Information Systems Frontiers (2021) 23:285–298
https://doi.org/10.1007/s10796-020-10026-3

LogGAN: a Log-level Generative Adversarial Network for Anomaly Detection using Permutation Event Modeling

Bin Xia[1] · Yuxuan Bai[1] · Junjie Yin[1] · Yun Li[1] · Jian Xu[2]

Published online: 16 June 2020
© Springer Science+Business Media, LLC, part of Springer Nature 2020

Abstract

System logs that trace system states and record valuable events comprise a significant component of any computer system in our daily life. Each log contains sufficient information (i.e., normal and abnormal instances) that assist administrators in diagnosing and maintaining the operation of systems. If administrators cannot detect and eliminate diverse and complex anomalies (i.e., bugs and failures) efficiently, running workflows and transactions, even systems, would break down. Therefore, the technique of anomaly detection has become increasingly significant and attracted a lot of research attention. However, current approaches concentrate on the anomaly detection analyzing a high-level granularity of logs (i.e., session) instead of detecting log-level anomalies which weakens the efficiency of responding anomalies and the diagnosis of system failures. To overcome the limitation, we propose an LSTM-based generative adversarial network for anomaly detection based on system logs using permutation event modeling named LogGAN, which detects log-level anomalies based on patterns (i.e., combinations of latest logs). On the one hand, the permutation event modeling mitigates the strong sequential characteristics of LSTM for solving the out-of-order problem caused by the arrival delays of logs. On the other hand, the generative adversarial network-based model mitigates the impact of imbalance between normal and abnormal instances to improve the performance of detecting anomalies. To evaluate LogGAN, we conduct extensive experiments on two real-world datasets, and the experimental results show the effectiveness of our proposed approach on the task of log-level anomaly detection.

Keywords Anomaly detection · Generative adversarial network · Log-level anomaly · Permutation event modeling

1 Introduction

Anomaly detection is an essential task in protecting our daily life from those intended or unintended malicious attacks such as the network intrusion, mobile fraud, industrial damage, and abnormal condition of system (Chandola et al. 2009). However, with the rapid development of computer science, systems and applications become increasingly complex which makes anomalies diverse and challenging to be detected even by human beings. Except for the intended malicious attacks, unknown bugs and errors which are seemingly controllable but caused by non-artificial reason in online systems damage the secure and reliable operating environment. Therefore, the effectiveness and efficiency of anomaly detection have become a big challenge for the further development of information-based society.

Currently, the automated generation of logs is an indispensable component of large scale systems. System logs trace the status of the system and record each critical event in detail to assist administrators in diagnosing bugs, failures, and errors of systems. Therefore, the density of arrival logs and the description of logs directly determine the value of the quantity of knowledge for improving the performance of running systems (Tang et al. 2011; Li et al. 2017). For example, if arrival logs are extremely dense, it is a challenge to analyze the dependency between events due to the concurrency of logs. Likewise, if the description

✉ Bin Xia
bxia@njupt.edu.cn

Yun Li
liyun@njupt.edu.cn

Jian Xu
dolphin.xu@njust.edu.cn

[1] Jiangsu Key Laboratory of Big Data Security and Intelligent Processing, Nanjing University of Posts and Telecommunications, Nanjing, China

[2] Nanjing University of Science and Technology, Nanjing, China

of logs is colloquial and obscure to represent the state of a system, it is challenging to trace the workflows. Figure 1 illustrates the arrival frequency of system logs in practical scenarios, where Fig. 1a shows the logs generated by 203 nodes during 2 days in HDFS and Figure 1b illustrates the logs generated by 1 node during 215 days in BlusGene/L. Observed from Fig. 1, the peak frequency of arrival logs is 198,878/min and 152,929/hour for HDFS and BGL, respectively. In addition, the number of normal instances is much more than that of anomalies, and generally, anomalies are unlabeled. Therefore, such an extremely frequent arrive of unlabeled logs results in a significant challenge to the prompt response and the precise diagnosis.

To overcome the challenges mentioned above, researchers take a lot of efforts on the anomaly detection based on system logs. The proposed approaches are mainly categorized into the supervised, semi-supervised, and unsupervised strategy based on the availability of labeled data (i.e., normal and abnormal instances). Most of these approaches have good performance in detecting anomalies based on diverse system logs. However, there exist three problems in restricting the further development of system diagnosis (Bodik et al. 2010; Lin et al. 2016; Lou et al. 2010; Liu et al. 2008). First, based on a session containing many logs and divided base on some rules (e.g., period, transaction, and node), these approaches detect session-level anomalies. In other words, the session will be detected if there exists at least an anomaly, however, the abnormal logs cannot be located. Therefore, administrators need to

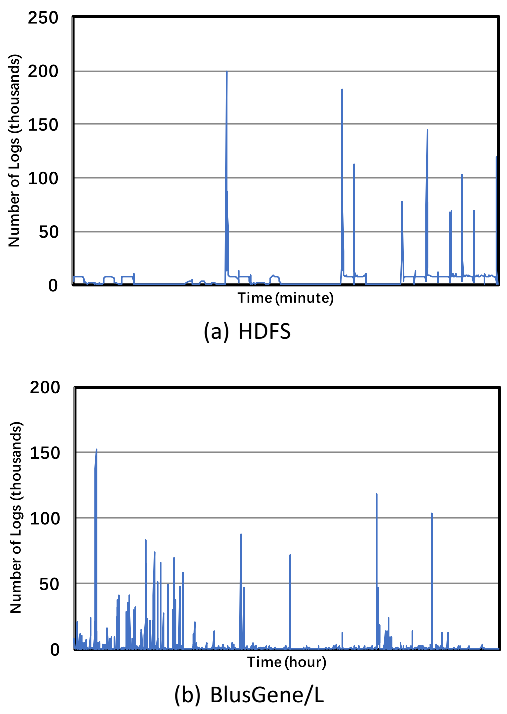


Fig. 1 Arrival frequency of system logs in the real-world datasets

diagnose the workflows in the session, which is a challenging task. Second, an anomaly is not alerted until logs are traversed in the session. In other words, the anomaly cannot be detected and responded efficiently when the abnormal log is arriving. Third, generally, system logs are temporal and dependent on previous logs. However, arrival logs are out-of-order due to unknown reasons for delays in the systems. Therefore, the techniques mining sequential information may capture false patterns from these out-of-order logs. Meanwhile, the techniques capturing information from a set of logs may ignore significant temporal features. These issues significantly limit the effectiveness and efficiency of system diagnosis.

In this paper, we cast the task of anomaly detection as a pattern-based sequential prediction and propose an LSTM-based generative adversarial network named LogGAN using permutation event modeling to distinguishing upcoming abnormal events based on temporal system logs. First, we exploit a customized log parser to extract the structured information (i.e., timestamps and signature) and transform each log into an event. Second, the combination of events (i.e., pattern) and the corresponding upcoming event are collected from temporal system logs using different sliding windows. The collected combinations of patterns and events are utilized to construct a real training dataset. For mitigating the strong sequential characteristics of LSTM, the permutation event modeling is applied to combine permutated out-of-order patterns with original upcoming events to extending the real training dataset. Third, LogGAN consists of two major components: (1) generator and (2) discriminator. The generator tries to capture the distribution of real training dataset and synthesizes plausible instances (i.e., fake training set comprised of normal and abnormal logs), while the discriminator aims to distinguish fake instances from the dataset, which is built using real and synthetic data. Finally, the fully-trained generator is applied to detect whether the upcoming log is normal or abnormal based on the latest events. According to the game setting of anomaly detection, LogGAN exploits the pattern-based training mode to mitigate the problem of the imbalance between normal and abnormal instances instead of using the instance-based training mode. In addition, the LSTM-based generator identifies whether each upcoming log is normal or abnormal, which efficiently responds alerts of anomalies and effectively assists administrators in diagnosing workflows instead of detecting abnormal sessions. To the best of our knowledge, this is the first attempt to apply a game setting (i.e., adversarial learning) for the anomaly detection based on system logs. Our contribution can be summarized as below:

– A generative adversarial network is proposed to mitigate the problem of imbalance between normal and abnormal instances while improving the performance of anomaly detection.

