DEEP LEARNING BASED LOG ANOMALY DETECTION WITH TIME DIFFERENCES

BARANSEL SAĞINDA

SEPTEMBER 2020

DEEP LEARNING BASED LOG ANOMALY DETECTION WITH TIME DIFFERENCES

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF NATURAL

AND APPLIED SCIENCES

OF ÇANKAYA UNIVERSITY

BY

BARANSEL SAĞINDA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF

MASTER OF SCIENCE

IN

COMPUTER ENGINEERING

AUGUST 2020

STATEMENT OF NON-PLAGIARISM

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: Baransel SAĞINDA

Signature: 

ABSTRACT

LOG ANOMALY DETECTION USING

DEEP LEARNING TECHNIQUES

SAĞINDA, Baransel

M.Sc., Computer Engineering Department

Supervisor: Assistant Prof. Dr. Roya CHOUPANI

Co-Supervisor: Prof. Dr. Erdoğan DOĞDU

2020, 51 pages

With the ever-growing digital transformation in our lives and the new computing systems with the adaption of microservices, systems generated log records are increasing tremendously. Monitoring and evaluation of these “big” log records are real challenges due to the size and growing pace of system log generation. Most of the time, these records are not utilized efficiently for the benefit of increased system availability and reliability due to the lack of resources to process these records timely and efficiently. In this work, we propose a method for parsing and evaluating system logs based on the length of time between the occurrence events in logs and the utilization of these time periods in learning-based anomaly detection. We specifically use Seq2seq networks for anomaly detection. Results show that our method is successful at distinguishing between normal and anomaly events, even without any information about log keys.

Keywords: Log records analysis, anomaly detection, deep learning

ÖZ

ZAMAN FARKLARI İLE DERİN ÖĞRENME

TABANLI LOG ANORMALLİĞİ ALGILAMA

SAĞINDA, Baransel

M.Sc., Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Danışman: Dr. Öğr. Üyesi Roya CHOUPANI

Ortak Danışman: Prof. Dr. Erdoğan DOĞDU

2020, 51 pages

Sürekli büyüyen dijital hizmetler ve yeni mikro hizmetlerin adaptasyonu ile birlikte yeni bilgi işlem sistemleri ile oluşturulan kayıtlarin miktarı muazzam bir şekilde artmaktadır. Bu büyük kayıtların izlenmesi ve değerlendirilmesi, sistem günlüğü oluşturmanın boyutu ve artan hızı nedeniyle giderek zorlakmaştadır. Çoğu zaman, bu kayıtları zamanında ve verimli bir şekilde işlemek için kaynaklar yetmemektedir. Bu çalışmada, sistem günlüklerinin ayrıştırılması ve değerlendirilmesi için, günlüklerdeki meydana gelen olaylar arasındaki sürenin uzunluğuna dayalı anormallik tespitinde kullanımına bir yöntem öneriyoruz. Anormallik tespiti için özellikle Seq2seq nörön ağlarını kullanıyoruz. Sonuçlar, yöntemimizin olay kayıtlarının içeriği hakkında herhangi bir bilgi sahibi olmaksızın normal ve anormal olayları ayırt etmede başarılı olduğunu göstermektedir.

Anahtar Kelimeler: Log analizi, hata tespiti, derin öğrenme

ACKNOWLEDGMENTS

I would like to thank my advisors, Assistant Prof Roya CHOUPANI and Prof. Dr. Erdoğan DOĞDU for their support and motivation.

It would like to express my gratitude to my family for their continued support.

Lastly I would like to thank Assistant Prof Abdül Kadir GÖRÜR who was there for me both at the beginning and the end of this study.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

GRU : Gated Recurrent Unit

NLP : Natural Language Processing

Bi-LSTM : Long Short Term Memory

DNS : Domain Name System

ML : Machine Learning

ANN : Artificial Neural Network

GPU : Graphic Processing Unit

NMT : Neural Machine Translation

IT : Information Technology

CHAPTER 1

INTRODUCTION

Current information systems are more complex than ever. Modern large-scale information processing systems like Hadoop[1] and Spark[2] run over sometimes thousands of commodity servers. With the adoption of microservice architectures, even small systems consist of less coupled applications than traditional system designs. These advances make modern systems more resilient to hardware failures. Such systems carry a load of online services (e-commerce systems, social networks, archives, and online tools) on a 24/7 basis. Depending on the system type, downtime in any of these services can result in loss of capital or even worse consequences. For example, Heathrow airport baggage system had a computer failure in March of 2008. This event affected 140,000 people and cost $32 million to the airport[3]. Proper anomaly detection systems are necessary for building stable and secure computer systems.

