Log Anomaly Detection Through GPT-2 for Large Scale Systems

JI Yuhe[1], HAN Jing[2], ZHAO Yongxin[1], ZHANG Shenglin[1], GONG Zican[2]

(1. Nankai University, Tianjin 300071, China;
2. ZTE Corporation, Shenzhen 518057, China)

DOI: 10.12142/ZTECOM.202303010

https://kns.cnki.net/kcms/detail/34.1294.TN.20230802.1731.004.html, published online August 3, 2023

Manuscript received: 2022-12-08

Abstract: As the scale of software systems expands, maintaining their stable operation has become an extraordinary challenge. System logs are semi-structured text generated by the recording function in the source code and have important research significance in software service anomaly detection. Existing log anomaly detection methods mainly focus on the statistical characteristics of logs, making it difficult to distinguish the semantic differences between normal and abnormal logs, and performing poorly on real-world industrial log data. In this paper, we propose an unsupervised framework for log anomaly detection based on generative pre-training-2 (GPT-2). We apply our approach to two industrial systems. The experimental results on two datasets show that our approach outperforms state-of-the-art approaches for log anomaly detection.

Keywords: hybrid beamforming; hybrid architecture; weighted mean square error; manifold optimization; dynamic subarrays

Citation (Format 1): JI Y H, HAN J, ZHAO Y X, et al. Log anomaly detection through GPT-2 for large scale systems [J]. ZTE Communications, 2023, 21(3): 70–76. DOI: 10.12142/ZTECOM.202303010

Citation (Format 2): Y. H. Ji, J. Han, Y. X. Zhao, et al., "Log anomaly detection through GPT-2 for large scale systems," *ZTE Communications*, vol. 21, no. 3, pp. 70–76, Sept. 2023. doi: 10.12142/ZTECOM.202303010.

1 Introduction

As the scale of software systems expands, maintaining their stable operation has become an extraordinary challenge. System logs are the text generated in the process of software running, which records the status information of the program[1–2]. Traditional log anomaly detection relies on manual analysis by operation engineers. The increase in software systems leads to a surge in a log volume, manual analysis of logs and the design of regular expressions can consume considerable time, and the traditional log analysis is no longer feasible. In recent years, researchers have proposed a variety of automated log anomaly detection methods[3–8]. Early research on log exceptions focused on automatic exception rule generation and simple statistical methods[9]. Then, researchers divided log analyses into three directions: template extraction, model training, and anomaly detection. In recent years, many intelligent log analysis methods have been proposed with the rapid development of machine learning, especially deep neural networks. For example, models based on convolutional neural networks (CNN) and recurrent neural networks (RNN) can effectively capture the sequence characteristics and frequency characteristics of log text, and apply them to the detection of new logs[3–5]. However, the application of the deep learning model in practical scenarios faces the following challenges:

1) The normal pattern of logs is difficult to model. Most existing methods try to learn the normal pattern of log sequence and frequency. The team's O&M engineers pointed out that the semantics of logs in real-world scenarios are of considerable analytical importance. Since the production environment for generating and recording logs is not ideal, relying solely on sequence as well as frequency to encode logs can lead to a large number of false positives.

2) Complex log data contains massive noise. Due to the high concurrency of the software system and the uncertainty of network response, the log generated can be disordered and contain lots of noise.

To tackle the limitations of existing methods, in this paper, we propose an efficient and robust framework for log anomaly detection based on generative pre-training-2 (GPT-2)[10]. Inspired by the excellent performance of GPT-2 in serialization text generation and multiple types of downstream task processing, we leverage this model to capture patterns of normal log sequences.

The main contributions of this paper can be summarized as follows:

1) To tackle the first challenge, we generate sentence vectors for each log template to represent their semantic information. Specifically, we first apply Siamese Bidirectional Encoder Representations from Transformers (SBERT) networks to generate sentence vectors for all log templates and then in-

put the vectors into GPT-2 as the representation of templates. In this way, the models can study both sequential and semantic features of input logs.

2) To address the second challenge, we design an alarm strategy layer for this framework. This step will analyze the statistical characteristics through the existing log data to effectively reduce the impact of noise on model judgment and reduce the false positive rate.