- A permutation event modeling is proposed to mitigate the strong temporal sequential dependency of LSTM.
- An LSTM-based detector promotes the efficiency of responding anomalies and marks anomalies of logs instead of detecting session-level anomalies.

2 Related Work

The technique of anomaly detection is widely applied in many practical scenarios such as the financial statement fraud (Huang et al. 2017), the post-disaster situation analysis (Mondal et al. 2018), and the social media event detection (Troudi et al. 2018). Generally, the anomaly detection (i.e., outlier detection) is categorized as supervised, semi-supervised, and unsupervised anomaly detections. In this section, we will briefly introduce some popular anomaly detections in each category of techniques.

2.1 Supervised Anomaly Detection

Supervised anomaly detections operate under two general assumptions: (1) the labels of normal and abnormal instances are available; (2) the normal and abnormal instances are distinguishable given the feature space. Chen et al. proposed a decision tree-based approach to detecting the actual failures from large Internet sites (i.e., eBay) based on the temporal request traces (Chen et al. 2004). The decision trees simultaneously handle the varying types of runtime properties (i.e., continuous and discrete variables). Therefore, the proposed approach was widely used in many practical scenarios. Bodik et al. proposed a fingerprint (i.e., vector) to effectively demonstrate the performance state of systems and implemented a regularized logistic regression-based method for selecting the relevant metrics to build the appropriate fingerprints (Bodik et al. 2010). The anomalies can be precisely identified using the fingerprints which summarize the properties of the whole data center (e.g., CPU utilization). Liang et al. employed several classifiers (e.g., SVM and nearest neighbor) to detecting the failures in the massive event logs which were collected from the supercomputer IBM BlueGene/L (Liang et al. 2007). Similar to Bodik et al., they also derived the specific combination of features to effectively describe each event log for improving the performance of classification tasks, which demonstrates that the representation of normal and abnormal logs is significant. The supervised methods have a quick test phase for the online detections, however, the extreme dependency on the quality of labels limits the application scenarios (Jian et al. 2015; Jian et al. 2016; Jian et al. 2016). In this paper, the proposed LogGAN is a supervised anomaly detection algorithm and an extended version of our previous work, where the major extension part is the permutation event modeling. The permutation event modeling includes a preprocessing strategy of temporal events to addressing the problem of out-of-order events and a training mode to further relieving the imbalance between normal and abnormal samples (Xia et al. 2019).

2.2 Semi-supervised Anomaly Detection

The semi-supervised anomaly detection operates under the assumption: given the feature space, the normal samples are located closely while the anomalies are far from the clusters of normal ones (Chandola et al. 2009). The representative of the semi-supervised model is the nearest neighbor-based techniques which can be categorized as (1) distance-based neighbors, and (2) density-based neighbors. To address the problem of the high-dimensional feature space, Zhang et al. proposed a High-Dimension Outlying subspace Detection (HighDOD) to searching for the optimal subset of features to represent outliers (Ji and Wang 2006). Due to the subset of features (i.e., low-dimensional data), the Euclidean distance is capable of describing the actual distance between normal and abnormal instances. Besides distance-based approaches, the density-based method is also useful to distinguish anomalies. To improve Local Outlier Factor (i.e., a type of popular measure to calculating the density given the instance), Chawla et al. proposed a new measure called Spatial Local Outlier Measure (SLOM) (Sun and Chawla 2004; Chawla and Sun 2006). Du et al. proposed LSTM-based anomaly detection and diagnosis framework named DeepLog based on unstructured system logs (Min et al. 2017). DeepLog analyzes and detects anomalies using the log key and the parameter value vector to help administrators for diagnosing the system errors based on workflows. DeepLog is trained based on the normal patterns in system logs and provides a way to be incrementally updated using upcoming logs; therefore, DeepLog is categorized as semi-supervised anomaly detection. Tuor et al. also proposed a recurrent neural network-based approach to detecting abnormal instances where the proposed model considered system logs as sentences in language models (Tuor et al. 2018). Compared to the supervised anomaly detections, semi-supervised techniques do not extremely rely on the labeled data and the distribution of observed instances and outperform the unsupervised approaches generally. However, the selection of measuring distance is significant for the performance of semi-supervised anomaly detections.

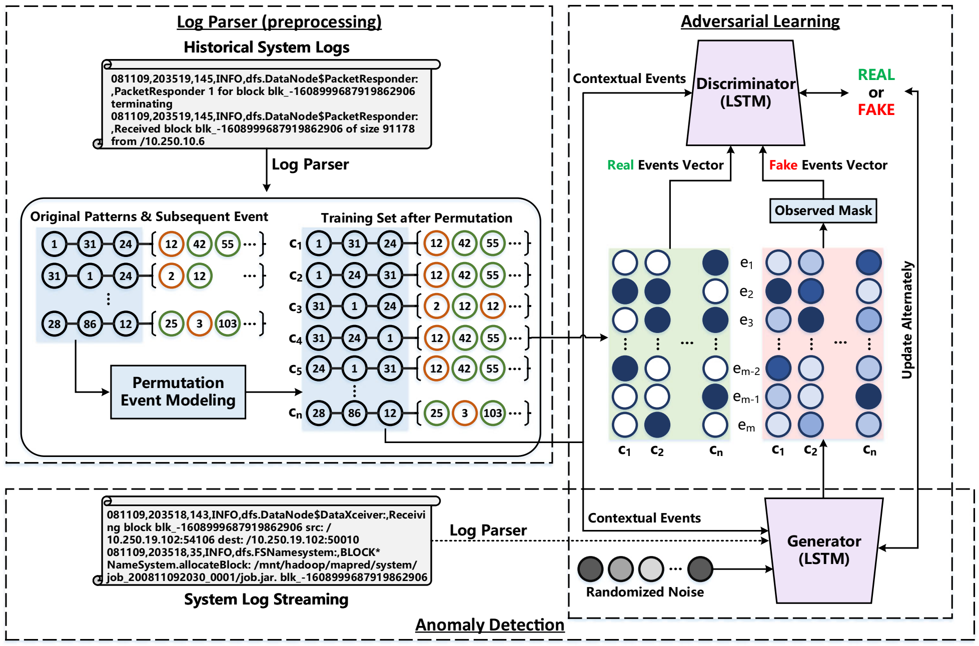
2.3 Unsupervised Anomaly Detection

The unsupervised technique is the most popular approach in the domain of anomaly detection because this technique still works even if the label of data is unknown. This characteristic of the unsupervised technique satisfies the assumption that anomalies are generally rare and unknown in practical scenarios. Lin et al. proposed a cluster-based approach (i.e., LogCluster) to addressing the log-based anomalies detection problem based on the data from Microsoft service product teams (Lin et al. 2016). LogCluster aims to cluster the historical and upcoming logs using the knowledge base, and engineers only need to distinguish several logs (i.e., events) in each cluster that can identify the type of anomalies which is located in the same cluster. Therefore, it is not necessary to obtain the label of logs, and the similarity between logs is more essential to operate LogCluster. Lou et al. proposed a novel anomaly detection approach to identifying program invariants based on the unstructured console logs (Lou et al. 2010). The proposed approach concentrates on structuring the free form description in console logs and mining the meaningful anomalies after grouping the structured logs with parameters. Different from the traditional anomaly detections which construct models fitting normal instances and distinguish instances that do not conform to the constructed model, Liu et al. proposed a novel concept that explicitly isolates abnormal instances (Liu et al. 2008). The proposed isolation forest (iForest) is capable of addressing the high-dimensional problems using an attribute selector (i.e., the characteristic of the decision tree). In addition, iForest achieves good performance even if there are no anomalies occurred in the training set. Xu et al. proposed a PCA-based anomaly detection and visualized the promising results using a decision tree (Wei et al. 2009). The main contribution of this work is that the source code is considered as a reference to parse console logs for improving the quality of structured data and the quality data will improve the representation of console logs (i.e., extracted distinguishable features). The advantage of unsupervised techniques is that the approaches are independent with the label information of the training set. The disadvantage of unsupervised techniques is that expert knowledge is still needed to utilize unsupervised approaches for detecting anomalies in practical scenarios, although the techniques reduce the massive workloads.