System logs contain information about significant events that happened in the system to help debug problems, solve performance issues, find out security breaches, and system failures. There are mainly three types of logs. “Error logs” include information about severe problems that happen in the systems. “Warning logs” include information about the possibility of abnormal situations and warnings for future failures. “Notice logs”

[1] https://hadoop.apache.org/
[2] https://spark.apache.org/
[3] https://www.cnbc.com/id/23892979

include information about healthy and normal operations during the execution of systems.

A study performed on two close-sourced and two open-source systems has shown that there is one line of logging code against every 58 lines of programming code in software [1]. Modern software systems produce a significant amount of log records, and inspecting these log records manually by human observance is very time-consuming and nearly impossible. Debugging large systems with human labor is challenging and time-consuming [2]. In a real-world banking system, 200 full-time operators are dedicated to log monitoring with 67 screens for 190 subsystems [3]. Even with such a workforce, there is always the possibility of human mistakes, in terms of missing the warnings or errors. When the system produces an error or a warning message, it is easy to detect the failure by reading the logs. This only happens after the problem occurs. Depending on the high availability configuration of the system, a failure on a single subsystem might stop the whole system. This is referred to as a single point of failure. It is often much more rewarding to detect the problems before they happen. Also, some issues and problems do not produce proper logs. Using human labor for analyzing logs poses many other challenges.

Modern systems such as Docker[4] and Kubernetes[5] utilize multiple different subsystems. Different containers take care of different tasks like pod scheduling, network switching, and disk operations, and a fault in any given subsystem can affect the whole system. All of these subsystems produce their own logs. To detect anomalies in these systems' logs,

[4] https://www.docker.com/
[5] https://kubernetes.io/

experts from multiple disciplines need to work together to investigate logs. This makes anomaly detection with human labor hard for such cases.

These problems show that there is a demand for intelligent systems that can analyze system logs automatically. Such systems can detect the problems before they happen and create a time window for responsible technical teams to take action before downtime happens. Human labor can often detect only the presence of error in logs. However, an intelligent system can detect the absence of a specific log message that can forewarn a problem in the system.

Modern systems produce vast amounts of logs. Some large systems produce 120-200 million lines of records per hour [4]. Therefore, log processing is a big data problem.

Dealing with big data brings its challenges. Traditional data processing techniques fail to deal with the well-known characteristics of big data, namely “volume”, “variety”, and “velocity”. These are called 3Vs of big data [5]. For console log problems, the volume comes from the size of the system, variety comes from the subsystems of a system (DNS, container engine, storage, databases). Velocity in such problems usually arises from the high-speed occurrence of many events including significant events, such as failures. For example, if the DNS subsystem fails, depending on the type of failure, it might not produce any logs, yet, all the other systems will start producing errors logs rapidly.

Components of the information technology production systems are regularly updated to newer versions for extra stability and performance. With these updates, old log lines can be removed from the source code, and new lines of logs can be added to the system. The

structure of the existing logs can change with updates. These changes have adverse effects on rule-based anomaly detection systems; artificial intelligence-based systems on the other hand just need to be retrained to adapt to the new format.

Machine learning is a subset of Artificial Intelligence. The design of such learning systems revolves around the idea that machines can learn from the proper amount of data under the right conditions. Essential elements and design principles of such systems are derived from the neuron structure of the human brain. Such applications had been very successful for problems like Natural Language Processing (NLP). There have been many new and exciting applications and designs for machine learning problems [6]. Such designs these days are mostly based on deep learning algorithms.

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN). LSTM uses a special memory cell structure to converse its error ratio. Such models

show great promise for learning from sequential data and for working on the NLP

problem [7]. The Seq2seq model uses LSTM networks with attention optimizations. The primary components of Seq2seq models are encoder and decoder networks. The encoder turns the given sequence into a corresponding hidden vector and the decoder reverses this process.

1.1 Problem Definition

As the number of computer systems, which affect our daily lives, keeps growing, their availability is getting more important for us. Nowadays, problems in a single server can quickly affect millions of people around the planet. Detecting such problems without human intervention is very important for keeping such systems available all the time.

