2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)
October 11-14, 2020. Toronto, Canada

Anomaly Detection in Distributed Systems via Variational Autoencoders

1st Yun Qian
School of Computer Science
Wuhan University
Wuhan, China.
qianyun@whu.edu.cn

2nd Shi Ying
School of Computer Science
Wuhan University
Wuhan, China.
yingshi@whu.edu.cn

3rd Bingming Wang
School of Computer Science
Wuhan University
Wuhan, China.
wbingming@whu.edu.cn

***Abstract*—Distributed systems have been widely used in the information technology industry. However, with the increasing scale and complexity of distributed systems, the efficiency and accuracy of manual anomaly detection in system logs have decreased. Therefore, there is a great demand for a highly accurate and efficient automatic anomaly detection method based on system log analysis to ensure the reliability and the stability of large-scale distributed systems. In this paper, we propose VeLog, an automatic anomaly detection method based on variational autoencoders (VAEs). In the offline training phase, VeLog learns the patterns of normal log sequences and then generates normal intervals. In the online detection phase, VeLog detects an anomaly by automatically evaluating whether the distance between the input vector and its estimated vector matches these normal intervals. We evaluate VeLog on log datasets collected from representative distributed systems. The experimental results demonstrate that VeLog can detect anomalies with high accuracy and good efficiency.**

***Index Terms*—Anomaly detection, deep learning, distributed systems, log data analysis**

I. Introduction

Log analysis is important for system debugging and optimization. Nowadays, distributed systems (e.g., Hadoop [1] and OpenStack [2]), which are currently widely used in industry, often involve thousands of nodes and provide critical business services. It is unrealistic to monitor and analyze all the log files of these systems by relying only on experts. Consequently, an automatic, efficient and accurate log-based anomaly detection method is urgently needed.

Over the years, many supervised methods have been proposed to detect anomalies in systems [3]–[5]. These methods derive models from prelabeled data that can subsequently be used to detect other unlabeled data. When training data is balanced and sufficient, the detection results of these methods are often highly accurate. However, in practice, it is difficult and expensive to acquire the abnormal logs of a service-provided system. Therefore, despite the good performance of supervised methods, these methods are not as popular as semisupervised or unsupervised methods for anomaly detection, owing to the lack of availability of obtaining sufficient abnormal data [6]–[8].

In addition to supervised methods, many unsupervised anomaly detection methods have been proposed [9]–[11]. These methods avoid the need for labeled data; instead, they infer anomalies from the intrinsic properties of unlabeled data. However, unsupervised anomaly detection methods are sometimes sensitive to noise and data corruption. These methods are often less accurate than supervised or semisupervised methods [6], [8], [12].

Compared with supervised and unsupervised methods, semisupervised methods can also achieve high accuracy and require only normal logs for training. Thus, in this paper, we propose a novel semisupervised log-based anomaly detection method, VeLog. The main contributions of this paper are as follows:

1) We propose VeLog, a semisupervised log-based anomaly detection method based on variational autoencoders (VAEs). VeLog can detect anomalies in distributed systems efficiently and accurately.
2) We adopt a novel feature extraction method that considers both the orders of logs in their execution paths and the number of log execution times.
3) We use log datasets collected from representative distributed systems to evaluate VeLog and other anomaly detection methods. The experimental results confirm that VeLog outperforms other methods.

The remainder of this paper is organized as follows: We introduce the preliminaries of our work in Section II and describe our method in Section III. Section IV presents our experimental design and results. Section V surveys related work. Section VI concludes this paper and discusses our future work.

II. Preliminaries

A. Log Parsing

Before engaging in anomaly detection, we must parse unstructured, free-text logs into a structured representation. This process is called log parsing.

In general, logs consist of constant and variable parts. For example, given the logs of "Receiving block blk_1608999687 919862906 src:/10.251.215.16:52002 dest:/10.251.215.16:500 10" and "Receiving block blk_7503483334202473044 src:/10 .251.71.16:51590 dest:/10.251.71.16:50010", the words "Receiving", "block", "dest", and "src" are considered to be constant parts, while the remaining parts are variable parts.

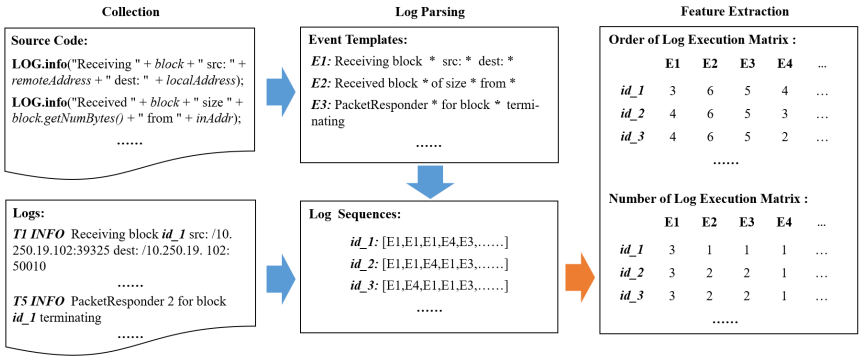


Fig. 1. Framework of log parsing and feature extraction.

Constant parts are predefined in source codes by developers, whereas variable parts (e.g., port number, IP address) are often generated dynamically based on system deployment environments. We do not discuss variable parts in this paper because system monitoring tools or frameworks are more suitable for monitoring and analyzing variable parts [13].