3 Method

In this paper, we propose a generative adversarial network-based anomaly detection approach named LogGAN which improves the performance of detecting anomalies. Figure 2 illustrates the overview of LogGAN. The main modules of LogGAN are categorized into three parts:

– *Log Parser*: is the module to parsing unstructured logs into structured logs (or events) which are considered as the minimum units for the following machine learning-based techniques.
– *Adversarial Learning*: is the module to training the LSTM-based anomaly detection model based on the timestamps, signatures, and attributes extracted from structured log.



```
081109,203519,145,INFO,dfs.DataNode$PacketResponder:
,PacketResponder 1 for block blk_-1608999687919862906
terminating
081109,203519,145,INFO,dfs.DataNode$PacketResponder:
,Received block blk_-1608999687919862906 of size 91178
from /10.250.10.6
```

```
081109,203518,143,INFO,dfs.DataNode$DataXceiver:,Receivi
ng block blk_-1608999687919862906 src: /
10.250.19.102:54106 dest: /10.250.19.102:50010
081109,203518,35,INFO,dfs.FSNamesystem:,BLOCK*
NameSystem.allocateBlock: /mnt/hadoop/mapred/system/
job_200811092030_0001/job.jar. blk_-1608999687919862906
```

Fig. 2 The framework of anomaly detection generative adversarial network

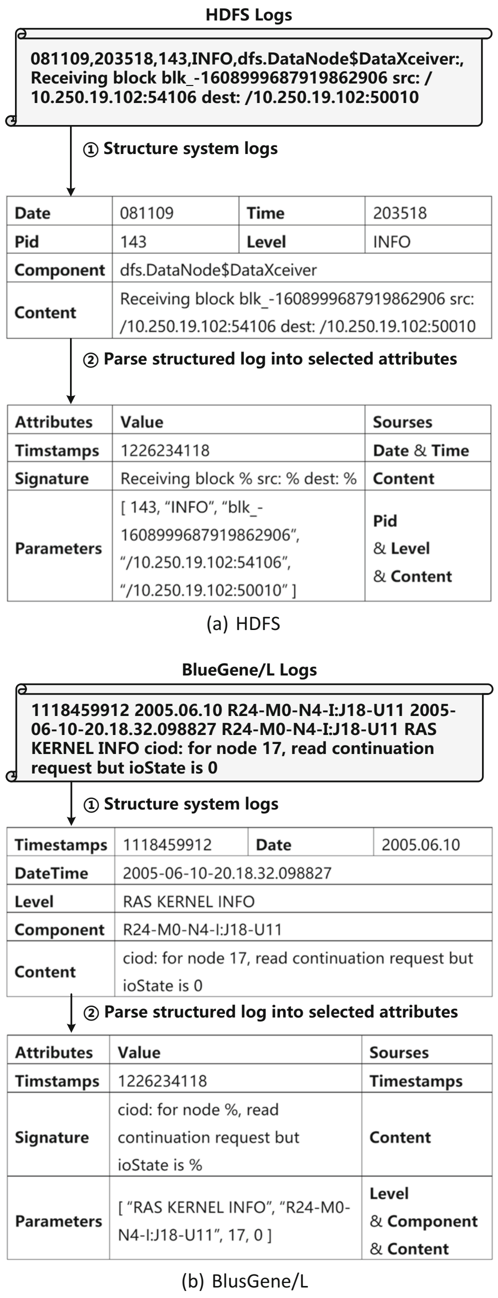
– *Anomaly Detection*: is the module to detecting and diagnosing anomalies using the LSTM-based model and incrementally update the model based on the upcoming logs and users' feedbacks.

In the following parts of this section, we will introduce each part of LogGAN in detail.

3.1 Log Parser

In the module of log parser, the original logs are converted into the structured logs. The log parsing, which is considered as the common preprocessing of unstructured logs, is a significant part of the majority of log analysis tasks. Many approaches are proposed to generate events, which are extracted and summarized based on original logs, for automated analysis of system (Tang et al. 2011; Guo et al. 2018). These template-free methods parse logs using statistical approaches. However, the performance of these methods is not convincing because the formations of logs from different systems are chaotic and challenging to be captured. Therefore, in this paper, we first divide the unstructured logs into several parts (e.g., datetime and content) using the corresponding template, then further extract meaningful information (i.e., event) from these parts (Zhu et al. 2018). Generally, the event consists of three major components: (1) timestamps, (2) signature, and (3) parameters. To make readers fully understand the process of log parser, Fig. 3 illustrates the examples of parsing unstructured logs from two real-world systems (i.e., HDFS and BlusGene/L), respectively.

Note that, HDFS and BlusGene/L are different in the system structures and workflows, hence the parsed structures from the first step are also different. Observed from Fig. 3a, the timestamps, signature, and parameters are extracted from the original log where the signature is a static content that presents a type of logs and the parameters record dynamic parts. The three-tuple representation (i.e., timestamps, signature, and parameters) effectively describes the status of each event, which provides administrators with sufficient references to diagnose the broken-down system. In this paper, we assume that the pattern of events is sufficient to determine whether the upcoming event is abnormal or normal, therefore, only timestamps and signature are utilized to describe temporal logs.


HDFS Logs

```
081109,203518,143,INFO,dfs.DataNode$DataXceiver:,
Receiving block blk_-1608999687919862906 src: /
10.250.19.102:54106 dest: /10.250.19.102:50010
```

① Structure system logs

Date	081109	**Time**	203518
Pid	143	**Level**	INFO
Component	dfs.DataNode$DataXceiver		
Content	Receiving block blk_-1608999687919862906 src: /10.250.19.102:54106 dest: /10.250.19.102:50010		

② Parse structured log into selected attributes

Attributes	**Value**	**Sourses**
Timstamps	1226234118	**Date** & **Time**
Signature	Receiving block % src: % dest: %	**Content**
Parameters	[143, "INFO", "blk_-1608999687919862906", "/10.250.19.102:54106", "/10.250.19.102:50010"]	**Pid** & **Level** & **Content**

(a) HDFS

BlueGene/L Logs

```
1118459912 2005.06.10 R24-M0-N4-I:J18-U11 2005-
06-10-20.18.32.098827 R24-M0-N4-I:J18-U11 RAS
KERNEL INFO ciod: for node 17, read continuation
request but ioState is 0
```

① Structure system logs

Timestamps	1118459912	**Date**	2005.06.10
DateTime	2005-06-10-20.18.32.098827		
Level	RAS KERNEL INFO		
Component	R24-M0-N4-I:J18-U11		
Content	ciod: for node 17, read continuation request but ioState is 0		

② Parse structured log into selected attributes

Attributes	**Value**	**Sourses**
Timstamps	1226234118	**Timestamps**
Signature	ciod: for node %, read continuation request but ioState is %	**Content**
Parameters	["RAS KERNEL INFO", "R24-M0-N4-I:J18-U11", 17, 0]	**Level** & **Component** & **Content**

(b) BlusGene/L

Fig. 3 Example of log parser to converting from logs to structured entities

3.2 Adversarial Learning

In this paper, we cast the anomaly detection as a task of adversarial learning and propose an LSTM-base generative adversarial network named LogGAN to improve the performance of detecting anomalies. The concept of the generative adversarial network (GAN) was proposed by Goodfellow et al. where GAN considers a machine learning problem as a game between two models (i.e., generator and discriminator) (Goodfellow et al. 2014). The generator (G) captures the distribution of real samples and generates plausible samples which are similar with real samples in the representation of features, while the discriminator (D) tries to identify whether the upcoming sample is real or synthetic one for improving the quality of samples generated by G. The iteration repeats until both G and D converge, then G is capable of generating 'real' samples. The fully-trained G can capture the distribution of anomalies which further improves the performance of detecting whether the upcoming log is normal or abnormal.