The remainder of this paper is organized as follows. Section 2 introduces the background of log anomaly detection. Section 3 describes our approach. In Section 4, we present our experimental design and results. Section 5 surveys related works and Section 6 concludes this paper.

2 Background

2.1 Logs Description

Logs, which are produced by the running program and contain information generated from the logging module, are a type of semi-structured text. They reflect the running flow and real-time status of the program, and can be used to detect and localize anomalies by operators. Logs consist of a structured part (constant) and an unstructured part (variable), as shown in Fig. 1. A structured part is fixed by the designer at the beginning according to certain designed rules, and can reflect the event of the log, such as the level of logs (e.g., warning and error), the logger name (e.g., root) and so on. This part would not change into the same module. The unstructured part contains specific information on logs and varies according to the input of the program and running status. An unstructured part would reflect the real-time status of the system, and it is essential to use this part to analyze the system and detect anomalies in the system. To make full use of the important features in logs, the semi-structured texts need to be parsed into the structured text with a parsing algorithm. The useless messages in the raw log would be filtered out, and the valuable information would be extracted out from the left information to train the model and detect the anomaly.

Generally speaking, the logs generated by normally operating hardware and software systems have a good regularity. Therefore, some logs that do not match the pattern of previous characteristics are considered anomalous. In the actual dataset studied in this paper, log exceptions can be broadly classified into two categories: 1) Business exceptions caused by network blocking, resource usage, etc. When such exceptions are generated, the program will actively retry the process, so some logs will be repeatedly generated several times in a short period of time. 2) Exceptions triggered by service deployment or termination failure. This type of exception usually generates only a few error logs, which can be evidenced by the fact that the sequence of logs is abnormal, and the semantics of the error logs differ significantly from the normal logs.

```
RAS KERNEL INFO CE sym 0, at 0x0b8580c0, mask 0x10
RAS KERNEL INFO CE sym 25, at 0x04bfb9e0, mask 0x20
RAS KERNEL INFO CE sym 2, at 0x1b85fe80, mask 0x02

RAS MMCS INFO mmcs db server has been started
RAS MMCS INFO mmcs db server has been closed
```

▲ Figure 1. Log instances, where black words are the structured part and red words are the unstructured part

2.2 Challenges and Analysis

1) Challenge 1 is modeling the normal patterns of logs. Traditional anomaly detection algorithms always work on learning the normal patterns of logs and finding out logs different from normal patterns. Most studies mainly focus on building the normal pattern according to the sequence and frequency of logs, however, these two features are not comprehensive enough to evaluate the overall state of the system, hence the normal patterns built on these two features are not accurate[3–4]. Intuitively, each log has its semantic characteristics through the log message, and these would describe the log meaning, such as inserting in a dataset, deleting from a dataset, and failure reporting. Besides, logs are generated by triggering corresponding events, and an event always triggers a series of logs. So, there is a sequential relation between logs, which would become different from the usual and could represent the status of logs and systems. As above, if we could combine the semantic information and sequential message with other statistical information, it could perform better in detecting anomalies in logs.

2) Challenge 2 is massive noises in the log. Logs are produced by a software system, which is highly concurrent and greatly influenced by the network. A working software system can run a large number of programs at the same time. These programs can produce a series of logs as well as useless messages, such as test output, and these messages have no effect on evaluating the log status. Some programs need to interact with other programs in the network, and thus the status of the network would affect the log sequence and delays can disrupt the order of logs. The disordered log and useless information are collectively called noise. Dealing with noise properly is necessary to improve anomaly detection in logs, and the simplest way is to remove the noise, which, however, roughly brings lots of missing areas in logs and could bring new problems to the model. Based on the above observation, intuitively, if we can filter out the false positive alarms instead of roughly removing these noises, log anomaly detection can achieve higher accuracy and lower the false alarm rate.

2.3 Preliminaries

1) The log parsing algorithm. To parse the semi-structured log into a structured text, the log parsing algorithm focuses on fetching the unstructured part from the structured part and extracting the features of logs. Specifically, the log parsing algorithm would replace meaningless information (e.g., IP address,

machine ID, etc.) with markers and extract the template of the log to distinguish between logs. In this paper, we adopt Drain[11], a heuristic log parsing algorithm, to parse logs. Drain parses the template of logs by maintaining a tree, which keeps the leaves as log groups with different heuristic rules in internal nodes. New logs would be distributed into log groups in leaves according to the corresponding tree path.