Therefore, similar to previous works on log parsing [14]–[16], we separate constant parts from variable parts and form structured log events (e.g., "Receiving block * src: * dest: *"). The log event of a log always describes the key information of this log. In addition, most system logs are recorded using thread or instance IDs to distinguish logs of different threads or workflows. Thus, we retain specific parameters to identify the specific threads or workflows; these parameters are called identifiers, such as the *block_id* in the Hadoop distributed file system (HDFS) and the *instance_id* in OpenStack. These identifiers can group logs into separate log sequences [9], [10], [17].

The log parsing in our work is shown in Fig. 1. First, we extract the log event templates based on the log print statements in the source code. Then, according to the identifiers, different logs are divided into different log sequences. Each log sequence represents a single execution path identified by a unique identifier. Finally, VeLog uses a longest common subsequence algorithm (LCS [18]) to identify the log events of logs in each log sequence based on log event templates.

B. Feature Extraction of Log Sequences

The input for feature extraction is log sequences generated in the log parsing step. A log sequence generated by a long-running system sometimes contains hundreds of log events. The goal of feature extraction is to extract valuable representative features from log sequences.

To accomplish this, many methods set a window based on the timestamp to divide a long log sequence into finite chunks to avoid high-dimensional input [19]. According to the experiment reported in [14], setting different sizes of windows (e.g., 1 hour, 3 hours, 6 hours, etc.) will affect the performance of anomaly detection models. In many cases, it is difficult to manually set an appropriate window size. Therefore, in our work, we set the dimensionality according to the types of log events to avoid high-dimensional input.

Log events that occur in the normal lifecycle of a task or a virtual machine (VM) are often scheduled by using a regular expression pattern. For example, in OpenStack, a VM life cycle starts with "VM create" and ends with "VM delete", while task pairs, such as "Stop-Start", "Pause-Unpause" and "Suspend-Resume" may randomly appear from 0 to 3 times within a life cycle [20]. In other words, in a normal lifecycle of a task or a VM, the order and number of each log execution must be within the normal range. Consequently, in this work, we save information on both the order and number of each log execution in a log sequence.

The output of the feature extraction is shown in Fig. 1. In these matrices, each row is generated by a log sequence identified by an identifier, and each column represents a specific type of log event. If a row in an order of log execution matrix is $[1, 2, 3, 5, \ldots]$, in this log sequence, event 1 was executed first, event 2 was executed second, etc. If a log event is executed more than once, we record only the order of its last execution. If a row in a number of log execution matrix is $[1, 0, 2, 0, \ldots]$, in this log sequence, event 1 occurred once, event 3 occurred twice, etc. In this way, we retain nearly complete information on log sequences while simultaneously avoiding manually setting the window based on timestamp; manually setting the window based on timestamp might truncate complete log sequences and lead to false positives.

III. VeLog: A Log-based Anomaly Detection Method

To improve the accuracy and efficiency of anomaly detection in distributed systems, we propose VeLog. An overview of VeLog is shown in Fig. 2. In general, VeLog can be divided into two phases: an offline training phase and an online detection phase.

Offline Training Phase: The training data is collected from logs generated during the normal operation of the system. Through log parsing and feature extraction, VeLog obtains the order and number of log execution matrices. Then, these matrices are used to train the relevant anomaly detection models.

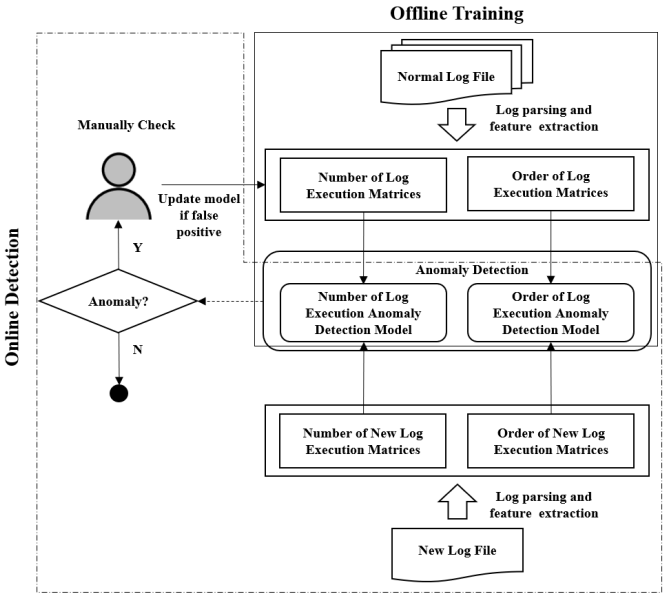


Fig. 2. The architecture of VeLog.

Online Detection Phase: When the system outputs a new log file, VeLog also needs to generate the order and number of log execution matrices through log parsing and feature extraction. Next, VeLog inputs these matrices into the relevant anomaly detection models. A new log sequence will be labeled as an anomaly if the order of log execution vector and number of log execution vector of this sequence are predicted to be abnormal. VeLog also provides the option of collecting user feedback. If the user reports a detected anomaly as a false positive, VeLog can use this anomaly as a labeled record to update anomaly detection models.

