Utah State University

DigitalCommons@USU

All Graduate Theses and Dissertations | Graduate Studies

8-2021

LogBERT: Log Anomaly Detection via BERT

Haixuan Guo
Utah State University

Follow this and additional works at: https://digitalcommons.usu.edu/etd

 Part of the Computer Sciences Commons

Recommended Citation

Guo, Haixuan, "LogBERT: Log Anomaly Detection via BERT" (2021). *All Graduate Theses and Dissertations*. 8110.
https://digitalcommons.usu.edu/etd/8110

This Thesis is brought to you for free and open access by the Graduate Studies at DigitalCommons@USU. It has been accepted for inclusion in All Graduate Theses and Dissertations by an authorized administrator of DigitalCommons@USU. For more information, please contact digitalcommons@usu.edu.

LOGBERT: LOG ANOMALY DETECTION VIA BERT

by

Haixuan Guo

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Data Science

Approved:

Shuhan Yuan, Ph.D. Major Professor	John Edwards, Ph.D. Committee Member
Heng-Da Cheng, Ph.D. Committee Member	D. Richard Cutler, Ph.D. Interim Vice Provost of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2021

Copyright © Haixuan Guo 2021

All Rights Reserved

ABSTRACT

LogBERT: Log Anomaly Detection via BERT

by

Haixuan Guo, Master of Science

Utah State University, 2021

Major Professor: Shuhan Yuan, Ph.D.
Department: Computer Science

System logs, which are usually produced by logging statements in source codes, play an important role to diagnose anomalous behaviors caused by system blackouts, malicious attacks, or performance degradation. Recently, due to the promising performance on computer vision and natural language processing, deep learning models are widely used for log anomaly detection. Most existing deep learning approaches adopt the recurrent neural network (RNN) and its variants to capture the sequential pattern of log data. Owning to the effectiveness of Bidirectional Encoder Representations from Transformers (BERT) in modeling sequential data, we propose LogBERT, a BERT-based neural network for log anomaly detection.

Specifically, LogBERT incorporates two self-supervised training tasks, Masked Log Key Prediction (MLKP) and Volume of Hypersphere Minimization (VHM). The goal of the MLKP task is to learn the contextual information in log sequences by predicting the randomly masked log keys. The VHM task maps the representation of normal log sequences in a hypersphere in which normal log sequences are concentrated around the center of the hypersphere while abnormal log sequences are far away from the center. After training on normal log sequences, LogBERT can detect abnormal log sequences that deviate from normal patterns. Our experiments on HDFS, BGL, and Thunderbird datasets show that

LogBERT can achieve better performance than the existing traditional machine learning and deep learning approaches. An ablation study shows that using two self-supervised training tasks in LogBERT is better than using solely one task.

(58 pages)

PUBLIC ABSTRACT

LogBERT: Log Anomaly Detection via BERT

Haixuan Guo

When systems break down, administrators usually check the produced logs to diagnose the failures. Nowadays, systems grow larger and more complicated. It is labor-intensive to manually detect abnormal behaviors in logs. Therefore, it is necessary to develop an automated anomaly detection on system logs. Automated anomaly detection not only identifies malicious patterns promptly but also requires no prior domain knowledge. Many existing log anomaly detection approaches apply natural language models such as Recurrent Neural Network (RNN) to log analysis since both are based on sequential data. The proposed model, LogBERT, a BERT-based neural network, can capture the contextual information in log sequences.

LogBERT is trained on normal log data considering the scarcity of labeled abnormal data in reality. Intuitively, LogBERT learns normal patterns in training data and flags test data that are deviated from prediction as anomalies. We compare LogBERT with four traditional machine learning models and two deep learning models in terms of precision, recall, and F1 score on three public datasets, HDFS, BGL, and Thunderbird. Overall, LogBERT outperforms the state-of-art models for log anomaly detection.

ACKNOWLEDGMENTS

I would like to express my gratitude towards my advisor, Dr. Yuan for his unfaltering support, insightful advice, and patient guidance. During the past year working with him, I gained valuable research experience. When I run into problems in the experiments, his thoughts inspired me to find different solutions.

I would like to extend my thanks to the committee members, Dr. John Edwards and Dr. Hengda Cheng. Their opinions are important for my research. Besides, I would like to thank Cora Price, Kaitlyn Fjeldsted, Genie Hanson, and Vicki Anderson. They are great supports for me to successfully finish my degree.

Finally, I wish to thank my family, particularly my mom who believes in me and loves me unconditionally. I also want to thank my friends for listening to me and supporting my decisions.

I am so grateful to have the opportunity to study at USU. It was a wonderful journey.

Haixuan Guo

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

AE	Autoencoder
BERT	Bidirectional Encoder Representations from Transformers
BGL	BlueGene/L Supercomputer System
CNN	Convolutional Neural Network
DAE	deep autoencoder
DAE-DBC	Deep autoencoders with density-based clustering
GRU	Gated Recurrent Units
HDFS	Hadoop Distributed File System
IDF	Inverse Documentary Fre-quency
KNN	K Nearest Neighbor
LSTM	Long short-term memory
MLKP	Masked LogKey Prediction
MLM	Masked LM
MLP	Multilayer Perception
MSE	mean squared error
NLP	natural language processing
NSP	Next Sentence Prediction
OCSVM	One-Class SVM
PCA	Principal Component Analysis
RNN	Recurrent Neural Network
Spell	Streaming structured Parser for Event Logs using LCS
SVM	Support Vector Machine
VHM	Volume of Hypersphere Minimization

CHAPTER 1

Introduction

1.1 Overview and Motivation

1.1.1 Background

Anomaly detection is an important topic discussed across various research areas and application domains. Anomalies are defined as data instances that stand out as being dissimilar to all others [7]. Anomaly detection indicates the problem of discovering patterns that do not comply to expected behaviors [1]. In literature, anomaly detection can also be described as outlier detection, novelty detection, noise detection, deviation detection, or exception mining [8].

Detecting anomalies is an imperative task for many security-critical environments. For example, in a computer system, anomaly detection monitors system calls, event occurrence, and network traffic to identify malicious activities or intrusions [9]. The key challenges in intrusion detection are the huge volume of data and data arriving in a streaming fashion. Another common anomaly in industries is fraud [10]. Fraud is considered a criminal act in commercial organizations. The typical method for fraud detection is to maintain a profile for each customer and monitor the profiles to detect any deviations. Fraud detection requires intermediate intervene to prevent profit loss. The anomaly in medical and public health domain is disease outbreak [11]. The detection techniques in this area aim to find rare events in patients' records such as medical images, which can provide early diagnoses and treatments to prevent disease progression. In a social network, an anomaly refers to behavior patterns of users that differ from normal patterns. Abnormal users are recognized as spammers, sexual predators, online fraudsters, fake users, or rumor-mongers [1]. Internet of Things (IoT) is a network of devices that connects software, servers, and sensors. Anomaly

detection in the IoT network detects faulty behaviors in those large-scale interconnected devices. A major challenge in this domain is that heterogeneous devices are connected mutually, which renders the system more complicated [12].

Human-based anomaly detection requires domain knowledge and is labor-intensive, which is impractical nowadays as a large volume of data is continually produced. Therefore, data-driven approaches are developed to automatically identify anomalies. Statistical knowledge and information theory are first applied to design anomaly detection models. Machine learning algorithms including K Nearest Neighbor (KNN), decision tree, and Support Vector Machine (SVM) provides an interpretable solution for anomaly detection problems. The procedure for log anomaly detection using machine learning techniques often involves log parsing, windowing, feature extraction, and detection modeling [3]. With the extensive usage of deep learning techniques for real-world problems, deep neural networks such as Long short-term memory (LSTM), Autoencoder, Convolutional Neural Network (CNN) are employed to detect anomalies in specific domains.

The anomaly detection techniques depend on various factors, such as the nature of input data, availability of labeled data, and the output of the anomaly detection model.

The nature of input data is the characteristic of data such as continuous and categorical. Considering the number of features, input data can be also classified as low or high-dimensional data. The nature of input data primarily determines the application of anomaly detection approaches. For example, in deep learning techniques, input data can be extensively classified into sequential data involving voice, text, music, and time series, or non-sequential data such as image [7]. The corresponding techniques for sequential data are CNN and Recurrent Neural Network (RNN), and the applicable techniques for non-sequential data are CNN and Autoencoder.

A label denotes whether a data instance is normal or abnormal. It is hard to obtain anomalous labels since there are few abnormal events occurring in a real-life situation. Additionally, the nature of anomalies is dynamically evolving. Based on the extent of availability of labels, anomaly detection techniques are classified into three modes: supervised anomaly

detection, semi-supervised anomaly detection, and unsupervised anomaly detection.

Currently, there are two forms of output to report anomalies, namely anomaly scores and binary labels. An anomaly score describes the degree to which an instance is considered an anomaly. To identify anomalies, all data are ranked in descending order with respect to anomaly scores, then a domain-specific threshold is defined by analysts. In some techniques, a binary categorical label is assigned as normal or anomalous to each data instance.

Log anomaly detection is a crucial application for anomaly detection. System logs, which record detailed run-time states and events during system execution, provide valuable sources for system anomaly detection. Logs consist of log messages (also commonly called log entries) generated by logging statements (print, logging.info) in source code. Log anomaly detection exploits log messages to detect anomalous behaviors in systems caused by hardware failure, energy consumption, or the environment [13].

The commercial application of anomaly detection is booming over the years. According to the *Global Anomaly Detection Industry* report, the global market of anomaly detection in 2020 was estimated at 2.8 billion dollars and is projected to escalate to 7.8 billion by 2027 [14]. Various large-scale online services bring convenient life to millions or even billions of users globally. Every second, user across the world rely on online systems for work, school, and entertainment. It is essential for enterprises to provide reliable and secure systems for users. Any incident that occurs in these systems will lead to unpredictable profit loss. To prevent potential risks happened in systems, it is important to reveal abnormal behaviors of systems timely. Timely anomaly detection helps system operators to pinpoint incidents and revolve them promptly.

1.1.2 Types of Anomaly

Anomalies are also referred to as abnormalities, deviants, or outliers in the data mining and statistical analysis [7]. Anomalies of logs exist not only in individual log entries but also in log sequences encompassing a series of log entries. According to anomaly detection reviews [1, 7], we define three types of anomalies: point anomalies, contextual anomalies, and group anomalies.

