Computer Modeling in Engineering & Sciences

Tech Science Press

DOI: 10.32604/cmes.2023.025774

check for updates

ARTICLE

Logformer: Cascaded Transformer for System Log Anomaly Detection

Feilu Hang[1], Wei Guo[1], Hexiong Chen[1], Linjiang Xie[1], Chenghao Zhou[2,*] and Yao Liu[2]

[1]Information Center, Yunnan Power Grid Company Limited, Kunming, 650034, China

[2]Network and Data Security Key Laboratory of Sichuan Province, University of Electronic Science and Technology of China, Chengdu, 610054, China

*Corresponding Author: Chenghao Zhou. Email: 202121090103@std.uestc.edu.cn

Received: 29 July 2022 Accepted: 14 September 2022

ABSTRACT

Modern large-scale enterprise systems produce large volumes of logs that record detailed system runtime status and key events at key points. These logs are valuable for analyzing performance issues and understanding the status of the system. Anomaly detection plays an important role in service management and system maintenance, and guarantees the reliability and security of online systems. Logs are universal semi-structured data, which causes difficulties for traditional manual detection and pattern-matching algorithms. While some deep learning algorithms utilize neural networks to detect anomalies, these approaches have an over-reliance on manually designed features, resulting in the effectiveness of anomaly detection depending on the quality of the features. At the same time, the aforementioned methods ignore the underlying contextual information present in adjacent log entries. We propose a novel model called Logformer with two cascaded transformer-based heads to capture latent contextual information from adjacent log entries, and leverage pre-trained embeddings based on logs to improve the representation of the embedding space. The proposed model achieves comparable results on HDFS and BGL datasets in terms of metric accuracy, recall and F1-score. Moreover, the consistent rise in F1-score proves that the representation of the embedding space with pre-trained embeddings is closer to the semantic information of the log.

KEYWORDS

Anomaly detection; system logs; semi-structured data; pre-trained embedding; cascaded transformer

1 Introduction

With the development of the Internet and the ever-increasing number of Internet users, online systems are evolving and growing in functionality and size. Anomalous events can occur at any time, and high demands have been made for quality of service and security guarantees [1–4]. More services mean more sources of anomalous events, and even a small anomalous event can cause the entire system to respond to it. If abnormal events cannot be detected and removed in a timely manner, the stability and availability of the entire system will be affected. Failure to ensure the stability of the system leads to customer distrust and financial losses. Logs record software status and system information at critical points, providing a rich source of data for system monitoring and anomaly detection. In other words, anomaly detection helps to detect anomalous events and safeguard the stable operation of the system.

This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

An online system typically produces large-scale semi-structured logs. It is impractical to detect and localize anomalies with traditional rule matching and manual screening methods based on logs. As an alternative, many machine learning algorithms have been proposed to extract features from semi-structured logs. PCA [5] is used to extract the principal components of logs to improve classification models. LR [6] and SVM [7] are proposed to classify logs based on the manual designed features. Nevertheless, the aforementioned approaches neglect the long-range dependence on the log. Later, deep learning algorithms were proposed to automatically learn representations for logs without manually designed features. However, most of them fail to extract the potential contextual information present in adjacent log entries, making them uncompetitive in terms of performance. RNN-based models have a natural advantage in capturing long-range dependencies, and LSTMs [8,9] have been proposed to learn extractive representations for logs. Given that randomly initialized LSTMs are difficult to optimize and LSTMs lack parallel computational capacity due to their recurrent structure, self-attention models have been proposed to replace LSTMs with efficient parallel structures. Transformer-based models [10] have been proposed to capture long-range dependencies in an efficient parallel fashion. However, these approaches neglect the potential contextual information present in the adjacent log entries.

To address the above issues, we propose a novel model called Logformer, which consists of two cascaded transformer architectures with an encoder head and a decoder head. The encoder head works well in encoding adjacent log entries into a vector representation, and the decoder head learns latent context information from adjacent log entries. In order to preserve the useful features of logs, we propose a log preprocessing method to replace the regular log parser. In the meantime, we adopt the Glove algorithm to train embeddings for logs, thus making the embedding space more closely related to the log semantics. Experimental results on the HDFS dataset and BGL dataset demonstrate that Logformer outperforms other approaches.