A. Deep Generative Model: Variational Autoencoder

The main task of anomaly detection from system logs is identifying abnormal log sequences that do not match the patterns of normal log sequences. To learn the patterns of normal log sequences, we adopt VAEs in VeLog.

Compared to other deep generative models, VAE also makes an approximation of the input, but the error introduced by this approximation is arguably small given high-capacity models [21]. In addition, VAEs have been used to generate many types of complicated data, including handwritten digits and faces. Thus, we use VAEs to learn the underlying probability distribution and output approximations of the training data.

The idea of VAE is to learn a low-dimensional latent representation of the training data, called latent variables, which are not directly observed but rather are inferred through a mathematical model. Consider a dataset $X = \{x^{(i)}\}_{i=1}^{N}$, consisting of N independent and identically distributed (i.i.d.) samples of some variable x. We hypothesize that the variable x is generated by an unobserved continuous random variable z.

VAE consists of two neural networks, as shown in Fig. 3: an encoder (denoted by $q_\varphi(z|x)$) and a decoder (denoted by $p_\theta(x|z)$). φ and θ represent the parameters of the encoder and decoder, respectively. Because the true posterior $p_\theta(z|x)$ is intractable, VAE uses the encoder $q_\varphi(z|x)$ to approximate the true posterior [21], [22].

In this work, we assume that $p_\theta(x|z)$ is a multivariate Gaussian whose distribution parameters are computed from z by using the multi-layer perceptron (MLP). The prior over the latent variables is the centered isotropic multivariate Gaussian $p_\theta(z) = \mathcal{N}(z; 0, I)$. We also assume the true posterior takes on an approximate Gaussian form with an approximately diagonal covariance. In this case, we can let the variational approximate posterior be a multivariate Gaussian with a diagonal covariance structure as follows [22]:

$$\log q_\varphi\left(z|x^{(i)}\right) = \log \mathcal{N}\left(z; \mu^{(i)}, \sigma^{2(i)} I\right) \tag{1}$$

where the mean $\mu^{(i)}$ and standard deviation $\sigma^{(i)}$ are outputs of the encoding MLP.

Applying the "reparameterization trick" from Kingma et al. [22], we can sample from the posterior $z^{(i,l)} \sim q_\varphi\left(z|x^{(i)}\right)$ by using $z^{(i,l)} = \mu^{(i)} + \sigma^{(i)} \odot \epsilon^{(l)}$, where $\epsilon^{(l)} \sim \mathcal{N}(0, I)$. Let J be the dimensionality of z and let $\mu_j^{(i)}$ and $\sigma_j^{(i)}$ simply denote the j-th element of these vectors. Finally, the resulting estimator for this model and $x^{(i)}$ is as follows:

$$\mathcal{L}\left(\theta, \varphi; x^{(i)}\right) \simeq \frac{1}{2}\sum_{j=1}^{J}\left(1 + \log((\sigma_j^{(i)})^2 - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2)\right) + \frac{1}{L}\sum_{l=1}^{L}\log p_\theta(x^{(i)}|z^{(i,l)}) \tag{2}$$

where $z^{(i,l)} = \mu^{(i)} + \sigma^{(i)} \odot \epsilon^{(l)}$ and $\epsilon^{(l)} \sim \mathcal{N}(0, I)$.

The process by which VeLog performs anomaly detection based on VAEs will be described in Section III-B.

B. Anomaly Detection

In VeLog, anomaly detection consists of two components: the order of log execution anomaly detection model and the number of log execution anomaly detection model.

As shown in Section IV-D, by combining these two aspects of anomaly detection results, we can detect nearly 99% of anomalies in execution paths.

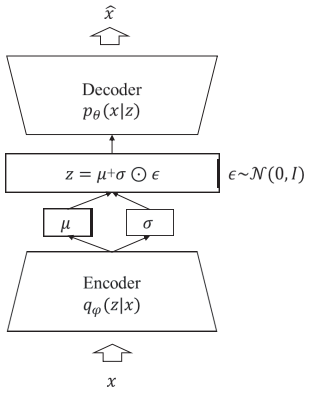


Fig. 3. The architecture of a variational autoencoder.

1) Order of Log Execution Anomaly Detection Model: By using the order of log execution anomaly detection model, we can detect anomalies with abnormal orders of logs in their execution processes. To describe this model conveniently, we represent the order of log execution matrices as a set K. Each row of K is an order of log execution vector generated by a log sequence.

In the offline training phase, we denote $K_{train} = \{k^{(i)}\}_{i=1}^{N}$ as the training set and $k^{(i)}$ as an order of log execution vector generated by a normal log sequence. The index i of $k^{(i)}$ corresponds to its log sequence identifier. The total number of vectors in the training set is N. The training flow for the order of log execution anomaly detection model follows 4 steps: 1) normalize $K_{train} = \{k^{(i)}\}_{i=1}^{N}$ and input these vectors to the VAE; 2) calculate the approximations of the input vectors and anti-normalize them to obtain their estimated vectors $\widehat{K_{train}} = \{\widehat{k^{(i)}}\}_{i=1}^{N}$; 3) calculate the Euclidean distances between the estimated and the input vectors $Dis\_K_{train} = \{(k^{(i)} - \widehat{k^{(i)}})^2\}_{i=1}^{N} = \{d^{(i)}\}_{i=1}^{N}$; and 4) obtain the final set of normal intervals. To begin with, Dis_K_{train} is divided into different intervals, where the length of each interval is α. The total number of intervals S is calculated by (3). The p-th interval G_p is denoted as (4), and the result from dividing Dis_K_{train} into G_p is denoted as (5). If $Q_p \neq \varnothing$, the p-th final normal interval $R_p = G_p$, $p = 1, 2, \ldots, S$. If $Q_p = \varnothing$, the p-th final normal interval $R_p = \varnothing$, $p = 1, 2, \ldots, S$. Finally, the final set of normal intervals is obtained by (6).