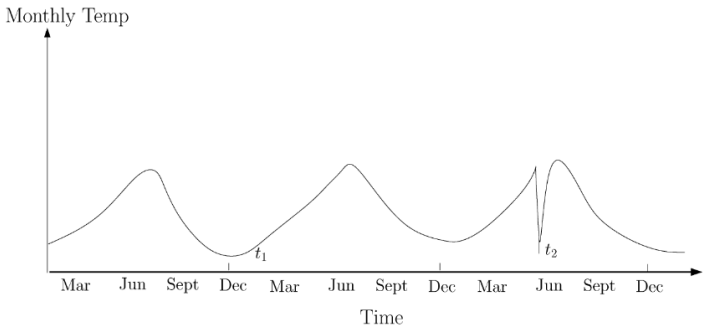


Fig. 1.1: Contextual anomaly in a temperature time series [1]

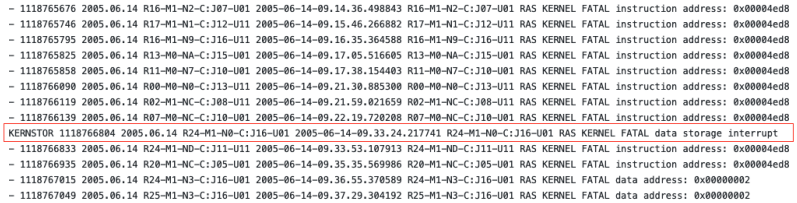


Fig. 1.2: Contextual anomaly in BGL log data

Point Anomalies

An instance is termed a point anomaly if the instance significantly deviates from the distribution of the rest of data. In log anomaly detection, point anomalies are outliers of parameters in log entries. For example, an anomalous log entry is "Created block blk_-8192956077351896648 of size 335544320 from /10.251.110.196". The block size 335544320 exceeds a reasonable range for block size. This type of anomaly is easy to detect as it often triggers programs to throw exceptions that can be discovered intermediately.

Contextual Anomalies

An instance is termed a contextual anomaly when its behavior is considered anomalous in a specific context and normal in other contexts [15]. This type of anomaly is also known as conditional anomalies. A data instance is defined as a contextual anomaly by considering both contextual and behavioral attributes. Contextual attributes are normally spatial or temporal features used to determine the context (neighborhood) of a data instance [15]. Behavior attributes, on the other hand, are non-contextual characteristics of data instances.

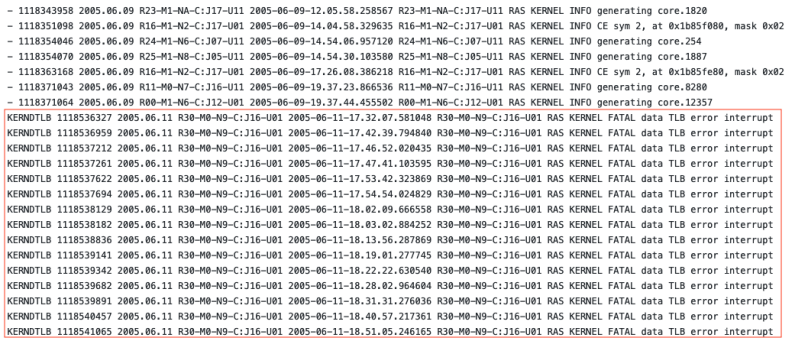


Fig. 1.3: Group anomalies in BGL log data

Figure 1.1 shows an example of a contextual anomaly in a temperature time series. Note that the temperature at time $t2$ is an anomaly, but the same temperature at time $t1$ is normal. This is because a low temperature is normal during the winter, but unusual during the summer. An example of contextual anomalies in BGL log data is illustrated in Figure 1.2. Note that "data storage interrupt" is treated as an anomaly in the middle of normal operations such as "instruction address".

Group Anomalies

Group anomalies also referred to as collective anomalies, appear as an anomalous group regarding the entire data set. Data instances may not appear as abnormal individually, but their presence together in a group is abnormal. Group anomalies in log data are log entries in a sequence exhibiting unusual characteristics. Figure 1.3 shows an example of group anomalies in BGL log data. Note that "data TLB error interrupt" is normal on its own, but a collection of "data TLB error interrupt" logs consistently produced seem to be a group anomaly.

1.1.3 Challenges

- Logs are mainly unstructured or semi-structured text data [16] that are hard for machines to interpret. The format and syntax of log messages vary across different software systems, making it challenging to adopt one universal technique to parse logs into structured data.

- It is difficult to define a precise boundary to separate normal logs from anomalous logs. The intrinsic characteristics of the boundary are dynamically evolving, which poses challenges for both machine learning-based and deep learning-based algorithms [17].

- Log data are imbalanced, as the majority of log instances are normal. Therefore, it is suboptimal to train a binary classifier for log anomaly detection [7].

- Log anomaly detection approaches should be conducted timely so that system operators or developers can intervene in ongoing attacks or system performance issues [18]. Applicable approaches for log anomaly detection are required to make decisions after training on a small number of logs.

1.1.4 Motivation

The existing log anomaly detection approaches can resolve one or more challenges mentioned above.

Currently, many traditional machine learning models are proposed for identifying anomalous events from log messages. These approaches extract useful features from log messages and adopt machine learning algorithms to analyze the log data. Due to the data imbalance issue, it is infeasible to train a binary classifier to detect anomalous log sequences. As a result, many unsupervised learning models, such as Principal Component Analysis (PCA) [19], or one class classification models, such as one-class SVM [20, 21], are widely-used to detect anomalies. However, traditional machine learning models, such as one-class SVM, are hard to capture the temporal information of discrete log messages.

Recently, deep learning models, especially RNNs, are widely used for log anomaly detection since they can model the sequential data [18, 22, 23]. However, there are still some limitations of using RNN for modeling log data. First, although RNN can capture the sequential information by the recurrence formula, it cannot make each log in a sequence encoding the context information from both the left and right context. However, it is crucial to observe the complete context information instead of only the information from previous steps when detecting malicious attacks based on log messages. Second, RNN is

trained by feeding log sequences one by one and cannot explicitly leverage the information from other normal log sequences during training. Since normal log sequences usually share similar patterns, if the model can observe all the information regarding normal sequences, we expect the model can achieve better performance for log anomaly detection.

To tackle the existing limitations of RNN-based models, in this work, we propose LogBERT, a self-supervised framework for log anomaly detection based on Bidirectional Encoder Representations from Transformers (BERT). Inspired by the great success of BERT in modeling sequential text data [6], we leverage BERT to capture patterns of normal log sequences. By using the structure of BERT, we expect the contextual embedding of each log entry can capture the information of whole log sequences. To achieve that, we propose two self-supervised training tasks: 1) masked log key prediction, which aims to correctly predict log keys in normal log sequences that are randomly masked; 2) volume of hypersphere minimization, which aims to make the normal log sequences close to each other in the embedding space. After training, we expect LogBERT encodes the information about normal log sequences. We then derive a criterion to detect anomalous log sequences based on LogBERT. Experimental results on three log datasets show that LogBERT achieves the best performance on log anomaly detection by comparing with various state-of-the-art baselines.

1.2 Contributions

The novelty of our research involves an optimized objective function and a novel model design for log anomaly detection. Specifically, our contributions are:

- We propose a BERT-based framework that takes log key embedding and position embedding as input and adopts a Transformer encoder to learn the contextual relations in log sequences.

- We propose two self-supervised training tasks to capture patterns of normal log sequences, namely MLKP and VHM respectively. The MLKP task can accurately

predict masked log keys in log sequences. The VHM task is used to minimize a data-enclosing hypersphere.

- The proposed approach achieves the state-of-art performance on three log datasets in contrast to baselines including several traditional machine learning models and two state-of-the-art RNN-based models.

1.3 Thesis Organization

The thesis is organized as follows. In Chapter 2, we provide a comprehensive review of prior work on log preprocessing and log anomaly detection models. Chapter 3 describes preliminary concepts, including DeepLog, Transformer, and BERT. Chapter 4 demonstrates the architecture of LogBERT as well as two training tasks. In Chapter 5, we present three public log datasets applied in our experiments and demonstrate experiment results. In Chapter 6, we summarize our findings and discuss future works.

CHAPTER 2

Related Work

2.1 Log Preprocessing

Log preprocessing converts raw log entries into structured data such as vectors. This process consists of log parsing and log sequence extraction.

Log Parsing

Log parsing is the first and crucial step to parse log messages into structured data for the subsequent log analysis. Each raw log message can be divided into two parts: constant and variable [24]. The constant part is a template with fixed texts that remain the same in every log entry, while the variable part contains parameters that dynamically change in every event occurrence. The constant part can be referred interchangeably as log key [18], log template [22], or log event [25]. The objective of log parsing is to extract the constant part and the variable part from log messages. As an example in Figure 2.1, log messages are parsed into log keys, where the constant part remains unchanged and the variable part is replaced with asterisks.

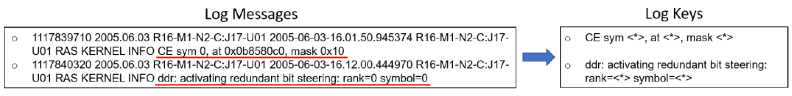


Fig. 2.1: Log messages in the BGL dataset and the corresponding log keys extracted by a log parser. The message with red underscore indicates the detailed computational event.

The traditional way of log parsing is based on handcrafted regular expressions [26] to match log keys and parameters in log messages. However, due to the massive amount of logs and frequent evolution in log statements, it is time-consuming to parse logs with manual rules. Since logs are generated from print statements in source code, some studies [27, 28]

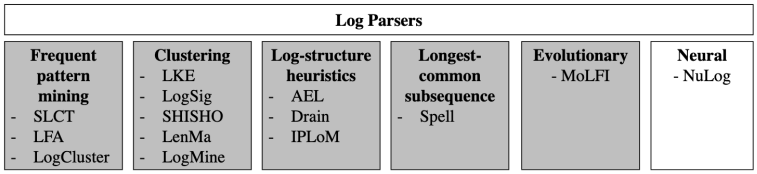


Fig. 2.2: Categories of log parsers [2]

have explored static methods to extract log keys from source code directly. The static methods are invalid if the source code can not be accessed, especially when importing third-party libraries.

In contrast to extensive manual efforts from rule-based and source code-based parsing, data-driven approaches can automatically extract log keys by learning patterns from log data. S. Nedelkoski et al. [2] summarize the characteristic of thirteen existing automated log parsers and categorized those parsers into six categories based on the parsing strategies, which are clustering, frequent pattern mining, evolutionary, log-structure heuristics, longest-common sub-sequence, and neural. Clustering methods assume that log messages with close string matching distances are in the same group. The clustering distance can be defined as edit distance, the word length in log messages, and so on. The representative log parsers in this category are LKE, LogSig, SHISO, LenMa, and LogMine [29–33]. The main assumption for the frequent pattern mining category is that a message can be represented by a set of frequent tokens across logs. The typical parsers in this category are SLCT, LFA, and LogCluster [27, 34, 35]. The only member in the evolutionary category is MoLFI [36] which searches all solutions to find the Pareto optimal set of log messages as the log event. Another category uses the longest common subsequence algorithm to dynamically extract log keys and group log messages. Spell [37] is the most representative technique in this category. The parsers in log-structure heuristics methods are Drain, IPLoM, and AEL [38–40]. Those log parsers exploit the properties of log structures. For example, Drain assumes message length and the preceding tokens in log messages are important properties to produce distinguishing log keys. Neural log parsing is a novel idea that trains a neural network to generate log keys [2]. Figure 2.2 demonstrates all six categories and the corresponding approaches.