Automatically processing these big data logs and detecting anomalies in them using the deep learning method are required to achieve this task.

1.2 Aim of the Study

We aim to utilize big data processing and deep learning methods to create a model that can detect anomalies in system logs. A framework to detect system anomalies by only using the time differences between the log lines is proposed. The approach uses LSTM based Seq2seq networks to detect anomalies.

1.3 Layout of Thesis

The chapters of this thesis are arranged as follows. Chapter 2 explains the related work in this research area. Chapter 3 presents the proposed method for intelligent anomaly detection in system logs in detail. Experimental evaluation is presented and discussed in Chapter 4, and a conclusion is provided in Chapter 5.

CHAPTER 2

LITERATURE REVIEW

A framework for the intelligent detection of anomalies in system logs usually consists of four different parts (Figure 1). “Log Collection” systems collect the logs; “Log Parsing” systems transform the logs into a different structure, and a “Feature Extraction” system extracts the critical features for models to detect anomalies. And, an automatic “Anomaly Detection” system that is based on the previously learned models. Here we review these parts in the context of previous work.

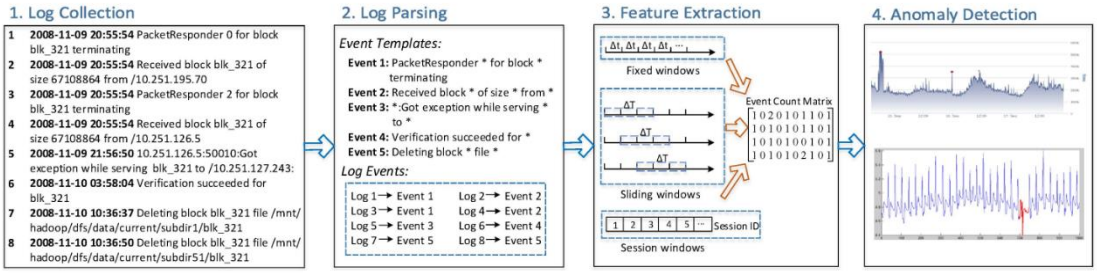

Figure 1 General Steps of Anomaly Detection [8]

2.1 Log Collectors

Computer systems generate logs on a regular basis to record runtime information. These logs contain valuable information for anomaly detection. Depending on the configuration, some systems write their logs into their storage. For analysis purposes, these logs need to be collected at a central location. Log collection systems take care of this task. Log collection is handled by specialized systems that store the logs for

permanent history or temporary inspection. These systems are planned for the workloads. Log collection presents more of a technical challenge than an academic challenge. Almost all of the production systems utilize some form of log collection solution. For example, BSD Syslog protocol [9] is very widely used in log collection and comes as a default in Linux. It can ship produced logs over a network.

Elasticsearch[6] is a search tool running on a distributed NoSQL database for storing full-text information in a non-relational manner. It is commonly used for log processing. Elastic beats provides a modern alternative to the Syslog with its log collector modules [10]. Elastic log collectors ship data from applications to the Elasticsearch database for storage and analysis.

These systems can operate in a pull-based or push-based manner. Pull based systems regularly pull the logs from application servers. Push based systems send the logs from application servers to log storage servers and usually transfers the logs faster than the other approach.

2.2 Log Parsers

The goal of parsing is to group raw logs into numeric tokens based on the static parts of the logs. Different researchers have given different names to these keys. They are called message types [11], log key [12], or event type [8], and they all refer to the same concept.

[6] https://www.elastic.co/

There are many different log parsing methods proposed in the literature. All have their advantages and disadvantages. The quality of parsing directly affects the performance of anomaly detection [13].

Spell (Streaming Parse for Event Logs) is an online streaming method to parse logs [14]. The design stands on the assumption that when we view the logline as a sequence, in most outputs, most of the line will consist of constants. This method works in a streaming manner (Figure 2).

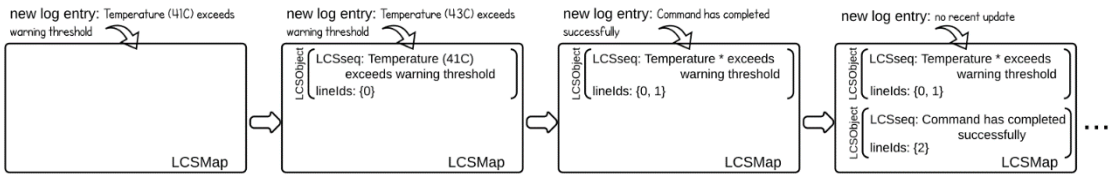

Figure 2 Spell Log Parsing Steps [14]

He, Pinjia et al. [15] proposed the Drain method. It is a representation algorithm for log parsing, namely, a fixed depth tree-based online log parsing. It can work in streaming and timely manner. The method utilizes a fixed depth parse tree. This method works with regular expressions that are created by using domain expertise. These regular expressions are used to parse logs. This method extracts the block ids from the HDFS dataset and parses logs (Figure 3).