$$S = \frac{d_{\max} - d_{\min}}{\alpha} \tag{3}$$

$$G_p = \left[d_{\min} + (p-1)\,\alpha, d_{\min} + p\alpha\right], p = 1, 2, \ldots, S \tag{4}$$

$$\begin{aligned} Q_p = &\{d \in Dis\_K_{train} \mid d_{\min} + (p-1)\alpha \leq d \leq d_{\min} \\ &+ p\alpha\}, p = 1, 2, \ldots, S \end{aligned} \tag{5}$$

$$Nor\_G = \bigcup_{p=1}^{S} R_p \tag{6}$$

where $d_{\min}$ is the minimal value and $d_{\max}$ is the maximal value of Dis_K_{train}.

Because VeLog trains the order of log execution anomaly detection model by using the order of log execution matrices generated by normal log sequences, the VAE learns only how to reconstruct the order of log execution vector generated by the normal log sequence. In the online detection phase, when facing an input vector generated by a normal log sequence, because the VAE was trained in advance, VeLog assumes that the VAE can ably reconstruct this input well. The Euclidean distance between this input and estimated vector will fall within the normal intervals generated during the offline training phase. However, when facing an input vector generated by an abnormal log sequence, because that sequence differs from a normal log sequence and the VAE was not trained on abnormal order of log execution matrices in advance, VeLog assumes that the Euclidean distance between this input and estimated vector will not fall within the normal intervals.

We denote $K_{detect} = \left\{k^{(j)}\right\}_{j=1}^{M}$ as the test set and $k^{(j)}$ be an order of log execution vector generated by a log sequence to be detected. The index j of $k^{(j)}$ corresponds to its log sequence identifier. The total number of vectors in the test set is M. To detect j-th log sequence, the detection flow for the order of log execution anomaly detection model also follows 4 steps: 1) normalize ${k_{detect}}^{(j)}$ and input it to the VAE; 2) calculate its approximation and anti-normalize it to obtain its estimated vector; 3) calculate its Euclidean distance; and 4) determine whether its Euclidean distance falls within the normal intervals. When its Euclidean distance falls within the normal intervals, this model judges its log sequence as normal; otherwise, this model judges its log sequence to be an anomaly.

2) Number of Log Execution Anomaly Detection Model: The design of the number of log execution anomaly detection model is similar to that of the order of log execution anomaly detection model. Using the number of log execution anomaly model, we can detect anomalies with abnormal log execution times. To describe this model conveniently, we represent the number of log execution matrices as a set H. Each row of H is a number of log execution vector generated by a log sequence.

In the offline training phase, we denote $H_{train} = \left\{h^{(i)}\right\}_{i=1}^{N}$ as the training set and $h^{(i)}$ as a number of log execution vector generated by a normal log sequence. The index i of $h^{(i)}$ also corresponds to its log sequence identifier. The total number of vectors in the training set is N. The training flow for the number of log execution anomaly detection model follows 4 steps: 1) normalize $H_{train} = \{h^{(i)}\}_{i=1}^{N}$ and input these vectors to the VAE; 2) calculate the approximations of these vectors and anti-normalize them to obtain their estimated vectors $\widehat{H_{train}} = \{\widehat{h^{(i)}}\}_{i=1}^{N}$; 3) calculate the Euclidean distances between the estimated and the input vectors $Dis\_H_{train} = \{(h^{(i)} - \widehat{h^{(i)}})^2\}_{i=1}^{N} = \{d^{(i)}\}_{i=1}^{N}$; and 4) obtain the final set of normal intervals (this step is similar to step 4 of the procedure followed by the order of log execution anomaly detection model during the offline training phase).

Based on the same idea of the order of log execution anomaly detection model, during the online detection phase, we denote $H_{detect} = \left\{h^{(j)}\right\}_{j=1}^{M}$ as the test set and $h^{(j)}$ as a number of log execution vector generated by a log sequence to be detected. The index j of $h^{(j)}$ corresponds to its log sequence identifier. The total number of vectors in the test set is M. To detect j-th log sequence, the detection flow for the number of log execution anomaly detection model follows also 4 steps: 1) normalize ${h_{detect}}^{(j)}$ and input it to the VAE; 2) calculate its approximation and anti-normalize it to obtain its estimated vector; 3) calculate its Euclidean distance; and 4) determine whether its Euclidean distance falls within the normal intervals. If its Euclidean distance falls within the normal intervals, this model judges its log sequence to be normal; otherwise, this model judges its log sequence to be

an anomaly.

C. Updating Anomaly Detection Models

Since VeLog applies the log sequences confirmed as normal in recent log files for training, it is more likely to classify a newly-emerged log sequence as abnormal. Thus, if the detection results obtained by the order and number of log execution anomaly detection models in evaluating a log sequence are abnormal, this log sequence will be considered to be an anomaly, and the log sequence will be sent to the users with its corresponding logs for verification. If the verification result of this log sequence is also abnormal, VeLog does not need to update anomaly detection models; otherwise, VeLog needs to add this falsely detected log sequence to the training set and retrain these models.