Both Spell and Drain are highly efficient when processing a large volume of log data. They can be deployed online to parse logs one by one instead of processing all after collecting the entire log data [34, 36, 39, 40]. Due to the practical application of Spell and Drain, we specifically introduce these two log parsers.

Spell. Streaming structured Parser for Event Logs using LCS (Spell) computes the longest common substring (LCS) of new log entries to the existing log key candidates. If the maximum of LCS is greater than a threshold, new entries are parsed to the log key candidate with the maximum LCS.

Drain. Drain is a state-of-the-art online log parsing method that builds a parse tree with fixed depth. The assumption of Drain is that log messages yielding the same log event have the same text length and constant proceeding tokens. Before parsing, regular expression scripts are written to remove common variables such as IP address, file path. A parse tree starts with nodes that use log message length as node value. Following the first layer, the parse tree grows by taking a fixed number of preceding tokens in log keys as nodes. A leaf node in the parse tree is a log group that includes a log event and a list of log IDs. When a new log entry reaches a leaf node in the tree, it calculates similarity scores between the new entry and a log event in the group. If similarity scores are beyond a threshold, then it adds the new log entry to the log group. Otherwise, it creates a new log group for the log entry.

After obtaining log events through parsing, there are different ways to transform log events into embedding vectors. Research in [3, 19, 37, 41, 42] use one-hot encoding to transform log events into numerical values which are later mapped into embedding vectors. Research in [22, 25] represent log events with semantic embedding vectors that extract semantic information in log messages. Nagappan et al. [27] propose to vectorize log events with a normalized IDF-based weight and a contrast-based weight.

Log Sequence Extraction

Logs are produced in an unbounded streaming fashion, whereas anomaly detection models only receive bounded inputs. Therefore, we divided streaming logs into sequences

by windowing. Research in [3, 22, 25, 37] extract log sequences by one of three windowing techniques, fixed window, sliding window, and session window.

Fixed Window. A fixed window assigns log keys into a window based on the timestamp when log keys occur. Window size t is time span or time duration, such as five minutes or one hour. A log sequence is constituted by log keys that appear in the same window. Log keys in a fixed window are non-overlapping, which implies that a log key exists in only one window. For example, if a fixed window with a size of five minutes is specified, a new window will start every five minutes.

Sliding Window. In addition to window size, a sliding window is determined by step size that is the sliding distance of a window. For instance, hourly window sliding every five minutes [3]. Step size is usually smaller than the window size. Hence, sliding windows can be overlapping which means a log key occurs in more than one window.

Session Window. Apart from timestamp-based windowing, session window groups log keys into different log sequences (also called log sessions) by identifiers. Identifiers are used to distinguish different execution paths in the same workflow. For instance, in HDFS logs, log sessions are identified by block_id. Each session records a life cycle of a block including allocation, replication, and deletion [3].

2.2 Traditional Machine Learning-based Approaches

2.2.1 Supervised Machine Learning

Conventional supervised approaches for log anomaly detection are mainly SVM, KNN, and Decision Tree. Those approaches take log key count vectors as input and binary labels as classification results.

SVM. SVM algorithm maps data instances in high dimension and constructs optimal hyperplanes to separate different classes. The optimal hyperplane has the maximum margin from the nearest points (also called support vectors) in various classes. In order to find optimal hyperplanes, kernel functions are employed such as linear kernel, polynomial kernel, and radial basis function kernel.

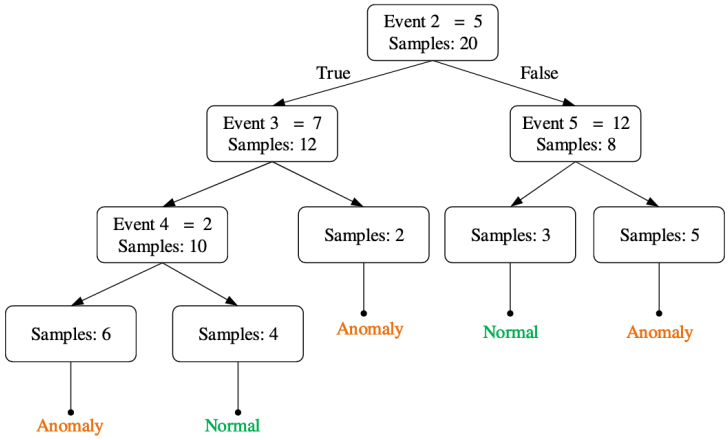


Fig. 2.3: An example of a decision tree for log anomaly detection [3]

KNN. KNN algorithm adopts the "majority voting" strategy. Given an unlabeled point p, the KNN algorithm selects k training points nearest to p. p is designated to the class that contains the majority of k training points. A common metric to calculate the distance between points is Euclidean distance. A drawback of using KNN in classification problems is that KNN is sensitive to skewed data. When applying KNN to log anomaly detection, normal logs tend to dominate the prediction of new samples. It is necessary to take measures such as sampling to balance data distribution. [41] trained an SVM classifier and a KNN classifier to diagnose system failures. The evaluation result shows KNN has better performance than SVM.

Decision Tree. Decision tree algorithm builds a top-down tree structure using training data. Each node is split into two branches by selecting the current best features. The best feature produces high information gain compared with other features. The division of nodes terminates when the information gain is 1, which indicates all samples in a node belong to the same class. For log analysis, log keys are features for each node. An example about building a decision tree with log data is illustrated in Figure 2.3. Chen et al. [42] utilize decision trees for failure diagnoses. They select features from log repositories that record software information and network requests.

2.2.2 Unsupervised Machine Learning

Unlike supervised learning, unsupervised learning doesn't require labels during training. Hence, it is applicable to use unsupervised approaches to resolve real-world issues. For log anomaly detection, common unsupervised approaches are PCA, clustering-based methods, and so on.

PCA. PCA is commonly used for dimension reduction. Xu et al. [19] first apply PCA to log anomaly detection. PCA projects training data to generate a normal space S_n and an abnormal space S_a. S_n is constructed by the first k principal components. S_a is constructed by the remaining $(n-k)$ components, where n is the original components. Given an event count vector y from a new log entry, its projection to S_a is denoted by $y_a = (1 - PP^T)y$, where P is the first k principal components. If the project y_a is larger than a threshold, we consider the log entry as an anomaly.

Isolation Forest. The basic idea of the Isolation Forest algorithm is that anomalous data instances tend to isolate from the rest of the data, compared to normal instances. In order to isolate a data instance, the algorithm recursively divides data by randomly selecting a feature and a valid split value for the feature. This partitioning is similar to tree generation, therefore it is called Isolation Tree. In an Isolation Tree, the path length of a data instance is defined as the length from the root to a leaf node. A data instance having a shorter average path length is regarded as an anomaly. Isolation Forest consists of an ensemble of Isolation Trees.

Invariants Mining. Invariants [43] in logs refers to the execution flow among normal log sequences. A invariant presents a linear relationship among different log messages (log keys) in a log sequence. The invariant is denoted as vector θ. Invariant mining discovers the invariants in log message groups which describes a program execution path related to the program variable. Then a matrix X is obtained from count vectors of log messages in the same program variable. The matrix is decomposed by singular value decomposition (SVD) to generate invariant space. The invariant space is a span of validated right-singular vectors. Intuitively, an anomaly manifests different execution flow from the normal ones,

which violates the invariants in system executions.

LogCluster. LogCluster [44] employs two training phrases, knowledge base initialization phrase and online learning phrase, to generate clusters. Training data is divided into two portions for the two phrases. In knowledge base initialization phrase, an event count matrix is first computed from training log sequences, then weighted by Inverse Documentary Frequency (IDF). After log vectorization, agglomerative clustering is used to group normal and abnormal clusters respectively as knowledge base. The centroid for each cluster is defined as its representation vector. The objective of online learning phrase is to adjust clusters obtained from the first phrase. Given an event count vector from the remaining training data, we compute the distances between the vector and representation vectors of existing clusters. If the minimum distance is greater than a threshold, a new cluster is created for this event count vector. Otherwise, the event count vector is added to the nearest cluster and the representation vector for the cluster is updated. The two phrases map training data into normal clusters and abnormal clusters separately. In the testing stage, a new log sequence is classified as normal/abnormal if its nearest cluster is normal/abnormal.

OCSVM. One-Class SVM (OCSVM) is an extension of SVM algorithm for novelty detection [45]. The OCSVM algorithm leverages normal data for training and maps training data into a high-dimensional feature space via kernels. The algorithm separates training data from the origin with the maximum margin hyperplane. For a new point, if it falls on the side of the hyperplane close to training data, it is labeled as normal. Otherwise, it is abnormal. The OCSVM can be considered as a special two-class SVM where training data lies in the first class, and the origin is the only member in the second class [46]. OCSVM is widely used for log anomaly detection [20, 21] by only observing normal data.

2.3 Deep Learning-based Approaches

2.3.1 RNN-based Approaches

Logs are produced by programs executing in a logic flow, which is similar to text generation in natural language processing (NLP). Inspired by the remarkable success of

NLP research using RNN-based approaches, many scholars applied RNN and its variants such as LSTM and Gated Recurrent Units (GRU) to log anomaly detection. LSTM is widely used in log anomaly detection as it can resolve the issue of gradient vanishing in a recurrent structure and keep long-term information in sequence. LSTM employs a cell state to "memorize" relevant information throughout the sequences. Additionally, forget gate, input gate, and output gate are designed in a LSTM unit to determine the relevance of historical information to current inputs. Forget gate decides the information to be discarded from previous hidden state and current input data. Input gate allows what information passes to the current cell state. Output gate, combined with the new cell state, generates the current hidden state as output.

LogAnomaly. Instead of using one-hot encoding to represent log keys, Meng et al [22] propose template2vec to convert log keys into template vectors. The template2vec uses a distributional lexical-contrast embedding model called dLCE [47] to generate vectors to represent words in log keys. The intuition of template2vec is that new templates are minor variants of existing templates. By merging new templates into existing templates with similar semantics information, template2vec automatically generates template vectors for new log templates. The log sequences are extracted by a sliding window. The underlying patterns of log sequences incorporate sequential patterns and quantitative patterns. Sequential patterns indicate the order and concurrence among log templates, while quantitative patterns are count vectors of unique log keys in log sequences. Both patterns can be captured by LSTM structures that are designed for sequential data.

