from which we can find that when the number of heads tends to eight, precision, recall, and F1-score all tend to stabilize. Therefore, in this experiment, we set the number of heads in self-attention to eight.

Since we introduced the multi-head attention mechanism, the number of heads in self-attention cannot be ignored. To observe its influence on MTCNLog, we took two as the step size and set the number of heads to values within [2,12]. Figure 13 shows its influence on MTCNLog, from which we can find that when the number of heads tends to eight, precision, recall, and F1-score all tend to stabilize. Therefore, in this experiment, we set the number of heads in self-attention to eight.

Figure 13

Effect of Heads of self-attention on Precision, Recall, and F1-score

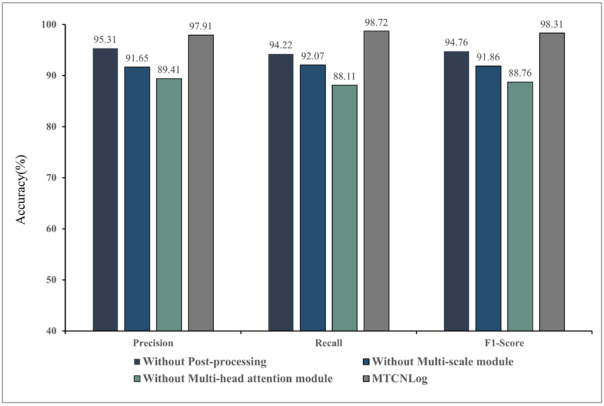


4.6. Ablation Study

We conducted ablation studies on the HDFS dataset to assess the impact of each module on the experimental outcomes, as illustrated in Figure 14. The results show that the multi-scale structure enhances the model's detection performance, validating the effectiveness of incorporating the multi-scale convolution structure for log sequence processing. The introduction of

Figure 14

Results of ablation study



the multi-head attention mechanism resulted in an 11 percentage point increase in the model's F1-score. This indicates that the multi-head attention module can autonomously learn the significance of different log sequences and extract higher-level hidden features within the sequence, thereby significantly boosting the accuracy of anomaly detection. Moreover, we attempted to replace the TCN with a 1D-CNN in the multi-scale feature extraction layer. The results once again demonstrated that this module plays a significant role in anomaly detection. This is due to the fact that TCNs are more sensitive to sequence order and can capture dependencies over longer distances, whereas 1D-CNNs typically only capture local features. These ablation experiments confirm that every module contribute to the model's performance.

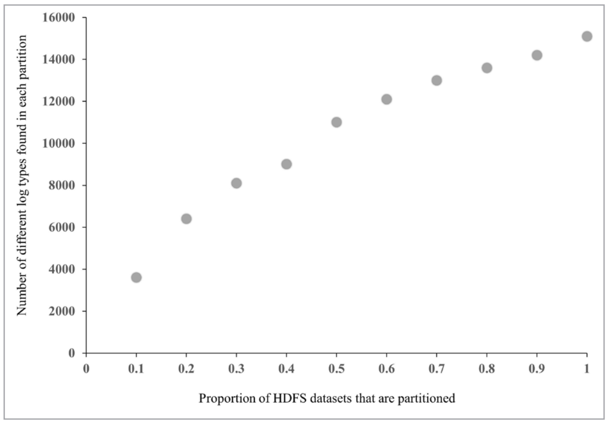
4.7. Robustness and Efficiency

In this section, we evaluate the robustness of the model based on unseen log types. We compare the final representations of different block_id log sequences to assess the log types. Absolutely, given the massive volume of system log data, processing time indeed becomes a critical factor. In our experiments, we also evaluated the time consumption of our model across different log datasets, highlighting the efficiency of our approach.

In our research, we analyzed the distribution of log template quantities in the log dataset to examine MTCNLog's detection effect on new types of logs. As depicted in Figure 15, due to the inherent instability of log data, an increase in the number of logs can introduce new log templates. To address this, MTCNLog learns the semantic features of logs to identify whether new log templates are anomalous. This approach allows for more robust and adaptive anomaly detection, even in the face of evolving log data.