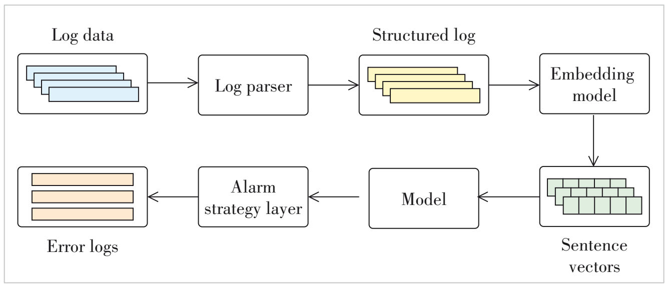
2) GPT-2. The GPT algorithm works based on the decoder of the transformer, and it pre-trains unsupervisedly on massive corpus data and then finetunes the model according to the specific tasks. GPT-2 removes the finetune step and increases the size of the training dataset and model, which makes the model perform well in multi-tasking.

3 Approach

In this section, we will introduce our log anomaly detection framework. Inspired by GPT-2, we employ transformer decoders to encode the normal pattern of system logs. Our framework is divided into four main parts: log parsing, sentence vector generation, model training and detection, and the alerting strategy layer. For Challenge 1, we embed each log template into a vector representing the semantic information. We use vectors instead of tokens as input to GPT-2, so the model can capture both semantic and sequence information to encode normal patterns. For Challenge 2, we design an alarm strategy layer for the framework, which can filter out false positives by the statistical characteristics of log data. The structure of the framework of our work is shown in Fig. 2. The raw logs collected are first transformed into structured logs by the log parser. Then a sentence vector generation model will be used to generate sentence vectors for each extracted log template.

3.1 Log Parser

System logs obtained from the database are semi-structured text, which is difficult to be used for model training. Therefore, before processing logs, we need to use a parser to convert logs into structured text. In this paper, we use Drain as our log parser, which can divide logs into templates and dynamic variables. Drain is an online log template miner, employing a parse tree with a fixed depth, and it can extract templates and variables from a stream of log messages. Drain first sorts logs into different buckets by length and then matches similar portions of the log from front to back in each bucket. Eventually, logs belonging to the same template will end up on the same leaf node. The structure of the depth-fixed tree is shown in Fig. 3. In the figure, sys is an abbreviation for system, and HEX and NUM are variables matched during the parsing process, representing hexadecimal and decimal numbers, respectively.



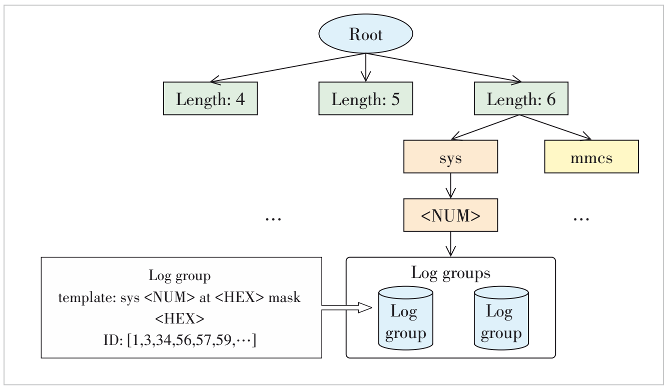
▲Figure 2. Approach overview

3.2 Sentence Vector Generation

To make GPT-2 better at encoding normal patterns, we generate semantically relevant sentence vectors for each log template. We choose SBERT[12] as our embedding model, which has been widely used in text similarity calculation and sentence classification problems and has achieved excellent results. SBERT modifies the pre-trained BERT model, and it implements Siamese or triplet net frameworks to generate semantically meaningful sentence embedding. Sentence vectors generated by templates with similar meanings have smaller cosine distances or Euclidean distances. Table 1 shows the Euclidean distance of sentence vectors in five log templates.