RobustLog. The existing approaches are not robust to the instability of log data due to the evolution of log statements and processing noises. To address this issue, Xu et al. [25] proposed a new deep learning approach called RobustLog, which mapping the log events into semantic vectors by tokenization, word embedding, and TF-IDF. It then trains an attention-based bidirectional LSTM model to detect anomalies in log sequences. RobustLog can not only eliminate the instability of logs by extracting semantic information from log events, but also capture bidirectional sequential information in log sequences.

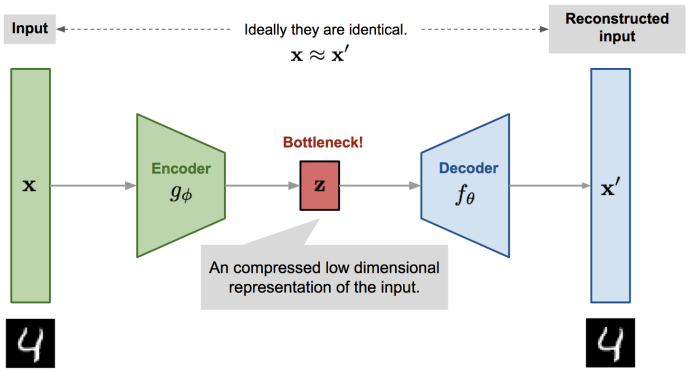


Fig. 2.4: Illustration of autoencoder model architecture [4].

2.3.2 AE-based Approaches

An autoencoder (AE) is an unsupervised neural network that learns the compressed representation of input data and reconstructs original data using the representation. Figure 2.4 demonstrates the architecture of an autoencoder. An autoencoder is used to reduce data dimensions by training the network to ignore noises. A typical autoencoder consists of three parts: an encoder to compress high-dimensional input data into a low-dimensional representation, a bottleneck layer that contains the representation of input data, and a decoder to reconstruct inputs from representations. The objective function for an autoencoder is to minimize the reconstruction (residual) error between the inputs and the outputs, which can be denoted by $loss = \|x - \hat{x}\|^2$, where x is the input data and $\hat{x}$ is the reconstructed data. When applied to anomaly detection, an autoencoder is solely trained on normal data instances, thus leading to a high residual error when reproducing abnormal data [7]. A data instance that produces a large reconstruction error is regarded as an outlier.

DAE-DBC. Deep autoencoders with density-based clustering (DAE-DBC) [48] is proposed to detect novelties in various domains such as intrusion detection, fraud detection, unusual event detection, disease condition detection etc. The proposed approach is comprised of two stages: dimension reduction and identification of novelties. In the first stage, two autoencoder models are adopted to find an optimized error threshold and the representation of input data. The autoencoder model has two dense layers in the encoder part and decoder part respectively. The first autoencoder is trained on all data including normal data

and novelties. An initial threshold is determined based on the histogram of reconstruction errors. The second autoencoder model is first trained by data close to the normal values. Then a final threshold to separate normal and novelties is computed by using the same thresholding technique to the reconstruction error from the second autoencoder model. After that, all data are fed into the second autoencoder model to obtain the final reduced representation and reconstruction error.

The novelty detection in the proposed approach applies density-based clustering to group data instances using low-dimensional feature space from the final representation. The idea of density-based clustering is that high-density regions can be separated from low-density regions. DBSCAN [49], a simple and effective density-based clustering algorithm, is used to find arbitrary shape data. DBSCAN requires two user-defined parameters, eps which determine a maximum radius of the neighborhood, and minPts which determine a minimum number of points in the eps of point [48].

After grouping all data points into different clusters via DBSCAN algorithm, the next step is to find out which clusters are novelties. The error threshold obtained from the second autoencoder model determines whether an instance is a novelty. If the majority of data instances in a cluster exceeds the threshold, the entire cluster is considered a novelty.

DAE. Nolle et al. [50] propose a deep autoencoder (DAE) approach to detect and analyze anomalies occurring during the execution of a business process. The method doesn't require prior knowledge of the business process and can train on noisy data containing anomalies. The training data are event logs that record execution traces of the system process and have no labels. The proposed method differentiates normal and anomalous executions purely based on patterns in log data. Anomalies (noises) are injected into event logs by randomly applying mutation to a fixed portion of execution traces. By adjusting the ratio of anomalous traces in event logs, training sets with different noise levels can be generated to test the autoencoder's generalization ability. Each activity and user in an event are individually encoded as an n-dimensional vector using one-hot encoding, where n is the total number of unique activities or users in the event logs. The input vector is an event

vector concatenating activity vectors and user vectors in the event. The target output for an autoencoder is the original input. The autoencoder architecture adopts multiple feed-forward neural network layers in the encoder part. To overcome overfitting in training data, a special layer is added to distribute Gaussian noise over input vectors. DAE approach assumes that autoencoder will reproduce abnormal executions with larger reconstruction errors than the normal ones. As shown in Equation 2.1, a threshold τ to classifies log events is defined by the mean reconstruction error over the training data, then scaled by a factor α. An event is considered an anomaly when the reconstruction error of the event is beyond τ.

$$\tau = \frac{\alpha}{n} \sum_{i=i}^{n} e_i \tag{2.1}$$

2.3.3 CNN-based Approaches

CNN leverages kernels in convolutional layers to extract hidden features for both sequential data and image data. A convolution operation in CNN uses a weighted matrix (also referred to as a kernel or a filter) to go over each pixel in an image. After aligning the center of the kernel to a pixel in the image, a pixel-wise multiplication is performed to calculate the weighted average of the neighborhood around the pixel. Then the pixel is replaced with the weighted average. Convolutional layers learn both low and high-level features from input data. CNN-based approaches for log anomaly detection are an on-going research area [51–53].

Shallow CNN. Lu et al. [51] propose a CNN-based framework to detect anomalies in system logs. The framework consists of a log key embedding layer, three convolutional layers, a dropout layer, a max-pooling layer, and finally a binary classifier. The classifier is composed of a fully connected layer and a softmax layer. The fully connected layer concatenates outputs from convolutional layers. The softmax layer is used to compute the distribution probability of results. This shallow CNN framework achieves higher and faster detection accuracy on HDFS logs compared to Multilayer Perception (MLP) and LSTM approaches.

Deep CNN. A deep CNN model is developed by Ren et al. [52] for log classification. The authors also propose a novel and effective log processing method which includes event categories labeling, event redundancy filtering, and numeric semantic feature vector generation. After log processing, log events are transformed into semantic embedding vectors. The vectors are then fed into a deep CNN model with five convolutional layers, three fully connected layers, and a softmax layer. The feature map at each layer captures different features from log sequences detected by the deep CNN model.

CHAPTER 3

Preliminary

3.1 DeepLog

DeepLog [18] adopts LSTM neural networks for online anomaly detection over system logs. In DeepLog, a model is trained on normal log keys to learn underlying patterns of normal execution paths. Anomalies are regarded as deviations from normal patterns learned by the model. In addition to log key anomalies that occur in execution paths, sometimes anomalies are shown as outliers in parameter values of log entries. DeepLog constructs an LSTM model to detect parameter value anomalies for each log key. With the two anomaly detection models, DeepLog is able to perform detection on session level as well as log entry level. In the training stage, DeepLog only consists of normal log sequences.

3.1.1 Log Key Anomaly Detection Model

A log key anomaly detection model detects anomalies in execution paths. The total number of distinct log keys in a program is constant. Let $k = \{k_1, k_2, ..., k_n\}$ be the set of distinct log keys. Let m_i denote the log key at position i in a log sequence. m_i is highly dependent on the most recent log keys that occur before m_i. The intuition of the log key detection model is that given a window w of h recent log keys, we want to predict if an incoming log key m_i is the next log key that follows the historical window w. The input of the model is $w = \{m_{t-h}, ..., m_{t-2}, m_{t-1}\}$. The output of the model is a conditional probability distribution $Pr[m_t = k_i|w]$ for each log key $k_i \in K$ $(i = 1, 2, ..., n)$ as the next log key. There are multiple log keys that can appear as m_i and follows normal patterns. Therefore, we select top g candidates from the output by sorting all log keys based on their probabilities. If the incoming log key is in top g candidates, we treat the log key as normal; otherwise, the log key is flagged as abnormal.

3.1.2 Parameter Value Anomaly Detection Model

When a log key is detected as normal by a log key anomaly detection model, a further inspection is performed on the parameter values in the log entry where the log key is extracted. Parameter values often reflect important metrics about systems. The parameter value anomaly detection model can detect performance degradation such as "slow down" in systems. In practice, a log entry stores different types of parameter values. The parameter value anomaly detection model puts the parameter values in a vector and treats the parameter value vectors as multivariate time series, which can be trained on an LSTM neural network. A parameter value model is similar to a log key model. However, instead of having a log key at every time step, a parameter value model receives a parameter value vector as the input and produces a real value vector as the prediction for the next value vector. In the training process, mean squared error (MSE) is used to minimize the loss between predicted vectors and the actual ones. Intuitively, the MSE between predictions and observations can be modeled as Gaussian Distribution. If the MSE of an incoming vector falls outside of the high confidence interval of the Gaussian Distribution, the vector is considered abnormal.

3.2 Transformer

RNN-based autoencoder models have proven their effectiveness in various sequence-to-sequence tasks such as machine translation. Variants of RNN, such as LSTM [54] or GRU [55], address the vanishing gradient issues, making RNN-based neural networks a priority for sequential modeling.

In 2014, Ilya Sutskever et al. [56] applied encoder-decoder structure to machine translation. The encoder part is an LSTM model that encodes information from the input sequence into a fixed-length vector, while the decoder part is another LSTM model that produces the output sequence. However, since the length of input sequences is uncertain, it is hard for the decoder to generate reasonable output sequences of arbitrary length. The problem was solved by Bahdanau et al. [57] in 2016. They proposed an attention mechanism to focus on the content of sentences.

Either encoder-decoder or attention mechanisms are based on RNN frameworks. The

recurrence of RNN limits the computation efficiency as it takes a long time for data to propagate. Therefore, some scholars decided to disperse recurrent structures. In 2017, the Facebook AI team [58] adopted CNN in sequence-to-sequence learning by using convolution kernels to extract the information of sequences. Although CNN-based approaches can be used in parallel computing, it is constrained by the long-distance dependency in sequences. In the same year, Google Brain published a paper [5], *Attention is All You Need*, where a new network architecture called Transformer was proposed and based solely on attention mechanisms.