2 Related Work

As an important part of a large-scale system, logs have been of great concern for many years. Many researchers have attempted to use log data to understand the runtime status of a system. However, system logs are usually semi-structured data, which is difficult to handle. In the beginning, keywords are used to find the location of systematic errors. This method can only detect an explicit single anomalous log entry and cannot detect an anomalous event based on the sequence of operations. In other words, an anomalous event in the system log cannot be detected by manually designed keywords. To address the above issues, matching methods [11,12] have been proposed for anomaly detection. These methods rely heavily on manually defined rules in advance and are unable to detect anomalous events from new sources.

With the development of deep learning, deep learning approaches are being applied in the field of anomaly detection. Most of them consist of three steps [13]: first, logs are converted into templates by a log parser; then, the templates are fed into a neural network to learn vector representation for logs; finally, traditional machine learning algorithms such as SVM are applied to classify a log entry into normal or abnormal categories with the learned vector representation. Due to the scarce fraction of anomalous log entries, it is often difficult to extract features from anomalous data. Many researchers use unsupervised approaches [14,15] for anomaly detection.

In terms of semi-structured log entry preprocessing, many methods use parsers. Common ones include Drain [16], AEL [17], IPLoM [18], and Spell [19]. There are also approaches that attempt to obtain semantic vectors directly from embeddings without using a parser. Instead of using parsers for log entries, embeddings are an alternative way to obtain vector representations. LogNL [8] is proposed to utilize the TF-IDF algorithm to obtain template feature representations, and then construct parameter value vectors for logs of different templates. There are several literature proposals to train embeddings for logs by Word2Vec and SIF algorithms.

The attention mechanism can better capture the long-term dependencies in the data and improve the model's ability to extract the most relevant input features. In recent years, attention has been successfully applied to image processing, natural language processing, recommendation systems, and other fields. Xu et al. [20] proposed a model based on attention and AutoEncoder. With attention, the decoder can adaptively select the desired features in the input sequence, which improves the performance of the Encoder-Decoder structure.

Recently, deep learning models [21–26] have achieved remarkable results in the field of anomaly detection. These approaches adopt RNN-based, LSTM-based, or transformer-based models as baselines to extract representations from log entries and predict normal or abnormal labels for log entries based on feature representations. Most of these approaches ignore the underlying contextual information present in the adjacent log entries. Different from regular deep learning models, Logformer effectively captures the long-range dependencies among adjacent log entries through a single decoder head and makes full use of the textual information of the log to train embeddings for the log.

3 Methodology

In this section, we describe in detail the log preprocessing method, the pre-trained embedding algorithm, and the model architecture.

3.1 Overview Pipeline

The model architecture of Logformer is a multi-layer of self-attention and feed-forward network, as shown in Fig. 1. The Logformer consists of an encoder head, which encodes log entries into vectors, and a decoder head, which extracts latent contextual information present in adjacent log entries. Before feeding log entries into Logformer, the preprocessing method is applied to output structured log data. Given an adjacent log entry $\mathrm{X} = (x_1, x_2, \ldots, x_n)$, the embedding layer maps X to word embeddings $\mathrm{E} = (E_1, E_2, \ldots, E_n)$, where E_i denotes the word embedding for a log entry x_i. To exploit the textual information of the logs, pre-trained embeddings trained with the Glove algorithm are used on the logs to initialize the embedding layer. To maintain the order of log entries, position embeddings are added to both the encoder and decoder heads. The encoder head outputs representation is given as $\mathrm{H} = (H_1, H_2, \ldots, H_n)$ for log entries, where H_i is the representation for log entry x_i, and H is regarded as a new sequence. The decoder head takes the H as input and learns the interactive information among different log entries, then outputs a new representation H' for each log entry with abundant context information. Finally, H' is fed into a linear classifier layer to predict whether a log entry is normal or not.