Figure 15

Distribution of log types for randomly partitioned HDFS datasets



It is noteworthy that new log templates often contain some out-of-vocabulary (OOV) words, which makes it difficult to extract the semantic information of the log templates. To reduce the OOV words, we employ the classic subword segmentation method in natural language processing to divide an OOV word into several subwords that have appeared before. For instance, the OOV long word "allocateBlock" is split into "allocate" and "Block", thereby reducing the number of OOV words and enhancing the effect of semantic feature extraction. To further assess the model's effectiveness on new types of logs, we trained the model on varying amounts of log sequences on the HDFS. Table 5 provides a summary of the proportion of new log templates and their detection results. The model's performance improves as the training ratio increases. Remarkably, even when the proportion of new templates is as high as 60%, the F1-score still achieves 96.55%. This can be attributed to the semantic representation of normal log events. Changes in the events can still be represented as vectors similar to the original events, while vectors of abnormal events and normal events differ significantly. Consequently, MTCNLog exhibits robustness to variations in log events.

In summary, Table 6 shows the time consumption of different log anomaly detection methods on various datasets. As can be seen, PCA is based on cluster computation and does not involve complex neural network weight calculations. LightLog is a lightweight network deployed on edge devices that reduces the number of model parameters through global average pooling, thus their training and testing times are relatively short. Among other methods involving neural network models, our proposed MTCNLog method has shorter training and testing times than LogAnomaly and Deeplog, but slightly higher compared to methods like LogRobust. This is due to the introduction of a pre-trained model, which increases the model's input from 1-dimensional to 300-dimensional, leading to an increase in model parameters, and the introduction of a multi-head attention mechanism results in more parameters than LSTM. In conclusion, although our method outperforms traditional methods and deep learning

Table 5
Evaluation results of a new type of log for HDFS dataset

Training log sequences	Num of log templates	Percentage of new templates (%)	Precision	Recall	F1-score
4000	13	56.7	0.9454	0.9631	0.9542
5000	14	53.3	0.9631	0.9645	0.9638
6000	15	50	0.9714	0.9687	0.9701
7000	16	46.7	0.9814	0.9832	0.9823
8000	17	43.3	0.9825	0.9844	0.9834

Table 6
Time consumption of different methods on log datasets

Method	HDFS		BGL	
	Training	Testing	Training	Testing
PCA [31]	27 min	1.5 s	13 s	1 s
DeepLog [7]	2 h 25 min	42 min	1 h 6 min	10.5 min
LogAnomaly [17]	2 h 40 min	50 min	2 h 20 min	24 min
LogRobust [27]	1h 13 min	12 min	51 min	7 min
LightLog [23]	29 min	2 min	16 min	1.5 min
MTCNLog (ours)	1 h 45 min	23 min	1 h 30 min	18 min

methods like DeepLog in terms of metric evaluation, it is slightly inferior to them in terms of log processing timeliness. Therefore, MTCNLog is suitable for tasks that require high accuracy in log detection and low time consumption. The slightly larger time consumption of MTCNLog is a shortcoming that we will focus on researching and improving in the future.

5. Conclusions

Anomaly detection plays a pivotal role in ensuring system reliability. In this paper, we introduce MTCNLog, a novel deep learning-based framework for log-based anomaly event detection. This framework enhances the accuracy of log parsing by incorporating post-processing operations, resulting in more precise log templates. Feature extraction is performed using the Word2Vec and TF-IWF weighting algorithms, yielding weighted sentence embedding vectors. These vectors leverage semantic information to generate more effective feature representations from log event sequences. The final anomaly detection model is constructed based on multi-head attention mechanisms, multi-scale convolution, and temporal convolution networks. This model simultaneously learns local features and long-distance dependency features of logs, enabling it to handle new types of log templates. We evaluated MTCNLog on the authoritative HDFS dataset. The results validate the effectiveness of MTCNLog and confirm the utility of the TCN model, multi-scale convolution module, and multi-head attention mechanism for log-based anomaly event detection. This paper primarily focuses on semantic feature extraction from individual log statements. However, in practical production, operators often analyze faults based on multiple logs. Therefore, future work will explore the semantic feature representation of multiple types of logs. As the scale of system logs and the number of log templates continue to grow, we will also investigate more efficient semantic extraction methods to reduce the execution time and memory computation costs of log anomaly detection.