The original GAN, which is utilized to generate continuous variables of images, do not match the scenario of predicting discrete event ID (i.e., signature) (Goodfellow et al. 2014). Therefore, we propose LogGAN to independently generate the continuous probability of each upcoming event instead of using the softmax layer to output the probability distribution of overall events (Wang et al. 2017; Chae et al. 2018). In details, given an observed set of temporal events $\mathbf{S} = \{e_{(1)}, e_{(2)}, ..., e_{(s)}\}$ from parsed system logs and a set of event $\mathbf{E} = \{e_1, e_2, ..., e_m\}$ where e_j presents a signature of the j_{th} event, the task of LogGAN is to predict whether the upcoming event (i.e., log) is normal or abnormal based on the context combinations from the set $\mathbf{C} = \{c_1, c_2, ..., c_n\}$ where c_i demonstrates the i_{th} combination $(e_{(k-2)}, e_{(k-1)}, e_{(k)})$ within a 3-size sliding window. As a game setting, we exploit Long Short Term Memory network (LSTM) for both G and D where G aims to generate fake normal and abnormal instances and D tries to distinguish whether the instance is real or fake. For G, we utilize a random noise $\mathbf{z}$ and a combination c_i as the input of LSTM[1] while the output is an $m-$dimensional vector representing the independent occurring probability of each event in $\mathbf{E}$. For D, we utilize a combination c_i as the input and an $m-$dimensional vector of the independent occurring probability as the parameter[2] of LSTM while the output is whether the $m-$dimensional vector is real or fake sample under the contextual combination c_i. Therefore, the objective function of G and D is defined as follows, respectively:

$$\begin{aligned} J^G &= \min_\theta \sum_{i=1}^{n} (\mathbb{E}_{\hat{\mathbf{e}}} \sim P_\theta[\log(1-D(\hat{\mathbf{e}}|\mathbf{c}))] + \sum_{j=1}^{m} (\hat{e_j} - e_j)^2) \\ &= \min_\theta \sum_{i=1}^{n} (\log(1-D(\hat{\mathbf{e}_{\mathbf{c_i}}}|\mathbf{c_i})) + \frac{1}{m} \sum_{j=1}^{m} (\hat{e_{c_i j}} - e_{c_i j})^2)), \quad (1) \end{aligned}$$

$$\begin{aligned} J^D &= \min_\phi - \sum_{i=1}^{n} (\mathbb{E}_{\mathbf{e} \sim P_{true}}[\log D(\mathbf{e}|\mathbf{c})] \\ &\quad + \mathbb{E}_{\hat{\mathbf{e}} \sim P_\theta}[\log(1 - D(\hat{\mathbf{e}}|\mathbf{c}))]) \\ &= \min_\phi - \sum_{i=1}^{n} (\log D(\mathbf{e_{c_i}}|\mathbf{c_i}) + \log(1 - D(\hat{\mathbf{e}_{\mathbf{c_i}}}|\mathbf{c_i}))), \quad (2) \end{aligned}$$

[1]Learned event embedding is used to demonstrate each event.

[2]In D, we cast the combination c_i as the input of LSTM and LSTM directly outputs the hidden layer without any manipulation. Then, we concatenate the $m-$dimensional vector with the hidden layer as an input of a two-layer full Connected neural network which outputs whether the $m-$dimensional vector is real or fake as a binary classification.

where θ and ϕ is the parameter of G and D, respectively. Note that, $\hat{\mathbf{e}_{\mathbf{c_i}}} = \mathbf{e}'_{\mathbf{c_i}} \odot \mathbf{o_{c_i}}$ is an $m-$dimensional vector representing the independent occurring probability of each event in $\mathbf{E}$ (i.e., input of D), where $\mathbf{e}'_{\mathbf{c_i}}$ is the output of G and $\odot$ is the element-wise mask multiplication. $\mathbf{o_{c_i}}$, which is an $m-$dimensional observed vector (i.e., $o_{c_i j}$ stands for the observation of e_j where $o_j \in \{1, 0\}$ represents whether e_j is an upcoming event next to $\mathbf{c_i}$ or not), is used to filter the occurring probability of unobserved events in $\mathbf{e}'_{\mathbf{c_i}}$. This setting assists LogGAN to only update the gradients based on the loss of observed events (i.e., both normal and abnormal instances) and avoid the disturbance generated by the unobserved one. In addition, during the process of updating G, we apply a reconstruction error (i.e., $\sum_{j=1}^{m} (\hat{e_{c_i j}} - e_{c_i j})^2$) to help G capture the actual distribution of training data for further improving the performance. Algorithm 1 shows the overall algorithm of LogGAN in detail.

Algorithm 1 The algorithm of LogGAN

```
Input:
    $G_\theta$: the generator $G$,
    $D_\phi$: the discriminator $D$,
    $B$ : the size of minibatch,
    $N$: the number of maximum iteration.
Output:
    $G_{\theta^*}$: converged generator $G$.
 1: Initialize $G_\theta$ and $D_\phi$ with random weights $\theta$ and $\phi$.
 2: Set $t \leftarrow 0$
 3: repeat
 4:     for G-steps do
 5:         Sample $B$ combinations of events as a minibatch $M_G$
 6:         Generate corresponding fake instances using generator $G_\theta$ and train $G_\theta$
 7:         Update $G_\theta$ by $\theta^* \leftarrow \theta - \frac{1}{B}\nabla_\theta J^G$
 8:     end for
 9:     for D-steps do
10:         Sample $B$ combinations of events as a minibatch $M_D$
11:         Generate corresponding fake instances using generator $G_\theta$
12:         Combine the generated instances with sampled real instances and train $D_\phi$
13:         Update $D_\phi$ by $\phi^* \leftarrow \phi - \frac{1}{B}\nabla_\phi J^D$
14:     end for
15:     Update $t \leftarrow t + 1$
16: until LogGAN converges OR $t >= N$
17: return $G_{\theta^*}$.
```

3.2.1 Permutation Event Modeling (PEM)

The imbalance between abnormal and normal logs and the strong sequential dependency of LSTM are big challenges in practical scenarios of anomaly detection. In this paper, we propose a permutation event modeling strategy that consists of two modules to solve these problems.

First, to mitigate the impact of the imbalance between abnormal and normal logs, we convert the conventional instance-based training mode into a pattern-based training mode. In the previous research, the proposed method uses sliding windows to split temporal logs and construct independent training samples (Min et al. 2017). There exist two issues under this training setting. On the one hand, anomalies are extremely sparse, therefore, it is hard for models to capture the patterns of anomalies sufficiently. On the other hand, the instance-based training mode will confuse the model because there may be several possible normal and abnormal events followed by a specific pattern. The pattern-based training mode is to collect possible subsequent abnormal and normal events followed by a specific pattern in the training set and considers the pair of pattern and possible subsequent events as a training sample. In other words, the label of a pattern is an $m-$dimensional vector where the observed normal events are set as 1, and the observed abnormal and unobserved events are set as 0. The observed mask $\mathbf{o_{c_i}}$ is used to keep the gradient of observed events and filter the gradient of unobserved ones. Based on the pattern-based training mode, the number of training instances, especially for normal logs, has dropped dramatically. Therefore, this strategy effectively mitigates the imbalance between abnormal and normal logs while increasing the efficiency of the training stage.

Second, to mitigate the strong sequential dependency of LSTM and the problem of distinguishing unobserved anomalies, the permutation event modeling is used to permutate original patterns for extending the training set (Yang et al. 2019).

On the one hand, due to unknown delays (e.g., network delay and disk I/O) and high concurrency of logs, the actual arrival order of logs cannot be guaranteed in practical scenarios. In fact, the temporal sequence of system logs contains sufficient information which is significant for detecting anomalies (Wang et al. 2018; Zeng et al. 2014). However, LSTM is extremely sensitive to the temporal order of logs within a pattern, in other words, LSTM tends to consider the new permutation of an observed sample as an unobserved sample. To overcome this limitation of LSTM, we propose a permutation event modeling strategy. Figure 4 shows a case[3] to make readers easy to understand the mechanism: (1) the subsequent events of the most frequent pattern (i.e., the second pattern which appears 1+40 times in the training set) are considered as the baseline subsequent events of permutation patterns; (2) for remaining observed patterns (i.e., the first and third patterns), the subsequent events of their permutation patterns consist of the subsequent events of these observed patterns and the difference between subsequent events of the most frequent pattern and the observed pattern. For example, the difference of (203,186,213) and (203,213,186) is 3 and the difference of (203,186,213) and (186,203,213) is 349. In this paper, we assume that the permutation of a given log sequence will still result in a sequence that is not an anomaly. Due to the imbalance between normal and abnormal samples in most current large-scale systems, this assumption is convincing in most cases.