IV. Evaluation

In this section, we evaluate VeLog on two large datasets. We begin by describing the datasets and the baseline methods. Next, we introduce the implementation and evaluation metrics used in this work. Finally, we present experimental results from VeLog and compare the performance of VeLog with other methods.

A. Datasets

HDFS log dataset: The HDFS log dataset is a commonly used dataset for log-based anomaly detection [9], [11], [20]; the dataset was generated by running Hadoop on more than 200 Amazon Elastic Compute Cloud (EC2) nodes. In total, the dataset contains 24,396,061 log messages. These log files contain different log sequences and are identified by *block_id*. We randomly chose 3,000 distinct normal log sequences (containing 58,464 logs) from the HDFS log dataset as a training set for semisupervised methods. We also randomly chose 6,000 abnormal log sequences (containing 102,090 logs) and 6,000 normal log sequences (containing 117,407 logs) from the HDFS log dataset as a test set.

OpenStack log dataset: This dataset was generated by running OpenStack. More details about this dataset can be found in [23]. These log files contain different log sequences identified by *instance_id*. We randomly chose 400 distinct normal log sequences (containing 9,226 logs) from the OpenStack log dataset as a training set. We also randomly chose 100 abnormal log sequences (containing 1,539 logs) and 600 normal log sequences (containing 13,839 logs) from the original OpenStack log dataset as an imbalanced test set.

B. Baseline Methods

Depending on whether the labeled data is used for training, we can classify the anomaly detection methods into supervised, semisupervised and unsupervised methods. Despite the high accuracy of supervised methods, these methods are not as popular as semisupervised or unsupervised methods in practice, because it is difficult to collect all types of abnormal logs as training data and the accuracy of these methods is affected by the imbalanced training data. Finally, we choose three representative anomaly detection methods as the baseline methods, including two unsupervised methods (i.e., Principal Component Analysis (PCA) [9] and Log Clustering [11]) and a semisupervised method (i.e., DeepLog [20]) .

C. Implementation and Evaluation Metrics

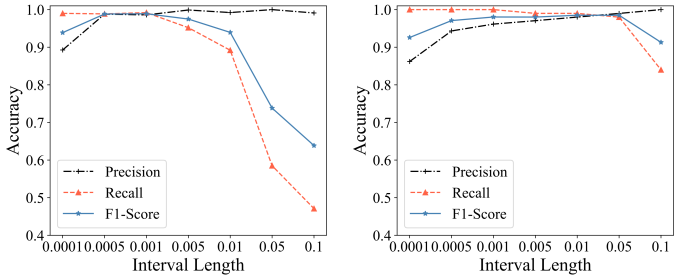
To begin our experiments, we parse logs and extract features based on the different input requirements of the baseline methods and VeLog. Then, we set the required parameters of the methods to the default or recommended values and report the best results of the methods on two datasets.

The metrics used to evaluate the accuracy of the results are precision, recall and F1-Score as metrics. Precision is the percentage of log sequences correctly classified as anomalies among all the log sequences that are classified as anomalies. Recall is the percentage of log sequences correctly identified as anomalies over all the abnormal log sequences. The F1-Score is the harmonic mean of precision and recall. These metrics are calculated as $Precision = \frac{TP}{TP+FP}$, $Recall = \frac{TP}{TP+FN}$, and $F1-Score = \frac{2 \cdot Precision \cdot Recall}{Precision+Recall}$, where TP is the number of abnormal log sequences correctly detected by the model, FP is the number of normal log sequences incorrectly identified as anomalies by the model, and FN is the number of abnormal log sequences that are not detected by the model.

We conduct all the experiments with an Intel Core 2.20 GHz CPU and 16G memory. We implement VeLog and DeepLog with Python 3.6.5 and PyTorch 1.2.0. For Log Clustering and PCA, we use the popular open-source toolkit implemented in [14]. For the encoders and the decoders of the VAEs in VeLog, we adopt neural networks with simple architectures, namely MLPs. The input and output sizes of VAEs depend on the number of log events. The number of layers in encoders and decoders are 3. We train the VAEs in VeLog by using the Adaptive Moment Estimation (ADAM) optimizer [24] with a learning rate of 0.0001 and a batch size of 128. By default, we set $\alpha = 0.001$ (where α is the interval length) for the HDFS log dataset and $\alpha = 0.01$ for the OpenStack log dataset. We also investigate the accuracy impacts of different interval lengths in our experiments.