Acknowledgement

This research was funded by grants from State Grid Corporation Science and Technology Program (SGSXDK00DJJS2250061).

References

1. Bai, Y., Chi, Y., Zhao, D. PatCluster: A Top-Down Log Parsing Method Based on Frequent Words. IEEE Access 2023, 11, 8275-8282. https://doi.org/10.1109/ACCESS.2023.3239012
2. Bin Lashram, A., Hsairi, L., Al Ahmadi, H. HCLPars: A New Hierarchical Clustering Log Parsing Method. Engineering, Technology & Applied Science Research, 2023, 13, 11130-11138. https://doi.org/10.48084/etasr.6013
3. Brown, A., Tuor, A., Hutchinson, B., Nichols, N. Recurrent Neural Network Attention Mechanisms for Interpretable System Log Anomaly Detection. In Proceedings of the First Workshop on Machine Learning for Computing Systems, Tempe AZ USA, June 12 2018, 1-8. https://doi.org/10.1145/3217871.3217872
4. Bai, S., Kolter, J. Z., Koltun, V. Bai, S., Kolter, J. Z., Koltun, V. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. arXiv preprint, 2018. arXiv:1803.01271.
5. Chen, M., Zheng, A. X., Lloyd, J., Jordan, M. I., Brewer, E. Failure Diagnosis Using Decision Trees. In Proceedings of the International Conference on Autonomic Computing, New York, NY, USA, 2004, 36-43. https://doi.org/10.1109/ICAC.2004.1301345
6. Debnath, B., Solaimani, M., Gulzar, M. A. G., Arora, N., Lumezanu, C., Xu, J., Zong, B., Zhang, H., Jiang, G., Khan, L. LogLens: A Real-Time Log Analysis System. In Proceedings of the 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), Vienna, 2018, 1052-1062. https://doi.org/10.1109/ICDCS.2018.00105
7. Du, M., Li, F., Zheng, G., Srikumar, V. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas Texas USA, 2017, 1285-1298. https://doi.org/10.1145/3133956.3134015
8. Farshchi, M., Schneider, J.-G., Weber, I., Grundy, J. Experience Report: Anomaly Detection of Cloud Application Operations Using Log and Cloud Metric Correlation Analysis. In Proceedings of the 2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE), Gaithersbury, MD, USA, 2015, 24-34. https://doi.org/10.1109/ISSRE.2015.7381796
9. He, S., He, P., Chen, Z., Yang, T., Su, Y., Lyu, M. R. A Survey on Automated Log Analysis for Reliability Engineering. ACM Computing Surveys, 2022, 54, 1-37. https://doi.org/10.1145/3460345
10. Hamooni, H., Debnath, B., Xu, J., Zhang, H., Jiang, G., Mueen, A. LogMine: Fast Pattern Recognition for Log Analytics. In Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, Indianapolis Indiana USA, 2016, 1573-1582. https://doi.org/10.1145/2983323.2983358
11. He, P., Zhu, J., Zheng, Z., Lyu, M. R. Drain: An Online Log Parsing Approach with Fixed Depth Tree. In Proceedings of the 2017 IEEE International Conference on Web Services (ICWS), 2017, 33-40. https://doi.org/10.1109/ICWS.2017.13
12. He, Y., Zhao, J. Temporal Convolutional Networks for Anomaly Detection in Time Series. Journal of Physics: Conference Series, 2019, 1213, 042050. https://doi.org/10.1088/1742-6596/1213/4/042050
13. He, K., Zhang, X., Ren, S., Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, 770-778. https://doi.org/10.1109/CVPR.2016.90
14. Huang, S., Liu, Y., Fung, C., He, R., Zhao, Y., Yang, H., Luan, Z. HitAnomaly: Hierarchical Transformers for Anomaly Detection in System Log. IEEE Transactions on Network and Service Management, 2020, 17, 2064-2076. https://doi.org/10.1109/TNSM.2020.3034647
15. Le, V.-H., Zhang, H. Log-Based Anomaly Detection with Deep Learning: How Far Are We? In Proceedings of the 44th International Conference on Software Engineering, 2022, 1356-1367. https://doi.org/10.1145/3510003.3510155
16. Li, X., Chen, P., Jing, L., He, Z., Yu, G. SwissLog: Robust Anomaly Detection and Localization for Interleaved Unstructured Logs. IEEE Transactions on Dependable Secure Computing, 2023, 20, 2762-2780. https://doi.org/10.1109/TDSC.2022.3162857
17. Meng, W., Liu, Y., Zhu, Y., Zhang, S., Pei, D., Liu, Y., Chen, Y., Zhang, R., Tao, S., Sun, P., Zhou, R. LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs. In Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, Macao, China, August 2019, 4739-4745. https://doi.org/10.24963/ijcai.2019/658
18. Makanju, A., Zincir-Heywood, A.N., Milios, E.E. A Lightweight Algorithm for Message Type Extraction in System Application Logs. IEEE IEEE Transactions on Knowledge and Data Engineering, 2012, 24, 1921-1936. https://doi.org/10.1109/TKDE.2011.138