[3]The case is selected from the training set and we adapt the case a bit to explain all types of processing using the permutation event modeling.

On the other hand, anomalies are extremely sparse, and detecting unobserved anomalies is more significant than observed ones. However, the extended permutation patterns contain abundant knowledge to distinguishing these unobserved anomalies. Recently, Niven et al. show that the reasoning ability of neural networks is limited and models are easy to be attacked by adversarial samples. The experimental results show that the performance of the model highly depends on the quality of training sets (Niven and Kao 2019). Therefore, based on the permutation event modeling, more meaningful samples are generated to provide models with meaningful knowledge which is capable of improving the performance of detecting unobserved anomalies and distinguishing unobserved normal events.

3.2.2 Negative Sampling

In practical scenarios, given a combination of events, the possible upcoming events are sparse. In other words, the real event vector (i.e., $\mathbf{e_{c_i}}$) is more like a one-hot or multi-hot encoding vector which causes the overfitting problem. Therefore, we exploit a negative sampling strategy to avoid the overfitting problem (Chae et al. 2018). During the G-steps, we randomly sample the unobserved instances according to a specific ratio and set the corresponding position of mask o_{c_i} as 1 for retaining the gradients.

3.3 Anomaly Detection

After completing the train of LogGAN, generator G is applied to detect anomalies based on the streaming events from system logs. During the stage of anomaly detection: (1) the historical and upcoming system logs are transformed into structured data (i.e., event) via the log parser; (2) the input of G is the combination of several latest events (i.e., several one-hot encoding vectors) and G generates a corresponding $m-$dimensional vector representing the

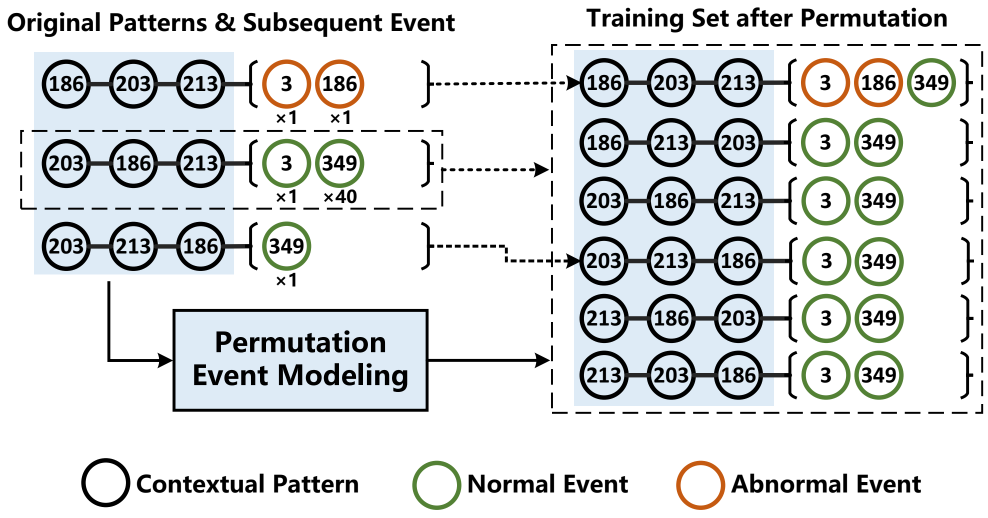


Fig. 4 The mechanism of permutation event modeling

independent occurring probability of each event; (3) a set of normal events is built based on the generated $m-$dimensional vector filtered using a predefined threshold of normal probability in the step 2; (4) the upcoming event is considered as a normal instance if the event has an intersection with the set of normal events, otherwise, the event will be alerted as an anomaly.

4 Experiment

In this section, we propose the experiments to evaluate the effectiveness of LogGAN on two real-world datasets, and mainly concentrate on the following issues:

– *Parameter*: We analyze the effect of different parameters on the performance of LogGAN.
– *Session-level Anomaly Detection*: The performance of LogGAN on the task of session-level anomaly detection is compared to that of baselines.
– *Log-level Anomaly Detection*: The performance of LogGAN on the task of log-level anomaly detection is compared to the performance of DeepLog.

4.1 Experimental Setup

Datasets: Generally, up-to-date system logs are rarely published and are sensitive data that describe the detailed information (i.e., business and transaction) about deployed large scale systems, however, the data collected from own small scale system hardly show the actual anomalies in practical scenarios. Therefore, we exploit two real-world datasets (i.e., HDFD and BGL[4]) collected several years ago which is published for research (Zhu et al. 2018). HDFS is collected from Amazon EC2 platform where 11,197,705 system logs are divided into 575,139 sessions and generated by 203 nodes during two days while BGL contains 4,747,963 logs collected from the BlueGene/L supercomputer system during 215 days. The detailed information of datasets is shown in Table 1. In addition, because we are concerned with the reasoning ability of proposed LogGAN on the log-level anomaly detection under different sizes of sliding window, the ratio of observed and unobserved types of samples in the temporal training (30%) and testing (70%) set of BGL is shown in Table 2.

Baselines: In the experiments, to evaluate the performance of our proposed approach, we compare LogGAN with several selected baselines:

– iForest (Liu et al. 2008): is an *unsupervised* tree-based isolation forest which tries to isolate anomalies from other normal instances, especially for the imbalanced training set.
– PCA (Wei et al. 2009): is an *unsupervised* principal component analysis-based anomaly detection technique which improves the parser of unstructured systems and visualizes the promising diagnosis of abnormal instances.
– Invariants Mining (Lou et al. 2010): is an *unsupervised* anomaly detection technique applied to build the structured logs based on the unstructured description in console logs.
– LogCluster (Lin et al. 2016): is an *unsupervised* cluster-based approach to clustering events extracted from the historical and upcoming logs based on the knowledge base.
– DeepLog (Min et al. 2017): is a *supervised* LSTM-based deep learning framework which utilizes LSTM to

[4]https://github.com/logpai/loghub

Table 1 The overview of two real-world datasets

System	Start Date	Days	Size(GB)	Rate(log/sec)	Messages	Alerts	Signatures
HDFS	2008-11-09	2	1.490G	64.802	11,197,705	16,916/575,139	29
BGL	2005-06-03	215	0.708G	0.256	4,747,963	348,698	394

fit the distribution of normal instances using the log key and the performance value vector extracted from each log. DeepLog, which provides an end-to-end anomaly detection solution, is a representative log-level baseline for the deep learning algorithms. In addition, the original version of DeepLog (i.e., the semi-supervised model) exploits misclassified samples to online update the model for improving the performance of detecting upcoming anomalies where the misclassified samples are labeled by administrators manually. To compare the performance of models, in the experiments, we train DeepLog on the same training set used to train LogGAN and prevent the intervention of administrators.

In the experiments, we exploit the first 30% of dataset as the training set while the remaining data as the test set based on time series. In addition, we will briefly introduce the key parameters of LogGAN for the reproduction of our model. The size of sliding window determines the capacity of contextual events to the upcoming log. The larger size demonstrates the more specific contextual patterns are used to identify anomalies while the smaller size means upcoming anomalies are determined by the latest events (i.e., the more regular contextual patterns). The event embedding is used to represent events in the continuous space. In this paper, we utilize the 2-size, 3-size, and 4-size sliding window to extracting contextual pattern of upcoming logs. To distinguish normal and abnormal events from the output of generator (i.e., an $m-$dimensional vector), we define a threshold to filtering normal logs. When the occurring probability of a event is below the predefined threshold, the event is considered as an anomaly based on the contextual pattern. In addition, we define the threshold as 0.90 which means the upcoming log is normal if the appearing probability of the log is 90% based on the output of generator. The ratio of negative sampling is set as 0.1. The 2-layer LSTM is applied as the basic model of generator and discriminator in LogGAN. The dimension of event embedding is set as 200. To keep the correspondence with DeepLog, in the experiments, we define the accurate identification of true anomalies as the true positive.