3.3 Detection Model

1) Model framework. We choose GPT-2 as our log anomaly detection model in this work. GPT-2 is an unsupervised Natural Language Processing (NLP) model stacked from the transformer's decoders. The structure of GPT-2 is shown in Fig. 4. Each decoder has a masked self-attention layer, a feed-forward neural network, and two normal layers. The structure of each decoder is the same, but each module maintains separate parameters. Different from the ordinary self-attention layer, the masked self-attention layer does not allow a node to

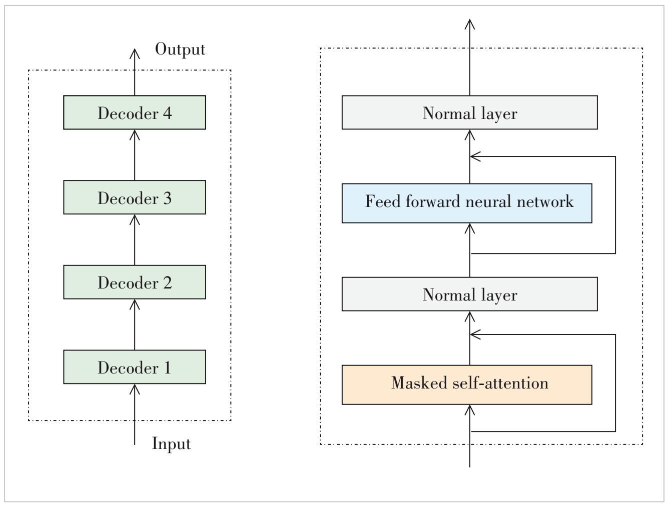


▲Figure 3. Structure of the depth-fixed tree

▼Table 1. Euclidean distance of sentence vectors of similar semantic templates

Templates	Euclidean Distance
httprequest except <*> permission denied	-
httprequest except <*> <*> permission denied	0.147 629 340 284 133 4
httprequest except <*> no such file or directory	0.595 852 332 701 891 4
httprequest except <*>	0.621 201 472 867 456 3
httprequest except EoF occurred in violation of protocol	0.838 852 193 154 771 3
httprequest except <*> connection reset by peer	0.880 359 580 380 884 6

EoF: end-of-file



▲Figure 4. Structure of generative pre-training-2 (GPT-2)

get information from subsequent nodes. This feature makes the model perform well in sequence prediction tasks.

2) Input representation. Like most NLP models, GPT-2 looks up the embedding vector corresponding to the word from the embedding matrices, and the embedding matrices are also part of the model training results. GPT-2 maintains two embedding matrices: a token embedding matrix and a position embedding matrix. Each row of the token embedding matrix is a vector that represents a token, and these vectors are randomly initialized and continuously adjusted during training. Each row of the position embedding matrix represents the position information of the token. Tokens in the same position in different sentences have the same position embedding vector. Each template embedding inputted into GPT-2 is the sum of position embedding and token embedding. In our framework, we initialize the token embedding matrix with sentence vectors generated by SBERT, which unlike the original random initialization, can make the model catch the initial semantic information of log templates more accurately. Besides, this approach gives operations more control over the model. We can adjust the dimension or generation method of sentence vectors according to the characteristics of logs and actual business requirements.

3) Model training. The core concept of GPT-2 is language modeling. Language modeling refers to distribution estimation from a group of unsupervised samples (x_1, x_2, x_3, ⋯). Each sample consists of symbol sequences of variable length (s_1, s_2, s_3, ⋯). Because there are explicit sequential relationships between phrases in natural languages, language modeling typically decomposes the joint probability of symbols as a product of conditional probabilities.

$$p_{(x)} = \prod_{i=1}^{n} p\left(s_i|s_1, s_2, \cdots, s_{n-1}\right). \tag{1}$$

Transforming the above equation into a logarithmic form, the goal of the language model is to maximize the probability of the following equation.

$$p(x) = \sum_{i=1}^{n} \log p\left(s_i|s_1, s_2, \cdots, s_{n-1}; \theta\right), \tag{2}$$

where n is the length of the language sequence, and the conditional probability p is modeled by the neural network with parameter θ. These parameters are trained by the stochastic gradient descent.