19. Nedelkoski, S., Bogatinovski, J., Acker, A., Cardoso, J., Kao, O. Self-Attentive Classification-Based Anomaly Detection in Unstructured Logs. In Proceedings of the 2020 IEEE International Conference on Data Mining (ICDM), Sorrento, Italy, 2020, 1196-1201. https://doi.org/10.1109/ICDM50108.2020.00148

20. Qin, T., Gao, Y., Wei, L., Liu, Z., Wang, C. Potential Threats Mining Methods Based on Correlation Analysis of Multi-type Logs. IET Networks, 2018, 7, 299-305. https://doi.org/10.1049/iet-net.2017.0188

21. Vaarandi, R., Pihelgas, M. LogCluster - A Data Clustering and Pattern Mining Algorithm for Event Logs. In Proceedings of the 2015 11th International Conference on Network and Service Management (CNSM), Barcelona, Spain, 2015, 1-7. https://doi.org/10.1109/CNSM.2015.7367331

22. Wang, Z., Chen, Z., Ni, J., Liu, H., Chen, H., Tang, J. Multi-Scale One-Class Recurrent Neural Networks for Discrete Event Sequence Anomaly Detection. In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, Virtual Event Singapore, 2021, 3726-3734. https://doi.org/10.1145/3447548.3467125

23. Wang, Z., Tian, J., Fang, H., Chen, L., Qin, J. LightLog: A Lightweight Temporal Convolutional Network for Log Anomaly Detection on the Edge. Computer Networks 2022, 203, 108616. https://doi.org/10.1016/j.comnet.2021.108616

24. Wang, R., Li, J. Bayes Test of Precision, Recall, and F1 Measure for Comparison of Two Natural Language Processing Models. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Florence, Italy, 2019, 4135-4145. https://doi.org/10.18653/v1/P19-1405

25. Xu, W., Huang, L., Fox, A., Patterson, D., Jordan, M. I. Detecting Large-Scale System Problems by Mining Console Logs. In Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, Big Sky Montana USA, 2009, 117-132. https://doi.org/10.1145/1629575.1629587

26. Yang, L., Chen, J., Wang, Z., Wang, W., Jiang, J., Dong, X., Zhang, W. PLELog: Semi-Supervised Log-Based Anomaly Detection via Probabilistic Label Estimation. In Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), Madrid, ES, 2021, 230-231. https://doi.org/10.1109/ICSE-Companion52605.2021.00106

27. Zhang, X., Xu, Y., Lin, Q., Qiao, B., Zhang, H., Dang, Y., Xie, C., Yang, X., Cheng, Q., Li, Z., Chen, J., He, X., Yao, R., Lou, J.-G., Chintalapati, M., Shen, F., Zhang, D. Robust Log-Based Anomaly Detection on Unstable Log Data. In Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Tallinn Estonia, 2019, 807-817. https://doi.org/10.1145/3338906.3338931