4.2 Result and Discussion

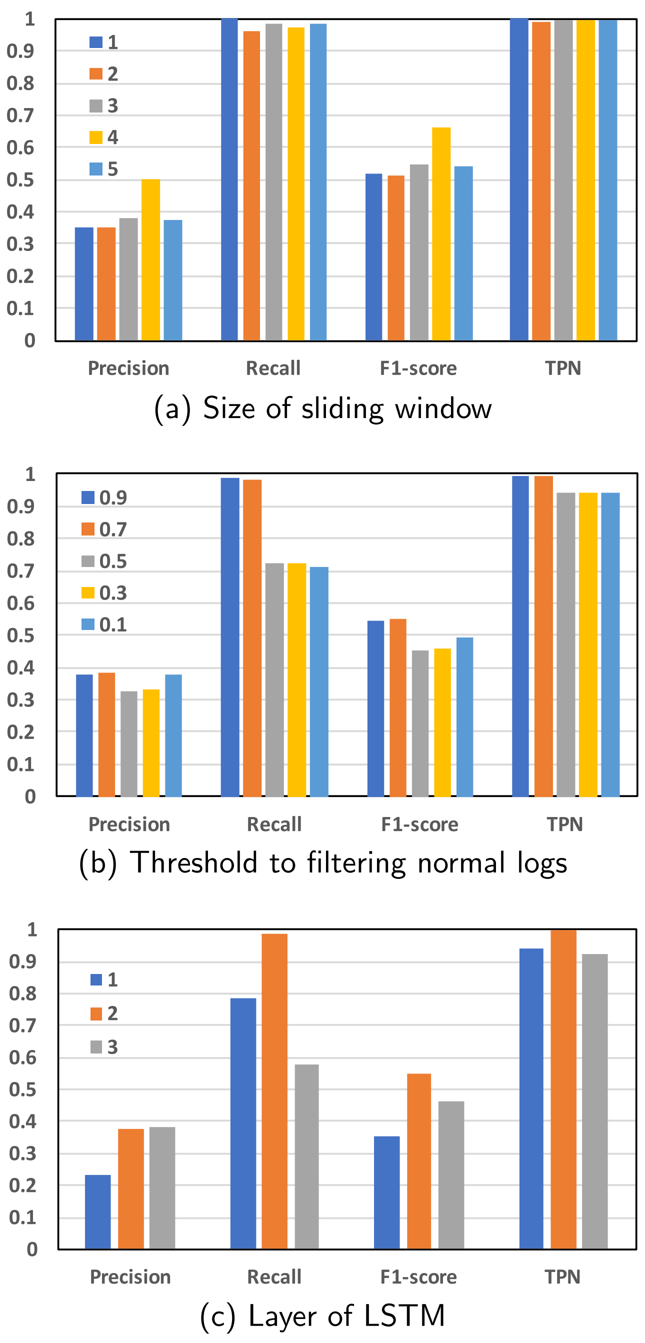
4.2.1 Parameters

In this section, to focus on the performance of independent LSTM-based GAN within different settings of parameters, we concentrate on the task of log-level anomaly detection without the permutation event modeling. Figure 5 illustrates the performance of LogGAN, including the size of the sliding window, the threshold to filtering normal logs, and the layer of LSTM.

First, Fig. 5a shows the performance of LogGAN on different sizes of the sliding window (i.e., size 1 to 5). Observed from Fig. 5a, abnormal logs are correlated with the appropriate context of events (i.e., 4-size sliding window). Compared the performance of 1-size and 4-size windows, the performance of recall and precision indicates that the 1-size window tends to consider normal events as anomalies while the 4-size window has a better overall performance without the permutation event modeling. As shown in Table 2, normal events are overwhelmingly more than abnormal events. LogGAN using the setting of a 4-size window has the failure of detecting several anomalies (i.e., Recall), however, LogGAN has great improvement

Table 2 The observed and unobserved types of samples in training and testing set of BGL

Window Size	Dataset	Observation	Normal Events	Abnormal Events	Total
2	Training	Observed	2,237	176	2,143
	Testing	Observed	1,084	37	1,121
		Unobserved	3,114	369	3,483
3	Training	Observed	4,190	293	4,483
	Testing	Observed	1,890	40	1,930
		Unobserved	6,244	617	6,861
4	Training	Observed	6,840	457	7,297
	Testing	Observed	2,942	45	2,987
		Unobserved	11,055	917	11,972



(a) Size of sliding window

(b) Threshold to filtering normal logs

(c) Layer of LSTM

Fig. 5 The performance of LogGAN within different settings of parameters

on distinguishing a large number of normal events (i.e., Precision).

Second, Fig. 5b illustrates the performance on different settings of threshold to filtering normal logs. Note that LogGAN on each threshold has good performance, in other words, the proposed model is capable of distinguishing between normal and abnormal events. In addition, there is a significant decline (e.g., Recall) on the overall performance if the threshold is decreased from 0.7 to 0.5. This result means the probability of some normal and abnormal events is around 0.6, in other words, the proposed model still cannot distinguish these obscure events. Finally, Figure 5c shows the performance of LogGAN using different layers of LSTM in the generator and discriminator. The experimental results demonstrate that appropriately using deep features (i.e., 2-layer LSTM) is capable of improving the performance of detecting anomalies.

4.2.2 Session-level Anomaly Detection

Table 3 shows the performance of baselines and LogGAN on HDFS dataset. Different from the version of LogGAN used in the log-level anomaly detection, we propose a session-level version of LogGAN (LogGAN-sess) in the session-level task. The generator of LogGAN-sess aims to match a 30−dimensional vector where the first 29 dimensions record the number of corresponding events appeared in the current session, and the last one represents the abnormal score instead of fitting an m−dimensional vector. The experimental results show that LogGAN-sess outperform other baselines except for LogClustering. The limitation of current LogGAN-sess is the model only exploits the statistics of independent event that occurred in the session. However, the traditional anomaly detection methods concentrate on the concurrence of several events in the temporal sequence. In addition, the structure of workflows is also significant information which describes the normal and integrated transactions in the system. Therefore, the performance of LogGAN-sess could be further improved using the statistics of specific patterns (i.e., the combination of temporal logs).

4.2.3 Log-level Anomaly Detection

Table 4 shows the comparison between DeepLog and LogGAN on the log-level anomaly detection. The overall performance of LogGAN outperforms the performance of DeepLog. In this section, we will discuss the experimental results from three aspects: (1) the effectiveness of permutation event modeling; (2) the impact of the size of

Table 3 The comparison between baselines and LogGAN on session-level anomaly detection on HDFS

Method	Recall	Precision	F1-score
Invariants Miner	**1.000**	0.084	0.154
PCA	0.346	0.707	0.465
DeepLog	0.016	0.939	0.032
iForest	0.318	**1.000**	0.482
LogClustering	0.362	**1.000**	**0.532**
LogGAN-sess	0.356	**1.000**	**0.525**

Table 4 The comparison between DeepLog and LogGAN on log-level anomaly detection

Metric	Window	Method											
		DeepLog			DeepLog+PEM			LogGAN-PEM			LogGAN		
		Obs[a]	UnObs	All	Obs	UnObs	All	Obs	UnObs	All	Obs	UnObs	All
Precision	2	0.089	0.255	0.163	0.135	0.260	0.204	0.923	0.295	0.385	0.986	0.414	0.505
	3	0.180	0.266	0.233	0.196	0.267	0.241	0.983	0.300	0.379	**0.989**	0.300	0.383
	4	0.109	0.258	0.183	0.068	0.263	0.196	0.668	0.449	0.499	0.948	**0.460**	**0.546**
Recall	2	0.999	1.000	1.000	0.997	0.999	0.997	0.986	0.950	0.961	0.982	**0.986**	**0.983**
	3	0.975	0.995	0.987	0.998	1.000	0.999	0.987	0.986	0.986	0.987	0.957	0.966
	4	1.000	1.000	1.000	0.317	0.994	0.791	0.992	0.969	0.976	**0.992**	0.969	0.976
F1-score	2	0.162	0.406	0.280	0.233	0.413	0.339	0.953	0.450	0.550	0.984	0.583	0.667
	3	0.304	0.491	0.377	0.382	0.421	0.388	0.985	0.460	0.547	**0.988**	0.457	0.548
	4	0.196	0.411	0.309	0.112	0.416	0.314	0.785	0.614	0.660	0.969	**0.624**	**0.700**
TPN	2	1.000	0.994	1.000	1.000	0.992	0.998	0.999	0.923	0.991	**0.999**	**0.991**	**0.997**
	3	0.997	0.976	0.995	1.000	0.995	1.000	0.999	0.979	0.997	0.989	0.944	0.992
	4	1.000	1.000	1.000	0.926	0.968	0.928	0.999	0.984	0.995	**0.999**	0.985	0.995

[a]Obs is abbreviated for Observed

sliding window; (3) the reasoning ability of models; (4) the performance on combinations of techniques.