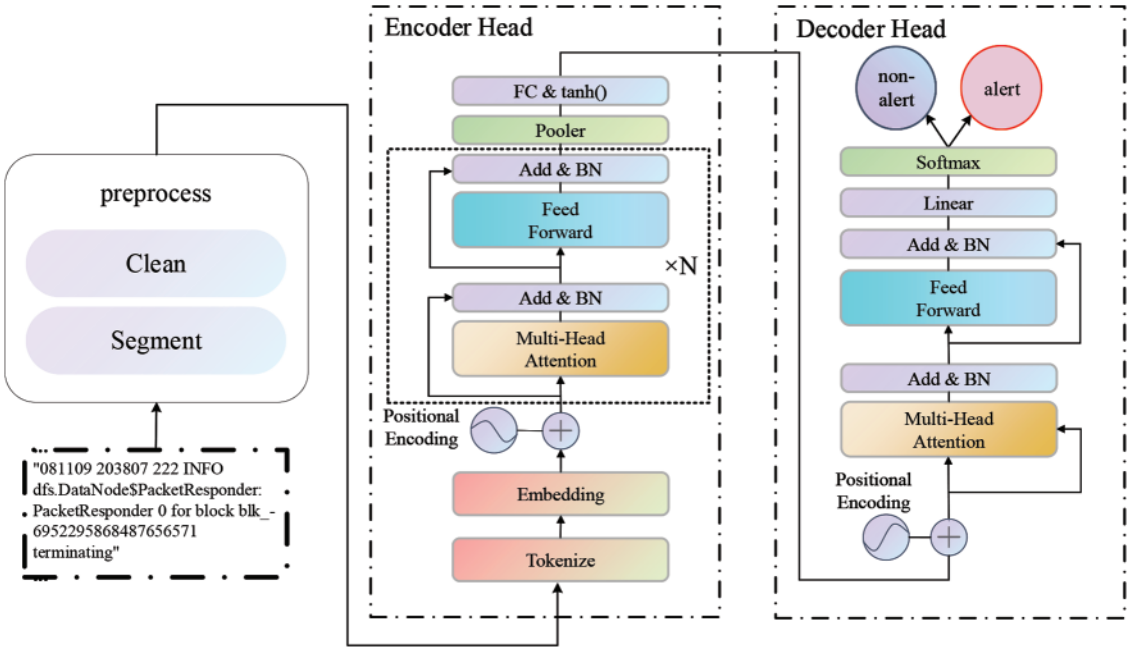

Figure 1: Model architecture: Logformer for log anomaly detection

3.2 Log Preprocessing

Original logs are semi-structured data, and log parsers are widely used in most deep learning methods [22,23]. Lupton et al. [27] carried out a statistic of log parser methods, including publishing year, the number of citations and performance, as shown in Table 1, where AvgPA is an indicator of the average PA of all 16 datasets in Loghub which is detailed described in He et al. [28]. It can be observed from Table 1 that the top three log parsers with the highest number of citations also fail to achieve an Avg PA of 0.9 in the large-scale open-source dataset, which means that these parsers still bring up an unacceptable error. At the same time, the log parsers discard log-level information, resulting in the final source feature log loss. As a result, Logformer takes a different approach to log preprocessing than log parsers. The log preprocessing of Logformer generally consists of two steps. First, each log entry is split by commas, spaces, and other common separators. The log is then converted to lowercase and some characters that do not contain valid information for anomaly detection are removed, such as numbers representing variables and characters not in the template such as 'for', 'ID' and 'of'. Fig. 2 shows the comparison between the original logs and the preprocessed logs.

Table 1: Effectiveness of log parser

Parser	Year	Citations	BGL PA	Avg PA
Drain	2017	122	0.963	0.865
Spell	2016	104	0.787	0.751
SHISO	2013	44	0.711	0.669
SwissLog	2020	36	0.970	0.962
LenMa	2016	19	0.690	0.721

(Continued)

Table 1 (continued)

Parser	Year	Citations	BGL PA	Avg PA
LTMatch	2021	18	0.933	0.889
Paddy	2020	1	0.963	0.895

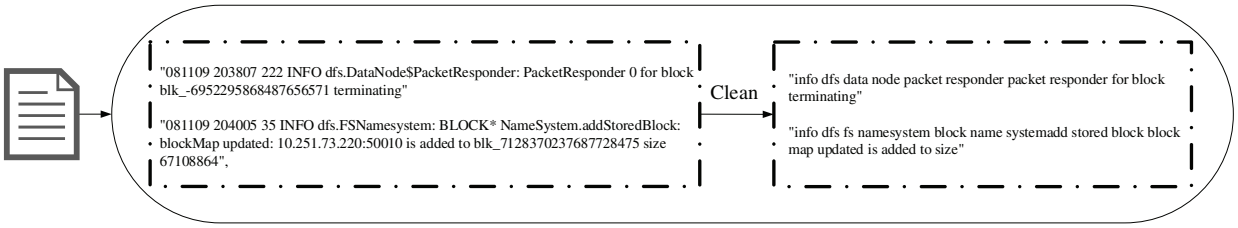

Figure 2: Original logs and preprocessed logs

3.3 Pre-Trained Embedding

This section contains the WordPiece and Glove algorithms, where WordPiece is a subword segmentation algorithm for natural language processing and Glove is a word embedding algorithm for pre-trained embeddings of a corpus.