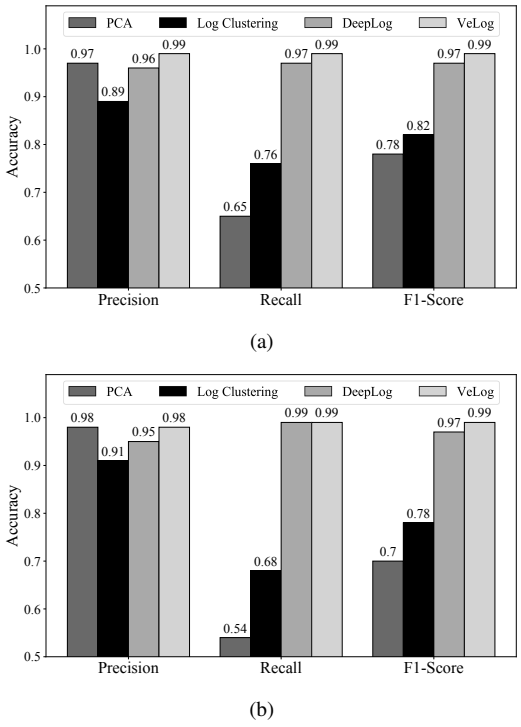
D. Experimental Results

1) Analysis of VeLog: The accuracy of VeLog is influenced by the interval length. Thus, we investigate the accuracy impacts of different interval lengths. Fig. 4(a) shows the accuracy of VeLog with different interval lengths on the HDFS log dataset, while Fig. 4(b) shows the accuracy of VeLog with different interval lengths on the OpenStack log dataset. In general, a smaller interval length always leads to higher recall but lower precision, and a larger interval length always leads to lower recall but higher precision. Overall, VeLog achieves the best accuracy on the HDFS log dataset when α is between 0.0005 and 0.001. In addition, VeLog achieves the best accuracy on the OpenStack log dataset when α is between 0.01 and 0.05. However, when $\alpha = 0.1$, VeLog achieves the worst accuracy on two datasets; specifically, VeLog achieves



(a) (b)

Fig. 4. Accuracy of VeLog with different interval lengths. (a) On the HDFS log dataset. (b) On the OpenStack log dataset.



(a)

(b)

Fig. 5. Accuracy of different anomaly detection methods on log datasets. (a) On the HDFS log dataset. (b) On the OpenStack log dataset.

an F1-Score of 0.639 on the HDFS log dataset and an F1-Score of 0.913 on the OpenStack log dataset.

These results are easily understood. If we set the value of the interval length α too low, although the model will detect as many anomalies as possible, it will become easy to misreport normal logs as abnormal (i.e., there will be more false positives), thus producing detection results with low precision. If we set the value of the interval length α too high, although the detection precision will improve, many anomalies will be ignored ((i.e., there will be more false negatives). Therefore, before actual deployment, we suggest testing VeLog on several log files with different values of α to obtain an appropriate value of the interval length.

2) Comparisons: In this section, we compare VeLog with the three baseline methods on two datasets. Fig. 5(a) and Fig. 5(b) show the accuracy of the different anomaly detection methods on two log datasets. Compared with the baseline methods, VeLog achieves the best accuracy on two log datasets; specifically, VeLog achieves an F1-Score of 0.99 on the HDFS log dataset and an F1-Score of 0.99 on the OpenStack log dataset. DeepLog also achieves good accuracy and is second only to VeLog. The respective accuracy of PCA and Log Clustering is accompanied by high precision but low recall; therefore, many anomalies are ignored. And the recall of them on imbalanced log dataset (i.e., on the OpenStack log dataset) is worse than the recall of them on balanced log dataset (i.e., on the HDFS log dataset).

TABLE I
FOUR TYPES OF ANOMALIES AND THEIR DETECTION RESULTS

Cat.	Anomaly Description	ACT[a]	VeLog	DeepLog
1	Delete a block that is not found in volumeMap	58	56	57
2	Received block that does not belong to any file	144	139	133
3	Write a block with exception	39	39	39
4	Redundant addStoredBlock	71	71	68
Total		312	305	297

[a] the number of actual anomalies labeled by experts.

If we can detect system anomalies as quickly as possible, then we can minimize the loss caused by system anomalies. Thus, for VeLog, we compared the average time cost of detecting per log sequence by using the three baseline methods. In the HDFS and OpenStack log datasets, the length of each log sequence usually ranges from 10 to 290. As shown in Fig. 6, PCA achieves the best efficiency and its average time cost of detecting per log sequence typically less than 0.05 milliseconds. The efficiency of VeLog is second only to PCA; for VeLog, the average time cost of detecting per log sequence is typically less than 0.56 milliseconds.The efficiency of DeepLog is affected by the lengths of detected log sequences and the parameter values of its long short-term memory (LSTM) neural network. For DeepLog, under the premise of high accuracy, the average time cost of detecting per log sequence usually more than 10 milliseconds. For Log Clustering, the average time cost of detecting per log sequence is typically less than 8.7 milliseconds, and the efficiency of Log Clustering is mostly affected by the size of the logs because the time complexity of the Log Clustering is $O(n^2)$.

3) Analysis of the Experimental Results: As shown in Fig. 5, VeLog achieves the best overall accuracy on two log datasets. VeLog achieves such accuracy principally because this method adopts normal intervals rather than a single threshold as the discriminating standard to identify anomalies. In addition, we retain nearly entire information on long log sequences as the input data for the VAEs through our feature extraction method. If, like DeepLog, we set a window based on the timestamp to divide long log sequences into finite chunks, some anomalies that happened in the current window may actually be related to events in the former time window; consequently, setting a window based on the timestamp may

break some important dependency between log events in a long log sequence. Therefore, VeLog can obtain better accuracy than DeepLog. In addition, the poor accuracy of Log Clustering is caused mainly by the high-dimensional and sparse characteristics of event count matrix. The poor accuracy of PCA is why both classes (normal and abnormal) are difficult to separate naturally by using any single threshold [14].