Effectiveness of PEM To evaluate the effectiveness of permutation event modeling, we propose DeepLog+PEM (i.e., DeepLog with PEM) and LogGAN-PEM (i.e., LogGAN without PEM). On the one hand, compared DeepLog with DeepLog+PEM, the performance of DeepLog+PEM outperforms that of DeepLog on the 2-size and 3-size sliding window. however, the performance becomes worse on the 4-size sliding window. DeepLog is an LSTM-based anomaly detection method that is trained using every pattern in the training set. In other words, there exist a large number of duplicate samples (especially for normal events) where the permutation samples generated by PEM are assumed to appear once. Therefore, PEM is capable of improving the performance of DeepLog, however, the overall results are not satisfactory. On the other hand, compared LogGAN-PEM with LogGAN, the overall performance of LogGAN has a significant improvement, especially on Precision. Note that LogGAN effectively improves the performance of distinguishing normal events (i.e., Precision) while maintaining the performance of detecting anomalies (i.e., Recall). In other words, LogGAN effectively reduces the false alarm rate (i.e., an actual normal event is considered as an anomaly). Therefore, PEM is capable of mitigating the strong sequential dependency of LSTM and improving the performance of LSTM-based models.

Impact of the size of sliding window To understand the impact of the size of the sliding window (i.e., length of pattern), we evaluate the performance of DeepLog and LogGAN within different settings on 2-size, 3-size, and 4-size sliding windows. Note that, the 3-size-sliding-window model produces the best performance of DeepLog while the 2-size and 4-size-sliding-window model produce the best performance of LogGAN. The experimental results show that the best choice of pattern length is based on the mechanism of models. DeepLog and LogGAN exploit different strategies for determining whether the upcoming event is normal or abnormal. Therefore, it is important to select the appropriate length of the pattern in practical scenarios.

Reasoning ability of models The reasoning ability of models is the most significant problem we are concerned about because the majority of anomalies are unobserved. The performance on unobserved samples directly determines the effectiveness and practicability of proposed models. To evaluate the reasoning ability of proposed models, we divide the testing set into the observed and the unobserved part. Table 4 shows the performance of DeepLog and LogGAN on the observed and unobserved sets. On the one hand, the performance of DeepLog on the unobserved set is better than that on the observed set. This phenomenon cannot indicate that the reasoning ability of DeepLog is great. The poor performance of DeepLog on the observed set presents that DeepLog is not well trained using the training data. On the other hand, LogGAN has better performance on the unobserved set than DeepLog while the performance of LogGAN on the observed set is significant. The experimental results show the great reasoning ability of LogGAN.

Performance on combinations of techniques In this paper, we exploit several components to improve the performance of LogGAN: (1) Observed Mask (OM); (2) Permutation Event Modeling (PEM); (3) Negative Sampling (NS). Table 5 shows the performance on different combinations of techniques using the 4-size-sliding-window LogGAN where '-' and 'O' represents the technique is used or not used, respectively. NS cannot be applied without OM, thus, the independent performance of NS is not listed. There are some interesting results. First, both OM and PEM can improve the performance by only 2-3%, however, the combination of OM and PEM has a great improvement of about 11%. Second, the combination of OM and NS is effective in reducing the false alarm rate (i.e., improve 27% on Precision), however, the performance of detecting anomalies is also decreased. Third, combining PEM with OM and NS can further reduce the false alarm rate while maintaining the performance of detecting anomalies. Therefore, if the practical scenario has a low tolerance for considering an anomaly as a normal event, the combination of OM and PEM is a good choice. Otherwise, the combination of OM, PEM, and NS is a better choice.

However, there still exist some limitations of LogGAN in practical scenarios. First, observed from Table 5, Recall decreased from 1 to 0.976, while Precision increased from 0.223 to 0.546 due to the supplement of techniques. In practical use, if Recall cannot reach 1 (i.e., a few anomalies would be classified as normal events), misclassified anomalies may bring crucial failures to the

Table 5 The performance of different combinations of techniques in BGL

Techniques			Precision	Recall	F1-score	TPN
OM	PEM	NS				
–	–	–	0.223	1.000	0.365	1.000
O	–	–	0.249	1.000	0.399	1.000
–	O	–	0.258	1.000	0.410	1.000
O	O	–	0.330	**0.995**	0.495	**0.999**
O	–	O	0.499	0.976	0.660	0.995
O	O	O	**0.546**	0.976	**0.700**	0.995

software system, which cannot be tolerant for stability. Therefore, the technique of negative sampling is not recommended in practical scenarios. Yan et al. proposed a unified cost-sensitive framework for automated malware (i.e., anomaly) classification which maximizes the detection rate while maintaining the false positive rate under a certain threshold (Yan 2015). This approach will further promote the performance of anomaly detection in practical uses. Second, observed from Table 4, Precision on observed samples is beyond 0.9 while Precision on unobserved samples is below 0.5. In addition, normal events are much more than anomalies in any software system. In other words, although Precision is as high as 50%, many normal events are considered as anomalies based on the classification of LogGAN. Therefore, LogGAN is still limited in practical use (i.e., detecting unobserved or unknown anomalies). Third, the performance of LogGAN on the session-level anomaly detection is weak. Although the logs (i.e., events) are temporal in each session (i.e., block) of HDFS, the capacity of a session (i.e., the number of logs) is from tens to hundreds and is unfixed. LogGAN is an LSTM-based deep learning approach that is weak for fitting the variable-length and long sequence. Therefore, observed from Table 3, Precision of LogGAN reaches 1 while Recall is low, in other words, LogGAN tends to consider session-level anomalies as normal sessions due to the imbalance between normal and abnormal sessions. That is the reason why the performance of LogGAN is weak in the session-level anomaly detection.

5 Conclusion

To overcome the limitation of diagnosing log-level anomaly detection, in this paper, we propose a sequence-based generative adversarial network to detecting abnormal events among system logs named LogGAN. In practical scenarios, we consider that the occurring anomalies depend on specific patterns which comprise the latest logs and regard specific patterns as the contextual information of upcoming logs. Due to the benefit of the generative adversarial network, the problem of the imbalance between normal and abnormal logs is relieved where LogGAN is capable of generating 'real' anomalies for supplying the lack of abnormal logs in system logs. In addition, LogGAN can be transformed into the session version only to changing the representation of samples without reforming the overall structure of LogGAN. The experimental results show the effectiveness of LogGAN on both the tasks of session-level and log-level anomaly detection.

The current LogGAN still has some problems that need to be solved for further improvement, and there exist several ideas to extend our work in the future. First, the current LogGAN has similar structures of discriminator and generator, and we exploit the generator to distinguish anomalies from system logs. Can the combination of outputs from discriminator and generator be used to identify anomalies? Second, only the signature and the temporal information of system logs are used to train LogGAN in this paper. The parameter of each event and other meaningful feature need to be considered to precisely describe anomalies. Third, the diagnosis of anomalies is also an important task which helps administrators solve anomalies efficiently. Therefore, the root cause analysis (RCA) should be considered in the process of detecting anomalies.

Acknowledgment This work was supported by the National Natural Science Foundation of China under Grant No.61802205, 61872186, and 61772284, the Natural Science Research Project of Jiangsu Province under Grant 18KJB520037, and the research funds of NJUPT under Grant NY218116.