3.3.1 Tokenizer

Although logs are preprocessed properly in the log preprocessing phase, there are still existing artificially generated words in preprocessed logs like 'DataNode$PacketResponder'. The directly extracted semantic vectors in the form of such words are not well understood, so we still need to perform further word segmentation on the extracted contents. We chose the WordPiece tokenizer to tokenize the preprocessed logs and output the tokenized corpus. WordPiece is trained from a small vocabulary and selects the words that are most likely to occur until all words have been learned.

3.3.2 Glove

To match the embedding spaces with logs, Glove [29] is adopted to train the pre-trained embedding, instead of randomly initializing the embedding layer. As Word2Vec [30] is trained from words in a local sliding window and Glove is trained based on the global statistic co-occurrence matrix, Glove is superior to Word2Vec with global information of the corpus. Let us denote the co-occurrence matrix as $X \in R^{|V|^2}$, where X_{ij} means the times of word j occurs in the context of the word i, and V means the vocabulary size. Denoting the embedding for word i and j as w_i and $\widetilde{w}_j$, respectively, X_{ij} can be approximated by calculating as Eq. (1).

$$w_i^T \widetilde{w}_j + b_i + \widetilde{b}_j = \log X_{ij} \tag{1}$$

The training objective function can be formulated as Eq. (2).

$$J = \Sigma_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \widetilde{w}_j + b_i + \widetilde{b}_j - \log X_{ij}\right)^2 \tag{2}$$

where $f(X_{ij})$ is a weighting function and can be parameterized as Eq. (3).

$$f\left(X_{ij}\right) = \begin{cases} \left(\frac{X_{ij}}{x_{max}}\right)^{\alpha} & X_{ij} < x_{max} \\ 1 & otherwise \end{cases} \tag{3}$$

where α and X_{max} are set to be 0.75 and 1.00, respectively. α works well in improving the accuracy of low-frequency words by scaling up the weight of low-frequency words. Meanwhile, X_{max} limits the maximal times of occurrence.

3.4 Logformer

Most existing deep learning approaches take embeddings of preprocessed logs as input and predict a log entry independently. A single log term may appear to be normal, but when it occurs consecutively in adjacent log entries, it may indicate the occurrence of an anomaly. Therefore, exploiting the long-range dependency information among adjacent log entries can be beneficial for anomaly detection. In this work, Logformer is proposed to address the aforementioned issues. The Logformer consists of a cascaded transformer architecture, an encoder head and a decoder head, where both the encoder and decoder heads have the same architecture. The encoder head contains 6 stacked layers, each consisting of two sub-layers, a multi-head self-attention layer, and a feed-forward network. While the decoder head contains only 1 layer, it also has two sub-layers similar to the encoder head. In the following, we describe the self-attention layer and the feed-forward network in detail.

3.4.1 Multi-Head Self-Attention

Logformer uses the multi-head self-attention to capture long-range dependencies in the adjacent log entries. The heart of the encoder and decoder heads is self-attention, which maps a query and key-value pair to an output where query, key, value, and output are all vectors. Conventional self-attention only contains one head, and multi-head self-attention contains several heads as shown in Fig. 3.

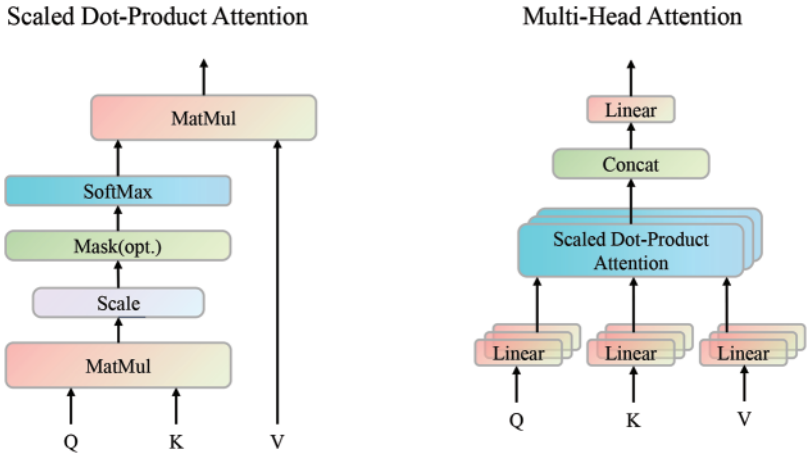

Figure 3: Attention mechanism

Denoting the query, key, and value as Q, K, and V $\in R^{d_{model}}$, separately, self-attention can be calculated as follows:

$$Z = softmax\left(\frac{Q \times K^{T}}{\sqrt{d_{model}}}\right) V \tag{4}$$

where d_{model} is the dimension of value.