To further evaluate the accuracy of VeLog, we deploy VeLog and DeepLog to detect four representative anomaly cases that happened on HDFS. Table I shows four typical types of anomalies and their detection results. As shown in Table I, the third type of anomaly is detected accurately by VeLog and DeepLog, because this type of anomaly always occurs with obvious keywords like "java.io.IOException" and short lengths of log sequences. However, keyword matching (e.g., searching for words like Exception , Error, or Warn) is not a good way of detecting anomalies. For example, in HDFS DataNode logs, we see many warning messages, such as "*: Got Exception while serving * to *". However, this behavior by HDFS is normal [9]. While traditional keyword matching would have labeled these messages as anomalies, both DeepLog and VeLog can successfully avoid this kind of false positive results because these methods are trained by this kind of normal log messages in advance. The first, second and fourth types of anomalies in Table 1 are the most easily ignored anomalies in PCA and Log Clustering mainly because these anomalies do not occur with noticeable keywords or short lengths of log sequences. The log sequences generated by these anomalies usually have only several log events that are different from normal log sequences. VeLog and DeepLog are good at detecting these types of anomalies and VeLog is slightly more accurate than DeepLog.

During system runtime, the number of accumulated new log events can become increasingly large. Thus, the anomaly detection method must be capable of detecting long sequences and obtain results as quickly as possible. As shown in Fig. 6, PCA is most efficient. However, PCA is difficult to apply to real work because of its not high accuracy. The efficiency of VeLog is second only to that of PCA, and the high accuracy of VeLog also satisfies the requirements of real work.

V. Related Work

In practice, logs have been widely used for the monitoring and diagnosis of systems. There have been many studies about anomaly detection in systems based on analyzing logs. Cinque et al. proposed a novel rule-based logging approach for the analysis of software failures [25]. He et al. proposed Log3C [26], a novel clustering-based approach to identify impactful system problems. Recently, with the rise of deep learning, many log anomaly detection methods based on deep neural networks have been proposed. For example, Lu et al. used a supervised convolutional neural network that can automatically learn event relationships in system logs and detect anomalies [27]. Although accurate, this method is limited in its specific application scenarios and requires domain expertise to label abnormal logs in advance. More importantly, the disadvantage of this method is that newer fault messages are easily undetected. Borghesi et al. proposed an approach for anomaly detection in high performance computing systems based on autoencoders [28]; this approach was slightly less accurate than our method.

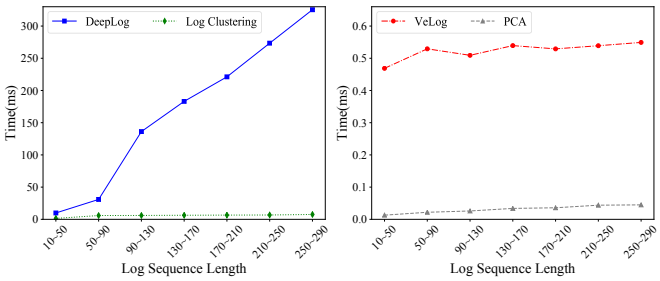


Fig. 6. The average time cost of detecting per log sequence.

Moreover, there have been many anomaly detection methods based on deep sequence models. Brown et al. proposed five attention mechanism implementations and used word vector to detect anomalous log events [29]. Zhang et al. adopted LSTM and term frequency–inverse document frequency (TF-IDF) weight to predict anomalous log messages [30]. However, these methods are more concerned with the granularity of log events rather than entire log sequences and ignore the contextual information in log sequences. In this work, we take nearly entire information on the log sequence into account.

Traditional VAEs do not consider the temporal dependency on data, thereby limiting the applicability of traditional VAEs to time series [31]. Some methods use a recurrent neural network, such as LSTM, as the encoder and decoder in the VAE [32]. However, in this work, we do not use LSTM as the encoder and the decoder to solve the problem that the VAE does not consider the temporal dependency on data. Instead, we solve this problem in the feature extraction phase. VeLog calculates the order of log execution matrices in the feature extraction phase. Then, these matrices are used to train the VAE in the order of log execution anomaly detection model. In this way, VeLog considers the patterns of temporal dependency on logs and can efficiently detect anomalies. Specifically, unlike other studies on anomaly detection using VAEs that set different thresholds as the discriminating standards [32], [33], we adopt normal intervals as the discriminating standard to identify anomalies. Thus, VeLog can detect anomalies with higher accuracy.

VI. Conclusion and Future Work

This paper proposes VeLog, a novel anomaly detection method based on VAEs. VeLog adopts VAEs to learn the patterns of normal log sequences and detect anomalies. We evaluate VeLog on two large-scale log datasets collected from widely used distributed systems. Our experimental results confirm the effectiveness of VeLog.

Our future work includes experimenting with VeLog and other deep generative models on a wider variety of datasets

collected from different types of systems. On the premise of ensuring the high accuracy, we want to minimize the required adjustments that must be made by anomaly detection models. Moreover, by trying more various deep learning techniques, we want to find an efficient and accurate way to detect specific types of anomalies and determine their possible causes.

REFERENCES

[1] Apache hadoop. [Online]. Available: http://hadoop.apache.org/.

[2] OpenStack. [Online]. Available: https://www.openstack.org/.