References

Bodik, P., Goldszmidt, M., Fox, A., Woodard, D.B., Andersen, H. (2010). Fingerprinting the datacenter: automated classification of performance crises. In *inproceedings of the 5th european conference on computer systems* (pp. 111–124): ACM.

Chae, D.-K., Kang, J.-S., Kim, S.-W., Lee, J.-T. (2018). Cfgan: A generic collaborative filtering framework based on generative adversarial networks. In *Inproceedings of the 27th ACM International Conference on Information and Knowledge Management* (pp. 137–146): ACM.

Chandola, V., Banerjee, A., Kumar, V. (2009). Anomaly detection: a survey. *ACM computing surveys (CSUR)*, *41*(3), 15.

Chawla, S., & Sun, P. (2006). Slom: a new measure for local spatial outliers. *Knowledge and Information Systems*, *9*(4), 412–429.

Chen, M., Zheng, A.X., Lloyd, J., Jordan, M.I., Brewer, E. (2004). Failure diagnosis using decision trees. In *International Conference on Autonomic Computing, 2004. Proceedings* (pp. 36–43): IEEE.

Min, D., Li, F., Zheng, G., Srikumar, V. (2017). Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (pp. 1285–1298): ACM.

Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Bing, X., Warde-Farley, D., Ozair, S., Courville, A.C., Bengio, Y. (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada* (pp. 2672–2680).

Guo, S., Liu, Z., Chen, W., Li, T. (2018). Event extraction from streaming system logs. In *Information Science and Applications 2018 - ICISA 2018, Hong Kong, China, June 25-27th, 2018* (pp. 465–474).

Huang, S.Y., Lin, C.-C., Chiu, A.-A., Yes, D.C. (2017). Fraud detection using fraud triangle risk factors. *Inf. Sys. Frontiers*, *19*(6), 1343–1356.

Li, T., Zeng, C., Zhou, W., Xue, W., Huang, Y., Liu, Z., Zhou, Q., Xia, B., Wang, Q., Wang, W., et al. (2017). Fiu-miner (a fast, integrated, and user-friendly system for data mining) and its applications. *Knowledge and Information Systems*, *52*(2), 411–443.

Liang, Y., Zhang, Y., Xiong, H., Sahoo, R. (2007). Failure prediction in ibm bluegene/l event logs. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)* (pp. 583–588): IEEE.

Lin, Q., Zhang, H., Lou, J.-G., Zhang, Y., Chen, X. (2016). Log clustering based problem identification for online service systems. In *Proceedings of the 38th International Conference on Software Engineering Companion* (pp. 102–111): ACM.

Liu, F.T., Ting, K.M., Zhou, Z.-H. (2008). Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining* (pp. 413–422): IEEE.

Lou, J.-G., Qiang, F., Yang, S., Ye, X., Li, J. (2010). Mining invariants from console logs for system problem detection. In *USENIX Annual Technical Conference* (pp. 1–14).

Mondal, T., Pramanik, P., Bhattacharya, I., Boral, N., Ghosh, S. (2018). Analysis and early detection of rumors in a post disaster scenario. *Inf. Syst. Frontiers*, *20*(5), 961–979.

Niven, T., & Kao, H.-Y. (2019). Probing neural network comprehension of natural language arguments. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers* (pp. 4658–4664).

Sun, P., & Chawla, S. (2004). On local spatial outliers. Fourth IEEE International Conference on Data Mining (ICDM'04) (pp. 209–216): IEEE.

Tang, L., Li, T., Perng, C.-S. (2011). Logsig: generating system events from raw textual logs. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management* (pp. 785–794): ACM.

Troudi, A., Zayani, C.A., Jamoussi, S., Amor, I.A.B. (2018). A new mashup based method for event detection from social media. *Inf. Syst Frontiers*, *20*(5), 981–992.

Tuor, A.R., Baerwolf, R., Knowles, N., Hutchinson, B., Nichols, N., Jasper, R. (2018). Recurrent neural network language models for open vocabulary event-level cyber anomaly detection. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*.

Wang, J., Lantao, Y., Zhang, W., Gong, Y., Yinghui, X., Wang, B., Zhang, P., Zhang, D. (2017). Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval* (pp. 515–524): ACM.

Wang, W., Zeng, C., Li, T. (2018). Discovering multiple time lags of temporal dependencies from fluctuating events. In *Web and Big Data - Second International Joint Conference, APWeb-WAIM 2018, Macau, China, July 23-25, 2018, Proceedings, Part II* (pp. 121–137).

Xia, B., Yin, J., Jian, X., Li, Y. (2019). Loggan: A sequence-based generative adversarial network for anomaly detection based on system logs. In Liu, F., Xu, J., Xu, S., Yung, M. (Eds.) Science of Cyber Security - Second International Conference, Scisec 2019, Nanjing, China, August 9-11, 2019, Revised Selected Papers, Volume 11933 of Lecture Notes in Computer Science (pp. 61–76): Springer.

Jian, X., Jiang, Y., Zeng, C., Li, T. (2015). Node anomaly detection for homogeneous distributed environments. *Expert Syst. Appl.*, *42*(20), 7012–7025.

Jian, X., Tang, L., Li, T. (2016). System situation ticket identification using svms ensemble. *Expert Syst. Appl.*, *60*, 130–140.

Jian, X., Tang, L., Zeng, C., Li, T. (2016). Pattern discovery via constraint programming. *Knowl.-Based Syst.*, *94*, 23–32.

Wei, X., Huang, L., Fox, A., Patterson, D., Jordan, M.I. (2009). Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles* (pp. 117–132): ACM.

Yan, G. (2015). Be sensitive to your errors: Chaining neyman-pearson criteria for automated malware classification. In Bao, F., Miller, S., Zhou, J., Ahn, G.-J. (Eds.) Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '15, Singapore, April 14-17, 2015 (pp. 121–132): ACM.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J.G., Salakhutdinov, R., Le, Q.V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. CoRR abs/1906.08237.

Zeng, C., Tang, L., Li, T., Shwartz, L., Grabarnik, G. (2014). Mining temporal lag from fluctuating events for correlation and root cause analysis. In *10th International Conference on Network and Service Management, CNSM 2014 and Workshop, Rio de Janeiro, Brazil, November 17-21, 2014* (pp. 19–27).

Ji, Z., & Wang, H. (2006). Detecting outlying subspaces for high-dimensional data: the new task, algorithms, and performance. *Knowledge and information systems*, *10*(3), 333–355.

Zhu, J., He, S., Liu, J., He, P., Qi, X., Zheng, Z., Lyu, M.R. (2018). Tools and benchmarks for automated log parsing. CoRR abs/1811.03509.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Bin Xia received the PhD degree in Computer Science from Nanjing University of Science and Technology, in 2018. Currently, he is current an assistant professor with the School of Computer Science and Technology, Nanjing University of Posts and Telecommunications (NJUPT). His research interests include recommender system, AI for IT Operations, and deep learning.

Yuxuan Bai was born in 1995. He received a bachelor's degree from Nanjing University of Posts and Telecommunications in 2018. He is now a second-year graduate student at Nanjing University of Posts and Telecommunications. His current research direction is Nature Language Processing and Data Mining.He is a member of the Jiangsu Key Laboratory of Big Data Security and Intelligent Processing.

Junjie Yin was born in 1994. He received his master degree from Nanjing University of Posts and Telecommunications in 2020. He is now a senior engineer at Zhongxing Telecommunication Equipment Corporation. His current research directions are recommend system and deep learning.

Yun Li received the Ph.D. degree in computer science from Chongqing University, Chongqing, China. He was a Post-Doctoral Fellow with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China. He is a Professor with the College of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing, China. His current research interests include machine learning, data mining, and parallel computing. Dr. Yun Li is a senior member of CCF, JSAI and member of IEEE.

Jian Xu received a Ph.D. in Computer Science in 2007 from Nanjing University of Science and Technology, Nanjing, China. Now he holds the position of a professor at Nanjing University of Science and Technology. His research interests are event mining, log mining and their applications to complex system management, and he has published about 30 papers in journals and refereed conference proceedings in those areas.