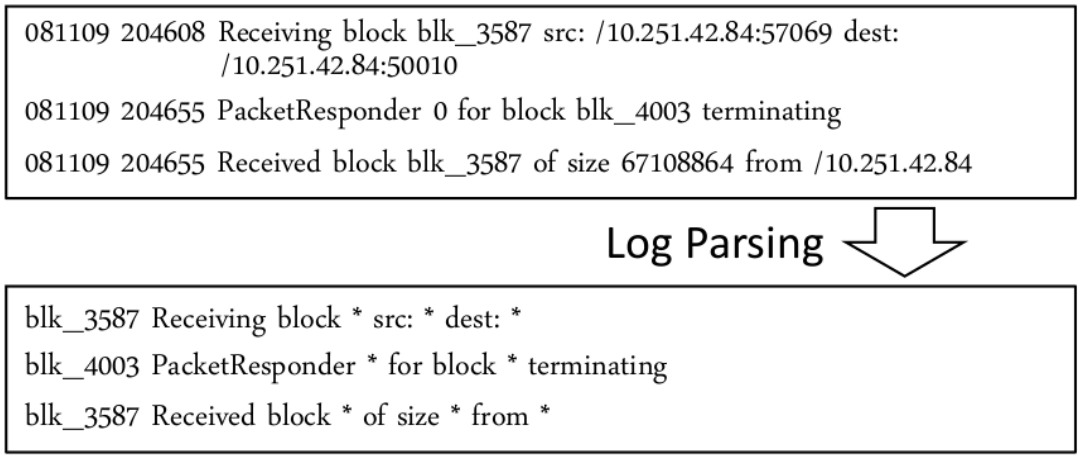

Figure 3 Log Parsing Example [15]

The first layer of nodes is created by using the length of the logline. This helps drain to be useful for online parsing. Leaf nodes are created by using the first word in the log as a token. When a new log arrives at the same leaf, it compares the similarity between log messages to decide whether to put this node into the existing log group or create a new group for the log. (Figure 4). Otherwise, a new group will be created for the log. Regex parsing prevents branch explosion.