[3] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen, "Fingerprinting the datacenter: automated classification of performance crises," in *ACM Proceedings of the 5th European conference on Computer systems*, pp. 111–124, 2010.

[4] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer, "Failure diagnosis using decision trees," *IEEE International Conference on Autonomic Computing*, pp. 36–43, 2004.

[5] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, "Failure prediction in ibm bluegene/l event logs," in *IEEE International Symposium on Parallel, and Distributed Processing*, pp. 1-5, 2008.

[6] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A survey," in *arXiv preprint arXiv:1901.03407*, 2019.

[7] S. Goldstein and S. Uchida, "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data," *PloS one*, vol. 11, no. 4, pp. e0152173-, 2016.

[8] P. Gogoi, B. Borah, and D. K. Bhattacharyya, "Anomaly detection analysis of intrusion data using supervised & unsupervised approach," *Journal of Convergence Information Technology*, vol. 5, no. 1, pp. 95–110, 2010.

[9] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *ACM Symposium on Operating Systems Principles*, pp. 117–132, 2009.

[10] J. Lou, Q. Fu, S. Yang, Y Xu, and J. Li., "Mining Invariants from Console Logs for System Problem Detection," in *USENIX Annual Technical Conference*, pp. 23–25, 2010.

[11] Q. Lin, H. Zhang, J. G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *ACM Proceedings of the International Conference on Software Engineering*, pp. 102–111, 2016.

[12] L. Ruff, R. A. Vandermeulen, N. Görnitz, A. Binder, E. Müller, K. Müller, and M. Kloft, "Deep Semi-Supervised Anomaly Detection," *arXiv preprint arXiv:1906.02694*, 2019.

[13] G. Iuhasz and I. Dragan, "An Overview of Monitoring Tools for Big Data and Cloud Applications," in *2015 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pp. 363-366, 2015.

[14] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: system log analysis for anomaly detection," in *IEEE International Symposium on Software Reliability Engineering*, pp. 207–218, 2016.

[15] Q. Fu, J. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *IEEE International Conference on Data Mining*, pp. 149–158, 2009.

[16] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, J. Chen, X. He, R. Yao, J. Lou, M. Chintalapati, F. Shen, and D. Zhang, "Robust log-based anomaly detection on unstable log data," in *ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 807–817, 2019.

[17] X. Yu, P. Joshi, J. Xu, G. Jin, H. Zhang, and G. Jiang, "Cloudseer: Workflow monitoring of cloud infrastructures via interleaved logs," *ACM SIGPLAN Notices*, vol. 51, no. 4, pp. 489–502, 2016.

[18] D. S. Hirschberg, "Algorithms for the longest common subsequence problem," *Journal of the ACM*, vol. 24, no. 4, pp. 664–675, 1977.

[19] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. FernÃ¡ndez-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, and S. Whittle, "The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1792-1803, 2015.

[20] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *ACM Conference on Computer and Communications Security*, pp. 1285–1298, 2017.

[21] C. Doersch, "Tutorial on variational autoencoders," *arXiv preprint arXiv:1606.05908*, 2016.

[22] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[23] S. Nedelkoski, J. Bogatinovski, A. K. Mandapati, S. Becker, J. Cardoso, and O. Kao, "Multi-Source Distributed System Data for AI-powered Analytics,", *Zenodo* 2019, [Online] Available: https://doi.org/10.5281/zenodo.3549604

[24] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[25] M. Cinque, D. Cotroneo, and A. Pecchia, "Event logs for the analysis of software failures: A rule-based approach," in *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 806–821, 2012..

[26] S. He, Q. Lin, J. Lou, H. Zhang, M. R. Lyu and D. Zhang, "Identifying impactful service system problems via log analysis," in *ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 60–70, 2018.

[27] S. Lu, X. Wei, Y. Li and L. Wang, "Detecting Anomaly in Big Data System Logs Using Convolutional Neural Network," in *IEEE International Conference on Dependable, Autonomic and Secure Computing, International Conference on Pervasive Intelligence and Computing, International Conference on Big Data Intelligence and Computing and Cyber Science and Technology Congress*, pp. 151–158, 2018.

[28] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini, 'Anomaly detection using autoencoders in high performance computing systems," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 9428–9433, 2019.

[29] A. Brown, A. Tuor, B. Hutchinson, and N. Nichols, "Recurrent neural network attention mechanisms for interpretable system log anomaly detection," in *ACM Proceedings of the First Workshop on Machine Learning for Computing Systems*, pp. 1–8, 2018.

[30] K. Zhang, J. Xu, M. R. Min, G. Jiang, K. Pelechrinis, and H. Zhang, "Automated IT system failure prediction: A deep learning approach," in *IEEE International Conference on Big Data*, pp. 1291–1300, 2016.

[31] S. Suh, D. H. Chae, H. G. Kang, and S. Choi, "Echo-state conditional variational autoencoder for anomaly detection," in *International Joint Conference on Neural Networks*, pp. 1015-1022, 2016.

[32] C. Zhang and Y. Chen, "Time Series Anomaly Detection with Variational Autoencoders," *arXiv preprint arXiv:1907.01702*, 2019.

[33] J. An and S. Cho, "Variational autoencoder based anomaly detection using reconstruction probability," *Special Lecture on IE*, vol. 2, no. 1, 2015.