The input of the decoder at the first layer consists of the token embedding vector and position embedding vector of log templates. The output of each decoder is processed from the previous layer's output. The output probability obtained by the model can be expressed as follows:

$$\begin{aligned} h_0 &= \boldsymbol{X}\boldsymbol{W}_e + \boldsymbol{W}_p, \\ h_l &= \text{Decoder}\left(h_{l-1}\right) \forall l \in [1,n], \\ p(x) &= \text{softmax}(h_n \boldsymbol{W}_e^T), \end{aligned} \tag{3}$$

where h_l represents the output of the l-th layer decoder, $\boldsymbol{X}$ is the matrix composed of the unique thermal encoding of the input log sequence, $\boldsymbol{W}_e$ is the token embedding matrix, and $\boldsymbol{W}_P$ is the position embedding matrix. To maximize the prediction probability, the model adjusts the decoder parameters of each layer during the learning process.

3.4 Alarm Strategy

Analyzing the actual data, we find that in the production environment, the logs printed by the machine are not always sequential. The main differences between these noises and severe systems are as follows: logs corresponding to noise appear frequently and usually have a certain seasonality, while severe anomalies occur infrequently and are difficult to predict. Based on the above observation, we use frequency and periodicity as criteria to determine whether model error reporting is noisy or a true anomaly.

For the abnormal log templates detected by the model, we first calculate the time interval of their occurrence in the training data and then use the auto-correlation coefficient to analyze whether the time interval of template occurrence has a specific pattern. The auto-correlation coefficient is a common parameter for finding repetitive patterns (e.g., periodic signals masked by noise) and is often used in signal processing problems. The sequence consisting of the auto-correlation coefficients is known as the auto-correlation function. For the obtained template interval sequence, its autocorrelation function is calculated, and the peak of the function is the possible period of the corresponding template. If the value of the function at a certain time is higher than a threshold value set based on expert experience, we consider the template to be periodic and its basic impossibility to be an error log template.

Then, for non-periodic templates, we go through the statistics of their daily frequency of occurrence. Based on the observation of the data and the communication with the operation and maintenance staff, we have learned that the probability of daily serious anomalies in a smoothly running system is extremely low. An exception log with a high frequency is often caused by minor errors such as network blocking and data locking. The system can often recover from such errors quickly, so such alarms are not necessary. Therefore, for exception log templates that occur more frequently than a certain threshold, the alert policy layer will filter them out as less serious exception logs. The selection of this frequency threshold is strongly correlated with the type of machine logs and relies on the involvement of business experts.

In this layer, error logs detected by GPT-2 will be analyzed, and if their characteristics are more like noise, the exception will not be reported. This step can greatly reduce the model false positives caused by noise, improve the accuracy of the log anomaly detection framework, and reduce the disturbance to operation engineers.

4 Experiment and Evaluation

4.1 Experimental Setup

1) Research questions. In this section, we evaluate the performance of our framework with the following research questions (RQs):

a) RQ1: How effective is our framework in log-based anomaly detection?

b) RQ2: How effective is the sentence vector generation and alerting strategy layer in improving the effectiveness of the model.

2) Datasets. We evaluate our approach on two large-scale systems called Ada and Bob. Ada is a framework for microservice deployment applications. Bob is a hardware network consisting of a large number of switch systems. The statistics of the datasets are shown in Table 2.

3) Baselines. We compare our framework with two baselines, LogAnomaly[4] and NeuralLog[8]. LogAnomaly is a log anomaly detection method based on a long short-term memory (LSTM) network. LogAnomaly first uses a Frequent Term Tree (FT-Tree) to analyze the semi-structured log text. Then, Template2Vec is implemented to generate vectors for each log template. Finally, the vectors representing log semantics and frequency are input into the LSTM to enable the model to learn the normal pattern of logs. NeuralLog is a novel log-based anomaly detection framework. Different from the traditional process, the algorithm does not require template parsing. NeuralLog generates semantic vectors for row logs. These representation vectors are then used to detect anomalies through a transformer-based classification model.

4) Evaluation metrics. We use Precision, Recall, and F1-Score (F1S) as our evaluation metrics, which are defined as follows. True Positive (TP) is the number of abnormal logs that are correctly detected by the model, False Positive (FP) is the number of normal logs that are wrongly identified as anomalies, and False Negative (FN) is the number of abnormal logs that are not detected by the model.

$$\text{Precision} = \frac{TP}{TP + FP}. \tag{4}$$

$$\text{Recall} = \frac{TP}{TP + FN}. \tag{5}$$

$$F1S = \frac{2*\text{Precision}*\text{Recall}}{\text{Precision} + \text{Recall}}. \tag{6}$$

4.2 Experimental Results

In this section, we will give response to the RQs mentioned above.