The Transformer architecture (See Figure 3.1) uses encoder-decoder attention mechanisms. It has six stacked identical layers on both the encoder side and decoder side. Each layer contains two sublayers. The first sublayer is a multi-head self-attention layer and the second sublayer is a position-wise feed-forward neural network. Moreover, each sublayer is followed by a residual connection and layer normalization layer. The self-attention is calculated as a scaled dot-product of query, key, and value. The multi-head attention linearly projects query, key, and value multiple times. In order to maintain the relative or absolute positions of tokens in sequences, the Transformer adopts sine and cosine functions as position encoding.

3.2.1 Self-attention

Self-attention uses attention mechanisms to infer the relations among tokens in a sentence and learns the representation of the same sentence. In Transformer, self-attention is implemented by scaled dot-product attention and multi-head attention.

Scaled Dot-Product Attention. Scaled dot-product attention computes the dot products of query and key of dimension d_k, then scales the dot products by $\frac{1}{\sqrt{d_k}}$, and applies a softmax function to gain the weights on value (See Equation 3.1). The query, key, and value are calculated by multiplying the input vector by three matrices W_Q, W_K, W_V, which can be denoted as $Q = W_Q X$, $K = W_K X$, $V = W_V X$, where X is the input vector. Dot-product attention function was first proposed by Luong et al. [59] for machine translation. It applied a softmax function on the dot product of the current hidden state h_t and all source

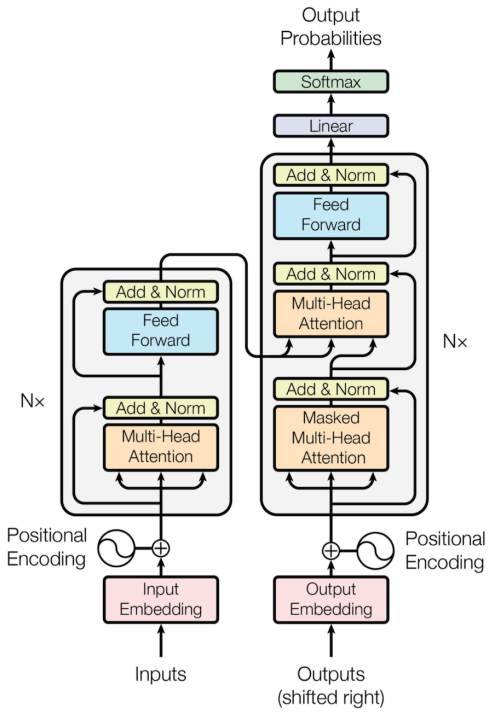


Fig. 3.1: The Transformer - model architecture [5]

states $\hat{h}_s$. The dot product attention is incorporated as the fundamental attention function in the Transformer due to the swiftness and space-efficiency in practice. The scaling factor of $\frac{1}{\sqrt{d_k}}$ can counterbalance the effect of a small gradient for softmax function caused by a large dot product:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V. \tag{3.1}$$

Multi-Head Attention. Multi-head attention multiplies the query, key and value gained from input vectors by different and learned matrices to generate multiple sets of query, key, and value. Each set of query, key, and value is fed into the scaled dot-product attention function and returns output values, namely head. Then all heads are concatenated as a matrix and projected to induce the final values. Multi-head attention expands the model's ability to attend information at different positions without increasing the total

computational cost compared against single-head attention.

$$\begin{aligned} MultiHead(Q,K,V) &= Concat(head_1,...,head_n)W^O \\ head_i &= Attention(QW_i^Q, KW_i^K, VW_i^V), \end{aligned} \tag{3.2}$$

where W^O, W_i^Q, W_i^K, W_i^V are parameter matrices for projection.

3.2.2 Position-wise Feed-Forward Networks

After a self-attention layer, a fully connected feed-forward network (FFN) is applied to each position of input sequences independently. The output from self-attention is fed to a linear function followed by a RELU function, then projected with another linear function.

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{3.3}$$

3.2.3 Residual Connection and Layer normalization

The output from each sub-layer in the encoder and decoder layers is modified by residual connection and layer normalization. Residual connection or residual learning [60] is proposed to resolve increasing training errors and degradation of model performance in deep convolution neural networks. Residual learning adds the input to the output of a neural network layer, as denoted by $y = f(x)+x$. Layer normalization [61] normalizes the input values in a neural network layer using the mean and standard deviation of all input values in the layer. This normalization accelerates the training time of neural networks, especially of RNNs. By combining the two processing techniques, the output of each sublayer in the Transformer can be represented as $LayerNorm(x + Sublayer(x))$

3.2.4 Positional Encoding

In contrast to RNNs or CNNs that inherently take into account the order of tokens in sequences, the Transformer structure doesn't store the position information. To address this issue, a position encoding is added to the input embedding at the entry of encoder and decoder stacks. The position encoding in the Transformer (shown in Equation 3.4) adopts

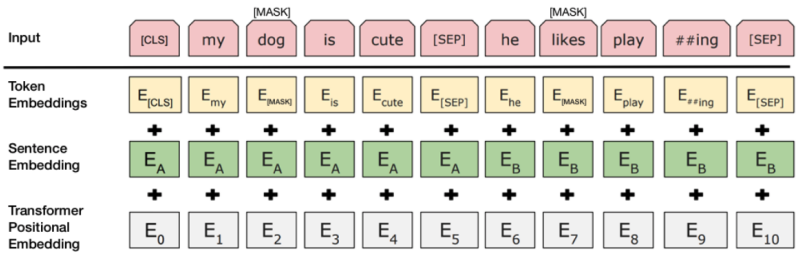


Fig. 3.2: BERT input representation [6]

sinusoidal functions to encode position information in sequences as it can attend relative positions effortlessly:

$$PE_{pos,2i} = sin(pos/10000^{2i/d_model})$$
$$PE_{pos,2i+1} = cos(pos/10000^{2i/d_model}). \tag{3.4}$$

3.3 BERT

BERT are state-of-the-art models for pre-trained language representations. The existing approaches for language representations such as ELMo, OpenAI GPT [62,63] are limited by the unidirectional architectures. BERT, on the other hand, considers bidirectional representations by performing a masked language model task and a next sentence prediction task. By minimizing the combined objective function of two strategies, BERT is able to pre-train language models based on the original transformer encoder architecture [5]. The input to BERT is shown in Figure 3.2.

3.3.1 Masked LM (MLM) Task

Masked LM task attends the left and right contexts of input sequences by randomly masking certain percentage of tokens in input sequences and predicting the masked tokens based on unmasked ones. Before transforming tokens in log sequences into embedding vectors, 15% of tokens are selected and replaced with a [MASK] token. In practice, the [MASK] token doesn't appear in the fine-tuning process. To mitigate this, when a token is chosen to be masked, 80% of the time the token is replaced with [MASK] token, 10% of the time it is replaced with a random token from the corpus, and 10% of the time it remains the same. The BERT objective function only takes into account the prediction of masked

tokens.

3.3.2 Next Sentence Prediction (NSP) Task

Next sentence prediction is designed for downstream tasks that require the understanding of sentence relationships such as Question Answering (QA) and Natural Language Inference (NLI). In pre-training, next sentence prediction generates sentence pairs consisting of two sentences, A and B from the corpus. Given the preceding sentence A, 50% of the time B is the subsequent sentence after A, while 50% of the time B is a random sentence from the corpus. In order to discriminate two sentences, a special token [SEP] is injected to the end of each sentence. In addition, a [CLS] token is inserted at the beginning of an input sample. The [CLS] representation can be fed to an output layer for classification. After incorporating these special tokens in input sequence, a segment embedding is added to a token embedding and a position embedding. The segment embedding implies to which sentence each token belongs. The NSP task uses IsNext or NotNext label to determine if there is a connection between a sentence pair. Therefore, the NSP loss function is treated as a binary classification loss.

CHAPTER 4

LogBERT

LogBERT is a deep learning neural network for log sequence anomaly detection based on BERT. The primary purpose of LogBERT is to learn the contextual information in log sequences. To do so, LogBERT trains two self-supervised tasks, namely Masked Log Key Prediction (MLKP) and Volume of Hypersphere Minimization (VHM), to analyze log sequences bidirectionally. MLKP task randomly replaces a fixed percentage of log keys in a sequence with [MASK] token, then predicts the masked log keys using a probability distribution. VHM task inserts a [DIST] token at the beginning of a log sequence and utilizes the training result of the [DIST] token as the representation of the sequence in latent space. The objective of VHM task is to minimize the volume of a hypersphere that encompasses the representation of normal log sequences. Intuitively, normal instances densely gather around the center of the hypersphere while abnormal instances are further away from the center. The main structure of LogBERT is a transformer encoder that solely depends on attention mechanisms. The input representation for the transformer encoder is the sum of log key embedding vectors and position embedding vectors of log sequences. The output of the transformer encoder is then fed to a fully connected neural network layer and a softmax layer to generate a probability distribution for each log key in the log key set as the prediction of a masked log key. This chapter goes over the details of LogBERT and its implementation.

4.1 Architecture

Given a sequence of unstructured log messages, we aim to detect whether this sequence is normal or anomalous. In order to represent log messages, following a widely used pre-processing approach, we first extract log keys from log messages via a log parser. Then, we can define a log sequence as a sequence of ordered log keys $S = \{k_1, ..., k_t, ..., k_T\}$,

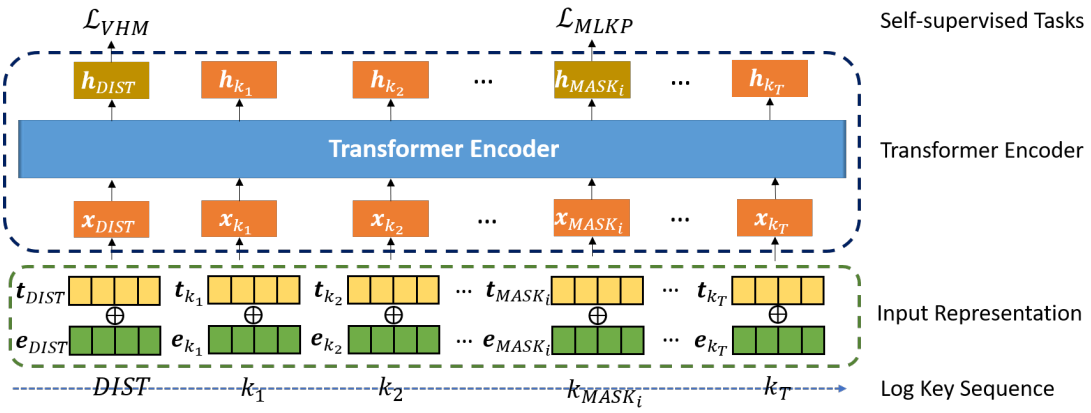


Fig. 4.1: The overview of LogBERT

where $k_t \in \mathcal{K}$ indicates the log key in the t-th position, and $\mathcal{K}$ indicates a set of log keys extracted from log messages. The goal of this task is to predict whether a new log sequence S is anomalous based on a training dataset $\mathcal{D} = \{S^j\}_{j=1}^{N}$ that consists of only normal log sequences. To achieve that, we propose an architecture, called LogBERT, to model the normal sequences and further derive an anomaly detection criterion to identify anomalous sequences. Figure 4.1 shows the whole framework of LogBERT.