Instead of performing a single attention function, Logformer captures richer contexts with multiple individual attention functions. Multi-head self-attention can be regarded as the repetition of h times of self-attention, where h is the number of attention heads. And multi-head self-attention can be calculated as Eq. (5).

$$Z_i = softmax\left(\frac{QW_i^Q \times KW_i^K}{\sqrt{d_h}}\right) VW_i^v \tag{5}$$

where W_i^Q, W_i^K and $W_i^V \in R^{d_{model} \times d_h}$, $d_h = d_{model}/h$.

3.4.2 Feed-Forward Network

The feed-forward network is used to obtain the information in the channel dimension. It applies an expansion operation to x $\in$ $R^{d_{model} \times l}$, and then recovers the intermediate output to the original dimension, which is calculated as Eq. (6).

$$FFN(x) = GELU(xW_1 + b_1)W_2 + b_2 \tag{6}$$

3.4.3 Positional Encoding

Logformer uses self-attention to extract features from adjacent log entries, but self-attention fails to learn information about the order of a sequence. Therefore, positional encoding is added to the encoder and decoder heads to maintain both the order of log entries and the tokens in each log entry. Let us denote $\mathrm{PE}_i \in R^{d_{model}}$ as the positional encoding for token t_i in a log entry, PE_t^i is the i-th element in PE_t, where d_{model} is the dimension of positional encoding and the maximal value of t is set to be 512. PE_t^i can be calculated as follows:

$$PE_t^i = \begin{cases} \sin(w_i t) & if \quad t = 2k \\ \cos(w_i t) & if \quad t = 2k+1 \end{cases} \tag{7}$$

where $w_i = \dfrac{1}{10000^{\frac{2i}{d_{model}}}}$, $i = 0, 1, 2, 3, \ldots, \frac{d_{model}}{2} - 1$.

3.4.4 Batch Normalization

It is well-known that normalizing the feature maps makes training faster and more stable. Logformer incorporates Batch Normalization (BN) in attention in place of the original Layer Normalization (LN) in transformer. Layer Normalization is commonly used in RNN, where the sequence length is often not fixed. Since LN does not depend on batch size and sequence depth, it performs better in RNN. Compared to LN, BN shows better robustness and generalization ability. Also, BN has the advantage that it is generally faster in inference than other batch-independent normalizers such as LN. BN treats the batch data as a whole. The batch dimension is used in the calculations of both mean and variance [31]. An important feature of Logformer is to combine contextual information from multiple adjacent log entries. We want the normalization of Logformer to be sensitive to batch size, so we choose BN instead of LN.

4 Experiment

4.1 Dataset

In this paper, we select open-source datasets HDFS and BGL in LogHub [28] to validate the effectiveness of Logformer, as Table 2 shown. These two log datasets are widely used in the fields of anomaly detection. The HDFS dataset is generated by the benchmark workload in the private cloud environment, and labels are made by manually designed rules to identify the abnormal events. The BGL dataset contains logs collected from the BlueGene/L supercomputer system by Lawrence Livermore National Labs (LLNL). BGL is divided into altered and no-alter log entries. The first column in the BGL log contains '-' or not, where '-' means no-alter log entry, and the other is altered.

Table 2: Statistic of dataset

Dataset	Log source	Number of logs	Type
HDFS	Hadoop distributed file system	11175629	Distributed system
BGL Blue Gene/L	Super computer	4747963	Super computer

4.2 Evaluation Metrics

We select precision, recall, and F1-score as the evaluation metrics. These three metrics are based on the confusion matrix. The confusion matrix has four categories: True positives (TP) are examples correctly labeled as positives. False positives (FP) refer to negative examples incorrectly labeled as positive. True negatives (TN) correspond to negatives correctly labeled as negative. Finally, false negatives (FN) refer to positive examples incorrectly labeled as negative.

Precision is calculated as the percentage of correctly predicted positive samples accounting for all predicted positive samples. Recall is calculated as the percentage of correctly predicted positive samples accounting for all real positive samples. And F1-score is an indicator to compute the average of precision and recall.

$$precision = \frac{TP}{TP + FP} \tag{8}$$

$$recall = \frac{TP}{TP + FN} \tag{9}$$

$$F1 = \frac{2 \times precision \times recall}{precision + recall} \tag{10}$$

where TP, FP, and FN mean the true positives, false positives, and false negatives, respectively.