1) RQ1: How effective is our framework in log-based anomaly detection?

In this RQ, we evaluate whether our framework can work effectively on logs generated in the production environment. We compare our framework with two baselines: LogAnomaly[4] and NeuralLog[8].

Table 3 shows the results of our method as well as two baselines on Ada and Bob. Both LogAnomaly and NeuralLog show poor Precision and Recall performance on Ada. LogAnomaly has very limited learning capability due to the limitation of model size, which makes it difficult to obtain good results on log datasets with a large number of templates and complex processes. Also, in the production environment logs, log templates that are not present in the training data often appear in the test set, making LogAnomly generate a large number of false positives often[18]. NeuralLog tends to consider every log unlikely to be anomalous on large unsupervised datasets due to the problem of data dilution. Our framework analyzes and learns the actual semantics of the logs, and designs an alert policy layer that incorporates the actual business characteristics of the machine. These measures make our model more capable of capturing the normal patterns of real logs in a produc-

▼Table 2. Statistics of evaluation datasets

Dataset	Training Data	Number of Templates	Test Dataset	
			Normal	Anomalous
Ada	6 626 865	599	7 911 944	2 648
Bob	7 021 577	84	1 067 850	904

▼Table 3. Evaluation results of our method vs the other two methods

Approach	Ada			Bob		
	Precision	Recall	F1S	Precision	Recall	F1S
LogAnomaly	0.394	0.190	0.256	0.353	0.332	0.342
NeuralLog	0.297	0.354	0.323	0.638	0.872	0.736
Our method	0.738	1.00	0.850	0.857	1.00	0.923

tion environment.

The complexity of Bob is much less than that of Ada, mainly because of less templates and a relatively fixed log sequence. However, due to problems such as network latency and data washout, the logs generated by the switch have a large number of errors and retry messages. In most cases, these error messages are not of concern because the program can be restored to normal after several retries and will not have a significant impact on the execution of the business. Also, these alarms of variable duration can disrupt the log sequence, making it more difficult to learn the normal pattern of the logs purely from frequency or sequence. The sequence confusion greatly affects the learning ability of LogAnomaly on this dataset, and NeuralLog has a lower Precision due to the report of the unimportant exceptions. Our framework captures the normal characteristics of logs from multiple perspectives and designs alerting policies based on the frequency and periodicity of logs, thus achieving good performance on the Bob dataset.

The experimental results show that our framework has good performance for complex log datasets in practice. The analysis of log semantics makes the model show good robustness on poor stability sequences, and the addition of the alert policy layer reduces the model' s disturbance to engineers and screens out some of the exceptions that can be fixed automatically.

2) RQ2: How effective is the sentence vector generation and alerting strategy layer in improving the effectiveness of the model?

As mentioned in the previous sections, we add the sentence vector matrix as well as the alert alarm strategy layer to GPT-2. In this RQ, we will verify the effect of the main parts of the framework on its effectiveness, by removing one or two components, namely the sentence vector (SV) and the alarm strategy (AS).

OM w/o SV & AS: We remove the sentence vector generation and alert policy layers from our framework. That is, we use only the GPT-2 model for sequence prediction to diagnose anomaly logs.

OM w/o AS: We remove the alarm strategy generation from our framework.

OM w/o SV: We remove the sentence vector layer from our framework.

OM: Anomaly detection work on logs using the completed framework proposed in this paper.

The experimental results in Table 4 show that the sentence vector generation part of the framework can improve the accuracy of the model to some extent and greatly enhance Recall. Because GPT-2 achieves better results in capturing the semantic information of normal and abnormal templates after receiving the sentence vectors of the templates as prior knowledge. This is demonstrated by the fact that log templates containing the same abnormal keywords, the vector representations of which have a closer distance, are easily detected together in the anomaly detection stage. At the same time, log templates that symbolize normal patterns are more difficult for the model to detect as false positives because they often contain positive-meaning words. Since the alarm strategy layer is built based on expert experience and has accurate filtering rules, it can filter out a large number of false abnormal logs and effectively improve the Recall of the model.