4.2 Input Representation

Given a normal log sequence S^j, we first add a special token, [DIST], at the beginning of S^j as the first log key. LogBERT then represents each log key k_t^j as an input representation $\mathbf{x}_t^j$, where the representation $\mathbf{x}_t^j$ is a summation of a log key embedding and a position embedding. In this work, we randomly generate a matrix $\mathbf{E} \in \mathbb{R}^{|\mathcal{K}|*d}$ as the log key embedding matrix, where d is the dimension of log key embedding, while the position embeddings $\mathbf{T} \in \mathbb{R}^{T*d}$ are generated by using the same sinusoid function to encode the position information of log keys in a sequence [6]. Finally, the input representation of the log key k_t is defined as: $\mathbf{x}_t^j = \mathbf{e}_{k_t} + \mathbf{t}_{k_t}$.

4.3 Transformer Encoder

LogBERT adopts Transformer encoder to learn the contextual relations among log

keys in a sequence. Transformer encoder consists of multiple transformer layers. Each transformer layer includes a multi-head self-attention and a position-wise feed forward sub-layer in which a residual connection is employed around each of two sub-layers, followed by layer normalization [5]. The multi-head attention employs H parallel self-attentions to jointly capture different aspect information at different positions over the input log sequence. Formally, for the l-th head of the attention layer, the scaled dot-product self-attention is defined as:

$$head_l = Attention(\mathbf{X}^j\mathbf{W}_l^Q, \mathbf{X}^j\mathbf{W}_l^K, \mathbf{X}^j\mathbf{W}_l^V), \tag{4.1}$$

where $Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_v}})\mathbf{V}$; $\mathbf{X}^j \in \mathbb{R}^{T*d}$ is the input representation of the log sequence; $\mathbf{W}_l^Q$, $\mathbf{W}_l^K$ and $\mathbf{W}_l^V$ are linear projection weights with dimensions $\mathbb{R}^{d*d_v}$ for the l-th head. Each self-attention makes each key attend to all the log keys in an input sequence and computes the hidden representation for each log key with an attention distribution over the sequence.

The multi-head attention employs a parallel of self-attentions to jointly capture different aspect information at different log keys. Formally, the multi-head attention concatenates H parallel heads together as:

$$f(\mathbf{X}^j) = Concat(head_1, ..., head_H)\mathbf{W}^O, \tag{4.2}$$

where $\mathbf{W}^O \in \mathbb{R}^{hd_v*d_o}$ is a projection matrix.

Then, the position-wise feed forward sub-layer with a ReLU activation is applied to the hidden representation of each activity separately. Finally, by combining the position-wise feed forward sub-layer and multi-head attention, a transformer layer is defined as:

$$\text{transformer\_layer}(\mathbf{X}^j) = FFN(f(\mathbf{X})) = ReLU(f(\mathbf{X})\mathbf{W}_1)\mathbf{W}_2, \tag{4.3}$$

where $\mathbf{W}_1$ and $\mathbf{W}_2$ are trained projection matrices.

The Transformer encoder usually consists of multiple transformer layers. We denote $\mathbf{h}_t^j$ as the contextualized embedding vector of the log key k_t^j produced by the Transformer

encoder, i.e., $\mathbf{h}_t^j = \text{Transformer}(k_t^j)$.

4.4 Objective Function

In order to train the LogBERT model, we propose two self-supervised training tasks to capture the patterns of normal log sequences.

4.4.1 Task I: Masked Log Key Prediction (MLKP)

In order to capture the bidirectional information of log sequences, we train LogBERT to predict the masked log keys in log sequences. In our scenario, LogBERT takes log sequences with random masks as inputs, where we randomly replace a ratio of log keys in a sequence with a specific [MASK] token. The training objective is to accurately predict the randomly masked log keys. The purpose is to make LogBERT encode the prior knowledge of normal log sequences.

To achieve that, we feed the contextualized embedding vector of the i-th [MASK] token in the j-th log sequence $\mathbf{h}^j_{[\text{MASK}_i]}$ to a softmax function, which will output a probability distribution over the entire set of log keys $\mathcal{K}$:

$$\hat{y}^j_{[\text{MASK}_i]} = Softmax(\mathbf{W}_C \mathbf{h}^j_{[\text{MASK}_i]} + \mathbf{b}_C), \tag{4.4}$$

where $\mathbf{W}_C$ and $\mathbf{b}_C$ are trainable parameters. Then, we adopt the cross entropy loss as the objective function for masked log key prediction, which is defined as:

$$\mathcal{L}_{MLKP} = -\frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{M} y^j_{[\text{MASK}_i]} \log \hat{y}^j_{[\text{MASK}_i]}, \tag{4.5}$$

where $y^j_{[\text{MASK}_i]}$ is the real log key for the i-th masked token, N is the total number of log sequences, and M is the total number of masked tokens in the j-th log sequence. Since the patterns of normal and anomalous log sequences are different, we expect once LogBERT is able to correctly predict the masked log keys, it can distinguish the normal and anomalous log sequences.

4.4.2 Task II: Volume of Hypersphere Minimization (VHM)

Inspired by the Deep SVDD approach [64], where the objective is to minimize the volume of a data-enclosing hypersphere, we propose a spherical objective function shown in Equation 4.6. The motivation is that normal log sequences should be concentrated and close to each other in the embedding space, while the anomalous log sequences are far to the center of the sphere. We first derive the representations of normal log sequences and then compute the center representation based on the mean operation. In particular, we consider the embedding vector of the [DIST] token $\mathbf{h^j}_{\text{DIST}}$, which encodes the information of entire log sequence based on the Transformer encoder, as the representation of a log sequence in the embedding space. To make the representations of normal log sequences close to each other, we further derive the center representation of normal log sequences $\mathbf{c}$ in the training set by a mean operation, i.e., $\mathbf{c} = \text{Mean}(\mathbf{h}^j_{\text{DIST}})$. Then, the objective function is to make the representation of normal log sequence $\mathbf{h}^j_{\text{DIST}}$ close to the center representation $\mathbf{c}$:

$$\mathcal{L}_{VHM} = \frac{1}{N}\sum_{j=1}^{N} ||\mathbf{h}^j_{\text{DIST}} - \mathbf{c}||^2. \tag{4.6}$$

By minimizing the Equation 4.6, we expect all the normal log sequences in the training set are close to the center, while the anomalous log sequences have a larger distance to the center. Meanwhile, another advantage of the spherical objective function is that by making the sequence representations close to the center, the Transformer encoder can also leverage the information from other log sequences via the center representation $\mathbf{c}$ since $\mathbf{c}$ encodes all the information of normal log sequences. As a result, the model should be able to predict the masked log keys with higher accuracy for normal log sequences because the normal log sequences should share similar patterns.

Finally, the objective function for training LogBERT is defined as below:

$$\mathcal{L} = \mathcal{L}_{MLKP} + \alpha\mathcal{L}_{VHM}, \tag{4.7}$$

where α is a hyper-parameter to balance two training tasks.

4.5 Anomaly Detection

After training, we can deploy LogBERT for anomalous log sequence detection. Given a testing log sequence, we first randomly replace some log keys with [MASK] tokens and use the randomly-masked log sequence as an input to LogBERT. Since LogBERT is trained on normal log sequences, it can achieve high prediction accuracy if the testing log sequence is normal. Hence, we can derive the anomalous score of a log sequence based on the prediction results on the [MASK] tokens. In particular, given a [MASK] token, the probability distribution calculated based on Equation 4.4 indicates the likelihood of a log key appeared in the position of the [MASK] token. Similar to the strategy in DeepLog [18], we build a candidate set of normal log keys based on their likelihoods computed by $\hat{y}_{[\text{MASK}_i]}$, and treat a key as normal if it achieves the top g highest probability. In other words, if the observed log key is not in the top-g candidate set, we consider the log key as an anomalous log key. Then, when a log sequence consists of more than r anomalous log keys, we will label this log sequence as anomalous. Both g and r are hyper-parameters and will be tuned based on the validation set.

CHAPTER 5

Experiment and Analysis

In this chapter, we start by introducing three public datasets (Chapter 5.1) and six baselines (Chapter 5.2) adopted in the experiments. Then we explicates our experiment setup in Chapter 5.3 and explore experimental results in Chapter 5.4. In our experiments, we first compare the performance of proposed model on the three datasets with the performances of baseline models in Chapter 5.4.1. We further conduct ablation study based on two self-supervised training tasks in Chapter 5.4.2 . In Chapter 5.4.3, we present the visualization on representations of log sequences. Finally, we analyze the sensitivity of models with respect to different hyper-parameters in Chapter 5.4.4.

5.1 Datasets

Loghub [65] includes many public datasets collected from different resources for log analysis. The existing labeled datasets includes HDFS, Hadoop, OpenStack, BGL, and Thunderbird. We evaluate the proposed LogBERT on three datasets, HDFS, BGL, and Thunderbird. Table 5.1 shows statistics of the datasets. For all datasets, we adopt around 5000 normal log sequences for training. The number in the brackets under the column "# Log Keys" indicates the number of unique log keys in the training dataset.

Table 5.1: Statistics of evaluation datasets

Dataset	# Log Messages	# Anomalies	# Log Keys	# of Log Sequences in Test Dataset	
				Normal	Anomalous
HDFS	11,172,157	284,818	46 (15)	553,366	10,647
BGL	4,747,963	348,460	334 (175)	10,045	2,630
Thunderbird-mini	20,000,000	758,562	1,165 (866)	71,155	45,385

5.1.1 HDFS

Hadoop Distributed File System (HDFS) [66]. HDFS dataset is generated by running

Hadoop-based map-reduce jobs on Amazon EC2 nodes and manually labeled through hand-crafted rules to identify anomalies. HDFS dataset consists of 11,172,157 log messages, of which 284,818 are anomalous. For HDFS, we group log keys into log sequences based on the block_id in each log message. The average length of log sequences is 19.