4.3 Implementation Details

We construct Logformer by two cascaded transformers. The number of transformer layers of encode head and decode head is 6 and 1, respectively. The hidden dimension is 128 and the number of attention heads in the logformer is 6. Layer normalization is replaced with batch normalization in the transformer layer. We use AdamW to optimize all parameters. The learning rate is set to 5e-4 and the batch size is set to 32.

4.4 Experiment Result

Experiments on HDFS and BGL datasets are shown in Table 3. We compared Logformer with several existing methods in two public datasets, including the data-driven method PCA [32], traditional machine learning method LR [33] and SVM [33], and the deep learning method LogRobust [34]. Due to its limitation, PCA achieves poor results in both HDFS and BGL datasets. It can be observed that both conventional machine learning and deep learning methods obtain consistent high performance in the HDFS dataset. The reason behind this is that log entries in HDFS tend to be more structured than BGL, and abnormal log entries are quite different from normal log entries. In the complicated BGL dataset, it can be seen in Table 3, that LogRobust and Logformer outperform LR and SVM by a large margin. Significantly, Logformer is superior to LogRobust with an increment of 8% in F1-score, which demonstrates that Logformer is more suitable for complicated semi-structured logs.

Table 3: Experiment results on HDFS and BGL datasets

Dataset		PCA	LR	SVM	LogRobust	Logformer
HDFS	Precision	0.06	**0.99**	**0.99**	0.98	**0.99**
	Recall	1.00	0.92	0.94	**1.00**	**1.00**
	F1	0.11	0.96	0.96	**0.99**	**0.99**
BGL	Precision	0.09	0.13	**0.97**	0.62	0.87
	Recall	**0.98**	0.93	0.30	0.96	0.79
	F1	0.17	0.23	0.46	0.75	**0.83**

4.5 Ablation Study

To invalidate the hypothesis that pre-trained embedding in logs makes the embedding space match the textual information of logs better, we conduct comparative experiments by respectively using pre-trained embedding and randomly initialized embedding in HDFS and BGL datasets. In addition, we validate the importance of extracting context information existing in adjacent log entries by adding/removing the encoder and decoder head. Finally, we discuss the influence of the number of log entries on anomaly detection through experiments.

As shown in Table 4, the pre-trained embedding is superior in randomly initializing embedding in all metrics in both HDFS and BGL datasets. Pre-training can extract the prior knowledge of the task, improve the performance of Logformer by training the embeddings, which makes good use of the textual information of logs.

Table 4: Experiment result on the effectiveness of pretraining embedding

Dataset		Random	Pretrain
HDFS	Precision	0.96	**0.99**
	Recall	0.97	**1.00**
	F1	0.96	**0.99**
BGL	Precision	0.81	**0.87**
	Recall	0.74	**0.79**
	F1	0.77	**0.83**

When validating the effectiveness of the encoder and decoder header, we set three comparative studies. w/o means predicting directly after an average sum of the pre-trained embedding; w encoder means predicting from the output of the dencoder head; w encoder-decoder means predicting from the output of the decoder head. It can be observed from Table 5 that the encoder head obtains a better result than pre-trained embedding in two datasets, which demonstrates that the self-attention layer in the encoder head can capture the log-range dependency information existing in a log entry. After adding both the encoder and decoder head, the results are better than only adding the encoder head, which means the decoder head improves the performance of Logformer by extracting potential information from adjacent log entries.

Table 5: Experiment result on the effectiveness of decoder head

Dataset		w/o	w encoder	w encoder-decoder
HDFS	Precision	0.79	0.94	**0.99**
	Recall	0.82	0.91	**1.00**
	F1	0.80	0.92	**0.99**
BGL	Precision	0.71	0.80	**0.87**
	Recall	0.62	0.71	**0.79**
	F1	0.66	0.75	**0.83**

As shown in Table 6, we try to use different batch sizes for experiments. In the Logformer, batch size represents the number of logs used for context combination. With the increase in batch size, the detection performance of the Logformer is also slightly improved, which is in line with the intuition. But a larger batch size means larger resource consumption and longer time, and the batch size can be adjusted as needed during actual use.

Table 6: Experiment result on the effectiveness of batch size

Dataset		64	128	256	512
HDFS	Precision	0.90	0.94	0.98	**0.99**
	Recall	0.93	0.96	0.99	**1.00**
	F1	0.91	0.95	0.98	**0.99**
BGL	Precision	0.76	0.82	0.85	**0.87**
	Recall	0.67	0.72	0.74	**0.79**
	F1	0.71	0.77	0.80	**0.83**

The experimental results show that the Logformer using the complete cascaded structure achieves the best results in all datasets, which proves that our proposed cascaded structure is efficient. The encoder first combines multiple semantic vectors to complete the encoding, and then the context information association between adjacent log entries is captured by the decoder. The semantic vector containing rich context information can make a significant contribution to system log anomaly detection.