With the inclusion of both components, the effectiveness of our framework has been significantly improved. For complex logging environments, more accurate exception identification, very low FPs and a fairly high Recall can be achieved.

5 Related Work

As a kind of operational data, logs are widely used for system anomaly detection in practice. To take advantage of the logs, previous work mainly focuses on detecting abnormal logs with artificial rules, which is not appropriate in scenarios with a large number of logs[5]. And deep learning is widely used in automatic anomaly detection in logs. DeepLog[3] uses LSTM to learn the normal pattern of the system and predict the next log template by log sequence. LogAnomaly[5] uses a word embedding model to mine semantic information of log templates and learn the sequential patterns and quantitative relationships for logs with LSTM. LogRobust[13] represents the semantic information of the log by word vector and takes advantage of the bi-directional LSTM to learn the normal pattern of the log. OneLog[14] merges components (such as parsers and classifiers) into a deep neural network to detect log anomalies. LogMerge[15] learns the semantic similarity of multi-syntax logs to realize the transfer of log exception patterns across log types, which greatly reduces the overhead of exception annotation. Transformers could also be used to represent the semantics of the log and model the log sequence, and anomalies would be detected with learned information[10, 15 – 17]. GPT-2[10] is proposed for unsupervised learning of text information based on transformers and performs well on text generation, text classification, semantic judgment, etc.

6 Conclusions and Future Work

In practical scenarios, large-scale systems produce logs that are different from the vast majority of laboratory open-source datasets. The log sequence is more complex, and normal patterns are harder to capture. In this work, we introduced a way

▼Table 4. Experimental results

Approach	Ada			Bob		
	Precision	Recall	F1S	Precision	Recall	F1S
OM w/o SV & AS	0.128	0.835	0.222	0.510	0.940	0.661
OM w/o AS	0.427	1.00	0.598	0.718	1	0.836
OM w/o SV	0.627	0.807	0.705	0.833	0.940	0.883
OM	0.738	1.00	0.850	0.857	1.00	0.923

AS: alarm strategy SV: sentence vector
OM: our method

to input semantic analysis data into GPT-2 and designed an alarm strategy layer. Through these ways, we improved the complex log sequence learning ability of the model and reduced the noise effect on the model prediction. Experimental results on two industrial datasets have shown that the false alarm rate of the model is significantly reduced, and our framework shows good performance in the actual operation scenario.

In the future, we will continue to improve the performance of the model on multiple datasets and reduce the dependence of the alarm strategy layer on expert experience.

References

[1] ZHANG S L, LIU Y, PEI D, et al. Rapid and robust impact assessment of software changes in large Internet-based services [C]//The 11th ACM Conference on Emerging Networking Experiments and Technologies. ACM, 2015: 1 – 13. DOI: 10.1145/2716281.2836087

[2] ZHU J M, HE S L, LIU J Y, et al. Tools and benchmarks for automated log parsing [C]//IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). IEEE, 2019: 121 – 130. DOI: 10.1109/ICSE-SEIP.2019.00021

[3] DU M, LI F F, ZHENG G N, et al. DeepLog: anomaly detection and diagnosis from system logs through deep learning [C]//ACM SIGSAC Conference on Computer and Communications Security. ACM, 2017: 1285 – 1298. DOI: 10.1145/3133956.3134015

[4] MENG W B, LIU Y, ZHU Y C, et al. LogAnomaly: unsupervised detection of sequential and quantitative anomalies in unstructured logs[C]//The 28th International Joint Conference on Artificial Intelligence. ACM, 2019, 19(7): 4739 – 4745

[5] ZHANG X, XU Y, LIN Q W, et al. Robust log-based anomaly detection on unstable log data [C]//The 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ACM, 2019: 807 – 817. DOI: 10.1145/3338906.3338931

[6] EKELHART A, EKAPUTRA F J, KIESLING E. The SLOGERT framework for automated log knowledge graph construction [C]//European Semantic Web Conference. ESWC, 2021: 631 – 646. DOI: 10.1007/978-3-030-77385-4_38