5.1.2 BGL

BlueGene/L Supercomputer System (BGL) [67]. BGL dataset is collected from a BlueGene/L supercomputer system at Lawrence Livermore National Labs (LLNL). Logs contain alert and non-alert messages identified by alert category tags. The alert messages are considered as anomalous. BGL dataset consists of 4,747,963 log messages, of which 348,460 are anomalous. For BGL, we define a time sliding window as 5 minutes to generate log sequences, where the average length is 562.

5.1.3 Thunderbird

Thunderbird [67] dataset is another large log dataset collected from a supercomputer system. We select the first 20,000,000 log messages from the original Thunderbird dataset to compose our dataset, of which 758,562 are anomalous. For Thunderbird, we also adopt a time sliding window as 1 minute to generate log sequences, where the average length is 326.

5.2 Baselines

We compare our LogBERT model with the following baselines that are widely used for log anomaly detection.

PCA. PCA builds counting matrix based on the frequency of log keys sequences and then reduces the original counting matrix into a low dimensional space to detect anomalous sequences.

One-Class SVM (OCSVM). One-Class SVM is a well-known one-class classification model and widely used for log anomaly detection by only observing the normal data.

Isolation Forest (iForest). Isolation forest is an unsupervised learning algorithm for anomaly detection by representing features as tree structures.

LogCluster. LogCluster is a clustering based approach, where the anomalous log sequences are detected by comparing with the existing clusters.

DeepLog. DeepLog is a state-of-the-art log anomaly detection approach. DeepLog adopts recurrent neural network to capture patterns of normal log sequences and further identifies the anomalous log sequences based on the performance of log key predictions.

LogAnomaly. LogAnomaly is a deep learning-based anomaly detection approach and able to detect sequential and quantitative log anomalies

5.3 Experimental Setup

Regarding baselines, we leverage the package *Loglizer* [3] to evaluate PCA, OCSVM, iForest as well as LogCluster for anomaly detection and adopt the open source deep learning-based log analysis toolkit to evaluate DeepLog and LogAnomaly [1]. For LogBERT, we construct a Transformer encoder by using two Transformer layers. The dimensions for the input representation and hidden vectors are 50 and 256, respectively. The hyper-parameters in our model, including m the ratio of masked log keys for the MKLP task, r the number of predicted anomalous log keys, and g the size of top-g candidate set for anomaly detection, are tuned based on a small validation set. In our experiments, both training and detection phases have the same ratio of masked log keys m. The code of our implementation are available online [2].

5.4 Experimental Results

5.4.1 Log Anomaly Detection

Table 5.2 shows the results of LogBERT as well as baselines on three datasets. We can notice that PCA, Isolation Forest, and OCSVM have poor performance on log anomaly

[1] https://github.com/donglee-afar/logdeep
[2] https://github.com/HelenGuohx/logbert

detection. Although these methods could achieve extremely high precision or recall values, they cannot balance the log anomaly detection on both precision and recall, which lead to extremely low F1 scores. This could be because using the counting vector to represent a log sequence leads to the loss of temporal information from log sequences. LogCluster, which is designed for log anomaly detection, achieves better performance than the PCA, Isolation Forest, and OCSVM. Meanwhile, two deep learning-based baselines, DeepLog and LogAnomaly, significantly outperform the traditional approaches and achieve reasonable F1 scores on three datasets, which show the advantage to adopt deep learning models to capture the patterns of log sequences. Moreover, our proposed LogBERT achieves the highest F1 scores on three datasets with large margins by comparing with all baselines. It indicates that by using self-supervised training tasks, LogBERT can model the normal log sequences and further identify anomalous sequences with high accuracy.

Table 5.2: Experimental results on HDFS, BGL, and Thunderbird datasets

Method	HDFS			BGL			Thunderbird		
	Precision	Recall	F-1 score	Precision	Recall	F-1 score	Precision	Recall	F-1 score
PCA	5.89	100.00	11.12	9.07	98.23	16.61	37.35	100.00	54.39
iForest	53.60	69.41	60.49	99.70	18.11	30.65	34.45	1.68	3.20
OCSVM	2.54	100.00	4.95	1.06	12.24	1.96	18.89	39.11	25.48
LogCluster	99.26	37.08	53.99	95.46	64.01	76.63	98.28	42.78	59.61
DeepLog	88.44	69.49	77.34	89.74	82.78	86.12	87.34	99.61	93.08
LogAnomaly	94.15	40.47	56.19	73.12	76.09	74.08	86.72	99.63	92.73
LogBERT	87.02	78.10	**82.32**	89.40	92.32	**90.83**	96.75	96.52	**96.64**

5.4.2 Ablation Study

In order to further understand our proposed LogBERT, we conduct ablation experiments on three log datasets. LogBERT is trained by two self-supervised tasks. We evaluate the performance of LogBERT by only using one training task each time. When the model is only trained by minimizing the volume of hypersphere, we identify anomalous log sequences by computing distances of the log sequence representations to the center of normal log sequences c. If the distance is larger than a threshold, we consider a log sequence is anomalous. Table 5.3 shows the experimental results. We can notice that when only using

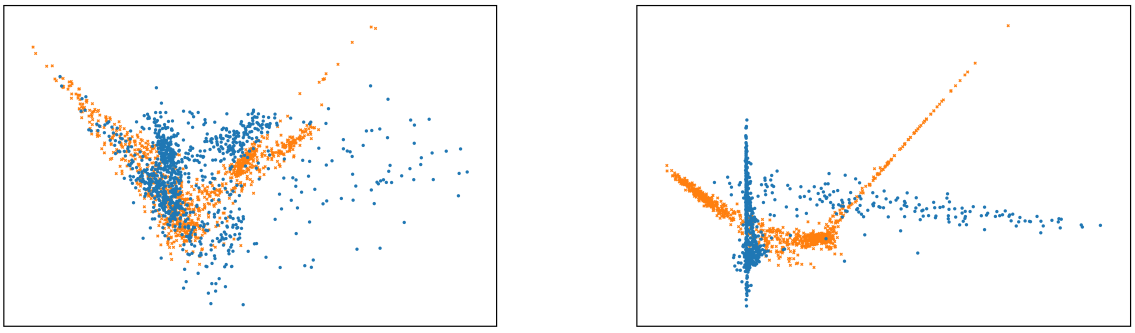
the task of masked log key prediction to train the model, we can still get very good performance on log anomaly detection, which shows the effectiveness of training the model by predicting masked log keys. We can also notice that even we do not train the LogBERT with the task of the volume of hypersphere minimization, LogBERT achieves higher F1 scores than DeepLog on all three datasets, which shows that compared with LSTM, Transformer encoder are better at capturing the patterns of log sequences. Meanwhile, we can observe that when only training the model for minimizing the volume of hypersphere, the performance is poor. It indicates that only using distance as a measure to identify anomalous log sequences cannot achieve good performance. However, combining two self-supervised tasks to train LogBERT can achieve better performance than the models only trained by one task. Especially, for the HDFS dataset, LogBERT gains a large margin in terms of F1 score (82.32) compared with the model only trained by MLKP (78.09). For BGL and Thunderbird, the improvement of LogBERT is not as significant as the model in HDFS. This could be because the average length of log sequences in BGL (562) and Thunderbird (326) datasets are much larger than the log sequences in HDFS (19). For longer sequences, only predicting the masked log keys can capture the most important patterns of log sequences since there are many more mask tokens in longer sequences. On the other hand, for short log sequences, we cannot have many masks tokens. As a result, the task of the volume of hypersphere minimization can help to boost the performance. Hence, based on Table 5.3, we can conclude that using two self-supervised tasks to train LogBERT always leads to better performance, especially when the log sequences are relatively short.

Table 5.3: Performance of LogBERT based on one self-supervised training task

	HDFS			BGL			Thunderbird		
	Precision	Recall	F-1 score	Precision	Recall	F-1 score	Precision	Recall	F-1 score
MLKP	77.54	78.65	78.09	93.16	86.46	89.69	97.07	95.90	96.48
VHM	2.43	39.17	4.58	71.04	43.84	54.22	56.58	43.87	49.42
Both	87.02	78.10	82.32	89.40	92.32	90.83	96.75	96.52	96.64

5.4.3 Visualization

In order to visualize the log sequences, we adopt locally linear embedding (LLE) algorithm [68] to map the log sequence representations into a two dimensional space, where the hidden vector of [DIST] token $\mathbf{h}_{\text{DIST}}$ is used as the representation of a log sequence. We randomly select 1000 normal and 1000 anomalous sequences from the BGL dataset for visualization. Figure 5.1 shows the visualization results of log sequences trained by LogBERT with and without the VHM task. We can notice that the normal log sequences group together, while the anomalous log sequences spread out in the whole latent space. However, without the VHM task (shown in Figure 5.1a), there are some overlaps between the normal and anomalous log sequence. Meanwhile, as shown in Figure 5.1b, by incorporating the VHM task, the normal and anomalous log sequences are clearly separated in the latent space. Therefore, the visualization presents that the VHM task is effective in regulating the model to split the normal and abnormal data in latent space.



(a) Trained without the VHM task

(b) Trained by MLKP and VHM tasks

Fig. 5.1: Visualization of log sequences by using the representations of [DIST] tokens $\mathbf{h}_{\text{DIST}}$. The blue dots indicate the normal log sequences, while the orange 'x' symbols indicate anomalous log sequences.

5.4.4 Parameter Analysis

We also investigate the sensitivity of model performance by tuning various hyper-parameters using BGL dataset.

Performance against Different α Value

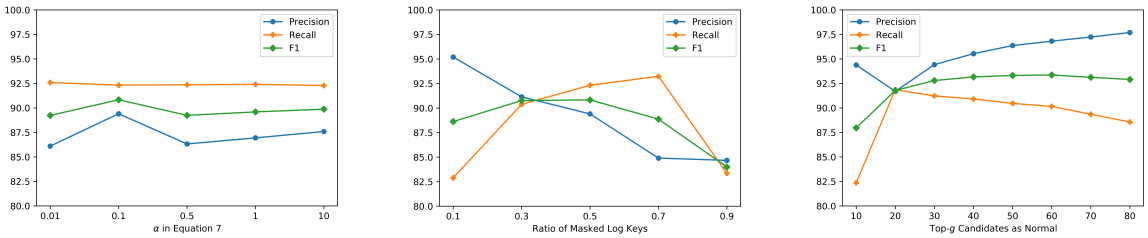
Figure 5.2a shows that the model performance is relatively stable by setting different α values in the objective function. This is because, for the BGL dataset, the loss from the masked log key prediction dominates the final loss value due to the log sequences. As a result, the weight for the VHM task does not have much influence on the performance.