5 Conclusion

This paper proposes an anomaly detection method with the cascaded structure that can make full use of potential context information among adjacent logs. We also propose a log preprocessing method to convert semi-structured logs into structured logs by removing common punctuation and other redundant characters in logs. To make the embedding space match with the textual semantic of logs, the WordPiece algorithm is adopted to tokenize the preprocessed logs into subwords, and the Glove algorithm is used to train embedding based on the log corpus. A cascaded structure model Logformer is finally designed to learn vector representation for each log entry and extract long-range dependency among adjacent log entries.

Logformer achieves superior performance compared to conventional machine learning algorithms and some deep learning models. From the perspective of two ablation studies, Logformer outperforms other approaches by efficiently extracting potential information existing in adjacent log entries. However, there are still more challenges in the real scenario. We do not provide the expert system to realize the feedback mechanism, and the feedback mechanism often plays a crucial role in an online system. The above problems are the direction of our future efforts.

Acknowledgement: The authors wish to express their appreciation to the reviewers for their helpful suggestions which greatly improved the presentation of this paper.

Funding Statement: This work was supported by the National Natural Science Foundation of China (Nos. 62072074, 62076054, 62027827, 61902054, 62002047), the Frontier Science and Technology Innovation Projects of National Key R&D Program (No. 2019QY1405), the Sichuan Science and Technology Innovation Platform and Talent Plan (No. 2020TDT00020), the Sichuan Science and Technology Support Plan (No. 2020YFSY0010).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

1. Bauer, E., Adams, R. (2012). *Reliability and availability of cloud computing*. New Jersey, USA: John Wiley & Sons.
2. Huang, X., Xiong, H., Chen, J., Yang, M. (2021). Efficient revocable storage attribute-based encryption with arithmetic span programs in cloud-assisted Internet of Things. *IEEE Transactions on Cloud Computing,* 1–12.
3. Kazemzadeh, R. S., Jacobsen, H. A. (2009). Reliable and highly available distributed publish/subscribe service. *2009 28th IEEE International Symposium on Reliable Distributed Systems*, pp. 41–50. New York, USA.
4. Xiong, H., Zhou, Z., Wang, L., Zhao, Z., Huang, X. et al. (2021). An anonymous authentication protocol with delegation and revocation for content delivery networks. *IEEE Systems Journal, 16(3),* 4118–4129.
5. Xu, W., Huang, L., Fox, A., Patterson, D., Jordan, M. I. (2009). Detecting large-scale system problems by mining console logs. *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, pp. 117–132. Montana, USA.
6. Bodik, P., Goldszmidt, M., Fox, A., Woodard, D. B., Andersen, H. (2010). Fingerprinting the datacenter: Automated classification of performance crises. *Proceedings of the 5th European Conference on Computer Systems*, pp. 111–124. Paris, France.