[7] GUO H X, YUAN S H, WU X T. LogBERT: log anomaly detection via BERT [C]//Proceedings of 2021 International Joint Conference on Neural Networks (IJCNN). IEEE, 2021: 1 – 8. DOI: 10.1109/IJCNN52387.2021.9534113

[8] LE V H, ZHANG H Y. Log-based anomaly detection without log parsing [C]//The 36th IEEE/ACM International Conference on Automated Software Engineering. ACM, 2021: 492 – 504. DOI: 10.1109/ASE51524.2021.9678773

[9] HE S L, HE P J, CHEN Z B, et al. A survey on automated log analysis for reliability engineering [J]. ACM computing surveys, 54(6): 1 – 37. DOI: 10.1145/3460345

[10] RADFORD A, WU J, CHILD R, et al. Language models are unsupervised multitask learners [EB/OL]. [2023-03-10]. https://www.semanticscholar.org/paper/Language-Models-are-Unsupervised-Multitask-Learners-Radford-Wu/9405cc0d6169988371b2755e573cc28650d14dfe

[11] HE P J, ZHU J M, ZHENG Z B, et al. Drain: an online log parsing approach with fixed depth tree [C]//IEEE International Conference on Web Services (ICWS). IEEE, 2017: 33 – 40. DOI: 10.1109/ICWS.2017.13

[12] REIMERS N, GUREVYCH I. Sentence-BERT: sentence embeddings using Siamese BERT-networks [C]//Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing. Association for Computational Linguistics, 2019: 3982 – 3992. DOI: 10.18653/v1/d19-1410

[13] HASHEMI S, MÄNTYLÄ M. OneLog: towards end-to-end training in software log anomaly detection [EB/OL]. [2022-12-12]. https://arxiv.org/abs/2104.07324

[14] CHEN R, ZHANG S L, LI D W, et al. LogTransfer: cross-system log anomaly detection for software systems with transfer learning [C]//IEEE 31st International Symposium on Software Reliability Engineering. IEEE, 2020: 37 – 47. DOI: 10.1109/ISSRE5003.2020.00013

[15] HUANG S H, LIU Y, FUNG C, et al. HitAnomaly: Hierarchical transformers for anomaly detection in system log [J]. IEEE transactions on network and service management, 2020, 17(4): 2064 – 2076. DOI: 10.1109/TNSM.2020.3034647

[16] YANG H T, ZHAO X, SUN D G, et al. Sprelog: log-based anomaly detection with self-matching networks and pre-trained models [C]//International Conference on Service-Oriented Computing. 2021: 736 – 743

[17] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is all you need [C]//Proceedings of the 31st International Conference on Neural Information Processing Systems. ACM, 2017: 6000 – 6010. DOI: 10.5555/3295222.3295349

[18] LE V H, ZHANG H Y. Log-based anomaly detection with deep learning: how far are we? [C]//IEEE/ACM 44th International Conference on Software Engineering (ICSE). IEEE, 2022: 1356 – 1367

Biographies

JI Yuhe received his bachelor's degree in software engineering from the College of Software, Nankai University, China in 2022. He is now pursuing his master's degree at the School of Software, Nankai University. His research interests include anomaly detection and natural language processing.

HAN Jing (han.jing28@zte.com.cn) received her master's degree from Nanjing University of Aeronautics and Astronautics, China. She has been with ZTE Corporation since 2000. She had been engaged in 3G/4G key technologies, from 2000 to 2016, and has become a technical director responsible for intelligent operation of cloud platforms and wireless networks since 2016. Her research interests include machine learning, data mining, and signal processing.

ZHAO Yongxin received her bachelor's degree in software engineering from Nankai University, China in 2021. She is currently pursuing her master's degree at the School of Software, Nankai University. Her research interests include anomaly detection and failure diagnosis.

ZHANG Shenglin received his BS degree in network engineering from the School of Computer Science and Technology, Xidian University, China in 2012 and PhD degree in computer science from Tsinghua University, China in 2017. He is currently an associate professor with the College of Software, Nankai University, China. His current research interests include failure detection, diagnosis and prediction for service management. He is an IEEE Member.

GONG Zican received his master's degree in professional computing and artificial intelligence from the Australian National University in 2019. He has been a machine learning engineer in ZTE Corporation since 2020. His research interests include machine learning, professional computing and system architecture.