Performance against Increasing Ratios of Masked Log Keys

Figure 5.2b shows that increasing the ratios of masked log keys in the sequences from 0.1 to 0.5 can slightly increase the F1 scores while keeping increasing the ratios makes the performance worse. This is because while the masked log keys crease in a reasonable range, the model can capture more information about the sequence. However, if a sequence contains too many masked log keys, it loses too much information for making the predictions.

Performance against Increasing Size of Candidates

Figure 5.2c shows that when increasing the size of the candidate set as normal log keys, the precision for anomaly detection keeps increasing while the recall is reducing, which meet our expectation. Hence, we need to find the appropriate size of the candidate set to balance the precision and recall for the anomaly detection.



(a) Different α values (b) Different ratios of masks (c) Different g-candidates

Fig. 5.2: Parameter analysis on the BGL dataset.

CHAPTER 6

Conclusion

Log anomaly detection is essential to protect online computer systems from malicious attacks or malfunctions. In this thesis, we develop a novel deep learning model based on BERT, LogBERT, to detect abnormal log sequences. We also evaluate the performance of LogBERT on three datasets, HDFS, BGL, and Thunderbird in terms of precision, recall, and F1 score by comparing with the existing state-of-the-art approaches. Our findings are four-fold. Firstly, deep learning approaches such as DeepLog and LogAnomaly outperform traditional machine learning approaches such as PCA, Isolation Forest, OCSVM, and LogCluster. Among all models, LogBERT achieves the best performance on three public datasets. Secondly, our ablation experiment on two self-supervised tasks (MLKP and VHM) demonstrates that using two training tasks in LogBERT is better than using solely one task. Especially, VHM task can help boost the detection ability when log sequences are short. Thirdly, we visualize normal and abnormal log sequences by mapping the representations of DIST into a two-dimensional space. After comparing the visualization results of log sequences trained by LogBERT with and without VHM task, we can observe that VHM task is effective when separating normal and abnormal log sequences. Last but not least, we conduct parameter analysis on BGL dataset by tuning different hyper-parameters. We find out that α has a trivial impact on the performance of detection. Increasing ratios of masked log keys can greatly improve detection accuracy but keep increasing the ratios can hurt the model's performance. The number of predicted candidates(Top-g candidates) has a positive effect on the detection ability of LogBERT but the effectiveness mitigates if the number of g candidates is too large.

In the future, we plan to further improve LogBERT in the following directions. In the log prepossessing stage, log keys are represented by one-hot encoding. It assumes that all log keys are independent mutually. It is worthwhile to discuss the similarity between different

log keys. To do so, some researchers [22, 25] transform log events into semantic vectors via pre-trained word embedding. Moreover, LogBERT only takes log keys as input features of log data, which are the constant part extracted from log messages. We may also take into account the extra information in the variable part of log messages such as timestamp. Last but not least, LogBERT is evaluated on public datasets for research purposes. It would be interesting to deploy our model on software such as OpenStack to test its real-time detection ability.

REFERENCES

[1] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

[2] S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso, and O. Kao, "Self-supervised log parsing," 2020.

[3] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: System log analysis for anomaly detection," in *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, 2016, pp. 207–218.

[4] L. Weng, "From Autoencoder to Beta-VAE," Aug. 2018. [Online]. Available: https://lilianweng.github.io/2018/08/12/from-autoencoder-to-beta-vae.html

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

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

[8] V. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artificial intelligence review*, vol. 22, no. 2, pp. 85–126, 2004.

[9] P. De Boer and M. Pels, "Host-based intrusion detection systems," *Amsterdam University*, 2005.

[10] Z. Zojaji, R. E. Atani, A. H. Monadjemi *et al.*, "A survey of credit card fraud detection techniques: Data and technique oriented perspective," *arXiv preprint arXiv:1611.06439*, 2016.

[11] C. Cao, F. Liu, H. Tan, D. Song, W. Shu, W. Li, Y. Zhou, X. Bo, and Z. Xie, "Deep learning and its applications in biomedicine," *Genomics, proteomics & bioinformatics*, vol. 16, no. 1, pp. 17–32, 2018.

[12] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep learning for iot big data and streaming analytics: A survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 2923–2960, 2018.

[13] J. Wang, Y. Tang, S. He, C. Zhao, P. K. Sharma, O. Alfarraj, and A. Tolba, "Logevent2vec: Logevent-to-vector based anomaly detection for large-scale logs in internet of things," *Sensors*, vol. 20, no. 9, p. 2451, 2020.

[14] "Global Anomaly Detection Industry." [Online]. Available: https://www.reportlinker.com/p05797895/Global-Anomaly-Detection-Industry.html?utmsource=GNW

[15] G. Aarish, "Anomaly Detection for Application Log Data," Master's Project, San Jose State University, San Jose, CA, May 2018. [Online]. Available: https://scholarworks.sjsu.edu/etd_projects/635/

[16] R. B. Yadav, P. S. Kumar, and S. V. Dhavale, "A survey on log anomaly detection using deep learning," in *2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*. IEEE, 2020, pp. 1215–1220.

[17] S. Bulusu, B. Kailkhura, B. Li, P. K. Varshney, and D. Song, "Anomalous instance detection in deep learning: A survey," *arXiv preprint arXiv:2003.06979*, 2020.

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

[19] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Largescale system problem detection by mining console logs," *Proceedings of SOSP'09*, 2009.

[20] K.-L. Li, H.-K. Huang, S.-F. Tian, and W. Xu, "Improving one-class svm for anomaly detection," in *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No. 03EX693)*, vol. 5. IEEE, 2003, pp. 3077–3081.

[21] Y. Wang, J. Wong, and A. Miner, "Anomaly intrusion detection using one class svm," in *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004.* IEEE, 2004, pp. 358–364.

[22] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun *et al.*, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs." in *IJCAI*, vol. 7, 2019, pp. 4739–4745.

[23] Z. Wang, Z. Chen, J. Ni, H. Liu, H. Chen, and J. Tang, "Multi-scale one-class recurrent neural networks for discrete event sequence anomaly detection," 2020.

[24] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "Towards automated log parsing for large-scale log data analysis," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 6, pp. 931–944, 2018.

[25] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li *et al.*, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 807–817.

[26] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2019, pp. 121–130.

[27] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, 2009, pp. 117–132.

[28] M. Nagappan, K. Wu, and M. A. Vouk, "Efficiently extracting operational profiles from execution logs using suffix arrays," in *2009 20th International Symposium on Software Reliability Engineering*. IEEE, 2009, pp. 41–50.

[29] Q. Fu, J.-G. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *2009 ninth IEEE international conference on data mining*. IEEE, 2009, pp. 149–158.

[30] L. Tang, T. Li, and C.-S. Perng, "Logsig: Generating system events from raw textual logs," in *Proceedings of the 20th ACM international conference on Information and knowledge management*, 2011, pp. 785–794.

[31] M. Mizutani, "Incremental mining of system log format," in *2013 IEEE International Conference on Services Computing*. IEEE, 2013, pp. 595–602.

[32] K. Shima, "Length matters: Clustering system log messages using length of words," *arXiv preprint arXiv:1611.03213*, 2016.

[33] H. Hamooni, B. Debnath, J. Xu, H. Zhang, G. Jiang, and A. Mueen, "Logmine: Fast pattern recognition for log analytics," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, 2016, pp. 1573–1582.

[34] M. Nagappan and M. A. Vouk, "Abstracting log lines to log event types for mining software system logs," in *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. IEEE, 2010, pp. 114–117.

[35] A. Nandi, A. Mandal, S. Atreja, G. B. Dasgupta, and S. Bhattacharya, "Anomaly detection using program control flow graph mining from execution logs," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 215–224.

[36] S. Messaoudi, A. Panichella, D. Bianculli, L. Briand, and R. Sasnauskas, "A search-based approach for accurate identification of log message formats," in *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*. IEEE, 2018, pp. 167–16 710.

[37] M. Du and F. Li, "Spell: Streaming parsing of system event logs," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 2016, pp. 859–864.

[38] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 2017, pp. 33–40.

[39] Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora, "An automated approach for abstracting execution logs to execution events," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 20, no. 4, pp. 249–267, 2008.

[40] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, 2009, pp. 117–132.

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

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

[43] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection." in *USENIX Annual Technical Conference*, 2010, pp. 1–14.

[44] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 2016, pp. 102–111.

[45] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural computation*, vol. 13, no. 7, pp. 1443–1471, 2001.

[46] K. Heller, K. Svore, A. D. Keromytis, and S. Stolfo, "One class support vector machines for detecting anomalous windows registry accesses," 2003.

[47] K. A. Nguyen, S. S. i. Walde, and N. T. Vu, "Integrating distributional lexical contrast into word embeddings for antonym-synonym distinction," *arXiv preprint arXiv:1605.07766*, 2016.

[48] T. Amarbayasgalan, B. Jargalsaikhan, and K. H. Ryu, "Unsupervised novelty detection using deep autoencoders with density based clustering," *Applied Sciences*, vol. 8, no. 9, p. 1468, 2018.

[49] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[50] T. Nolle, S. Luettgen, A. Seeliger, and M. Mühlhäuser, "Analyzing business process anomalies using autoencoders," *Machine Learning*, vol. 107, no. 11, pp. 1875–1893, 2018.

[51] S. Lu, X. Wei, Y. Li, and L. Wang, "Detecting anomaly in big data system logs using convolutional neural network," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. IEEE, 2018, pp. 151–158.

[52] R. Ren, J. Cheng, Y. Yin, J. Zhan, L. Wang, J. Li, and C. Luo, "Deep convolutional neural networks for log event classification on distributed cluster systems," 12 2018, pp. 1639–1646.

[53] D. Kwon, K. Natarajan, S. C. Suh, H. Kim, and J. Kim, "An empirical study on network anomaly detection using convolutional neural networks," in *2018 IEEE 38th*

International Conference on Distributed Computing Systems (ICDCS), 2018, pp. 1595–1598.

[54] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[55] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014.

[56] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *arXiv preprint arXiv:1409.3215*, 2014.

[57] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[58] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1243–1252.

[59] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[60] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[61] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[62] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," 2018.

[63] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[64] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft, "Deep one-class classification," in *International conference on machine learning*. PMLR, 2018, pp. 4393–4402.

[65] S. He, J. Zhu, P. He, and M. R. Lyu, "Loghub: A large collection of system log datasets towards automated log analytics," 2020.

[66] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Online system problem detection by mining patterns of console logs," in *2009 Ninth IEEE International Conference on Data Mining*. IEEE, 2009, pp. 588–597.

[67] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*. IEEE, 2007, pp. 575–584.

[68] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *science*, vol. 290, no. 5500, pp. 2323–2326, 2000.