7. Chen, M., Zheng, A. X., Lloyd, J., Jordan, M. I., Brewer, E. (2004). Failure diagnosis using decision trees. *Proceedings of International Conference on Autonomic Computing*, pp. 36–43. New York, USA.
8. Meng, W., Liu, Y., Zhu, Y., Zhang, S., Pei, D. et al. (2019). Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. *Proceedings of the 2019 International Joint Conference on Artificial Intelligence*, pp. 4739–4745. Macao, China.
9. Zhu, B., Li, J., Gu, R., Wang, L. (2020). An approach to cloud platform log anomaly detection based on natural language processing and LSTM. *Proceedings of the 2020 3rd International Conference on Algorithms, Computing and Artificial Intelligence*, pp. 1–7. Sanya, China.
10. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L. et al. (2017). Attention is all you need. *Proceedings of the 30th Advances in Neural Information Processing Systems*, CA, USA, Curran Associates, Inc.
11. Cinque, M., Cotroneo, D., Pecchia, A. (2013). Event logs for the analysis of software failures: A rule-based approach. *IEEE Transactions on Software Engineering, 39(6),* 806–821. DOI 10.1109/TSE.2012.67.
12. Yen, T. F., Oprea, A., Onarlioglu, K., Leetham, T., Robertson, W. et al. (2013). Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks. *Proceedings of the 29th Annual Computer Security Applications Conference*, pp. 199–208. LA, USA.
13. He, S., Zhu, J., He, P., Lyu, M. R. (2016). Experience report: System log analysis for anomaly detection. *Proceeding of the IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 207–218. ON, Canada.
14. Xu, W., Huang, L., Fox, A., Patterson, D., Jordan, M. (2009). Online system problem detection by mining patterns of console logs. *Proceeding of the 2009 Ninth IEEE International Conference on Data Mining*, pp. 588–597. Florida, USA.
15. Lou, J. G., Fu, Q., Yang, S., Xu, Y., Li, J. (2010). Mining invariants from console logs for system problem detection. *Proceeding of the 2010 USENIX Annual Technical Conference (USENIX ATC 10)*, MA, USA.
16. He, P., Zhu, J., Zheng, Z., Lyu, M. R. (2017). Drain: An online log parsing approach with fixed depth tree. *Proceedings of the 2017 IEEE International Conference on Web Services (ICWS)*, pp. 33–40. HI, USA.
17. Jiang, Z. M., Hassan, A. E., Flora, P., Hamann, G. (2008). Abstracting execution logs to execution events for enterprise applications (short paper). *Proceedings of the 2008 the Eighth International Conference on Quality Software*, Oxford, UK.
18. Makanju, A. A., Zincir-Heywood, A. N., Milios, E. E. (2009). Clustering event logs using iterative partitioning. *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1255–1264. Paris, France.
19. Du, M., Li, F. (2016). Spell: Streaming parsing of system event logs. *Proceedings of the 2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 859–864. Barcelona, Spain.
20. Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A. et al. (2015). Show, attend and tell: Neural image caption generation with visual attention. *International Conference on Machine Learning*, pp. 2048–2057. Lille, France.
21. Zhang, K., Xu, J., Min, M. R., Jiang, G., Pelechrinis, K. et al. (2016). Automated it system failure prediction: A deep learning approach. *Proceeding of the 2016 IEEE International Conference on Big Data (Big Data)*, pp. 1291–1300. Washington DC, USA.
22. Du, M., Li, F., Zheng, G., Srikumar, V. (2017). Deeplog: Anomaly detection and diagnosis from system logs through deep learning. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1285–1298. TX, USA.
23. Han, S., Wu, Q., Zhang, H., Qin, B., Hu, J. et al. (2021). Log-based anomaly detection with robust feature extraction and online learning. *IEEE Transactions on Information Forensics and Security, 16,* 2300–2311. DOI 10.1109/TIFS.10206.

24. Yuan, Y., Adhatarao, S. S., Lin, M., Yuan, Y., Liu, Z. et al. (2020). ADA: Adaptive deep log anomaly detector. *Proceeding of the 2020 IEEE INFOCOM IEEE Conference on Computer Communications*, pp. 2449–2458. ON, Canada.
25. Liu, F., Wen, Y., Zhang, D., Jiang, X., Xing, X. et al. (2019). Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise. *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1777–1794. London, UK.
26. Wang, Z., Chen, Z., Ni, J., Liu, H., Chen, H. et al. (2021). Multi-scale one-class recurrent neural networks for discrete event sequence anomaly detection. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 3726–3734. Singapore.
27. Lupton, S., Washizaki, H., Yoshioka, N., Fukazawa, Y. (2021). Online log parsing: Preliminary literature review. *2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 304–305. Wuhan, China.
28. He, S., Zhu, J., He, P., Lyu, M. R. (2020). Loghub: A large collection of system log datasets towards automated log analytics. arXiv preprint arXiv:2008.06448.
29. Pennington, J., Socher, R., Manning, C. D. (2014). Glove: Global vectors for word representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543. Doha, Qatar.
30. Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Proceedings of the 26th Advances in Neural Information Processing Systems (NIPS 2013)*, Nevada, USA, Curran Associates, Inc.
31. Yao, Z., Cao, Y., Lin, Y., Liu, Z., Zhang, Z. et al. (2021). Leveraging batch normalization for vision transformers. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 413–422. BC, Canada.
32. Guo, H., Yuan, S., Wu, X. (2021). Logbert: Log anomaly detection via bert. *Proceedings of the 2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. Shenzhen, China.
33. Le, V. H., Zhang, H. (2021). Log-based anomaly detection without log parsing. *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 492–504. Melbourne, Australia.
34. Zhang, X., Xu, Y., Lin, Q., Qiao, B., Zhang, H. et al. (2019). Robust log-based anomaly detection on unstable log data. *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 807–817. Tallinn, Estonia.