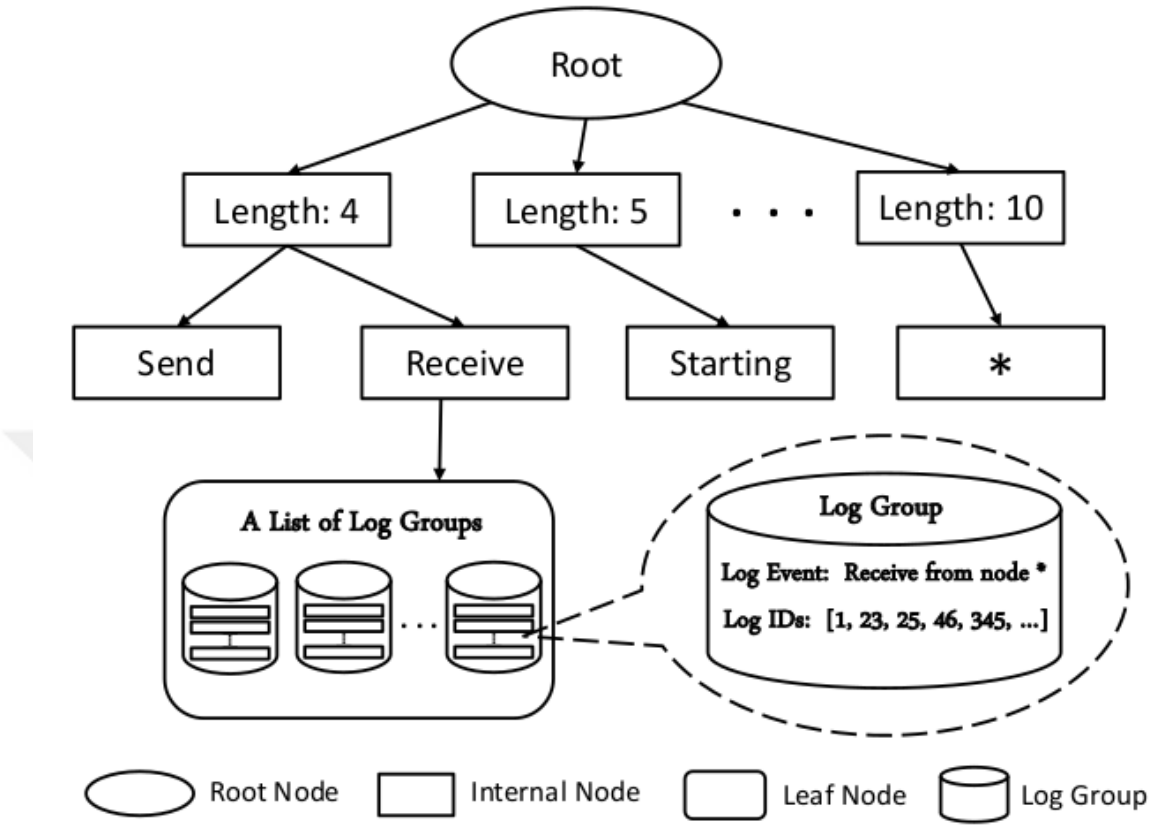

Figure 4 Drain Structure [15].

SLCT is another method for parsing logs, designed to cluster log files, so each cluster represents a particular line pattern. SLCT starts with making a pass over words in the logs and counting their occurrences and checking their positions. Words that appear more than a certain threshold are defined as frequent words; they are saved in the cluster candidates' table. After the first stage, SLCT makes a second pass over the logs, focusing only on frequent words. This second pass selects the cluster candidates. The last mandatory step is to prepare the output templates. Logs that are not chosen by the method are considered as outliers. [16]

Makanju et al. [17] proposed another parsing algorithm called IPLoM. This method takes advantage of the unique characteristics of log messages to extract message types.

This method starts with partitioning the data by event size, putting logs into a different cluster based on the log length. After clustering logs by lengths, in the second step, IPLoM clusters the data by token positions. This step is based on the assumption that it is likely for a column with the least number of unique words to be constants in that position. After this step, IPLoM starts to search for bijection. Two columns of the logs are selected for further partitioning by investigating the relationship between them. The last step of the algorithm is the log key extraction. For each column, the number of unique words is counted to choose wildcards and constants (Figure 5).

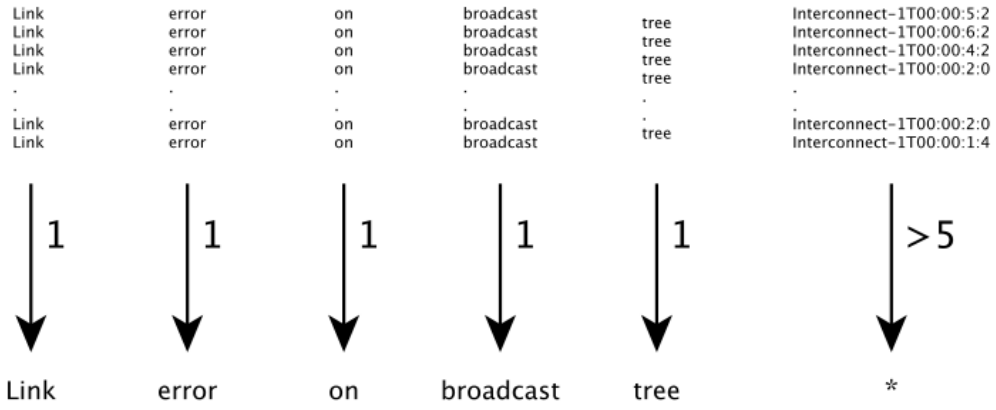

Figure 5 Partitioning of Logs [17]

Another method proposed for this problem is the LKE (Log Key Extraction) method. This method starts with erasing parameters by empirical rules. The second step of the algorithm is the raw log key clustering; it calculates the number of edit operations needed to transform a log into another log. After this operation, LKE splits the groups by checking the largest common sequence. By doing this, the method finds the constant

parts and the dynamic parts of the logs. The last step of the algorithm is to extract log keys from already split groups [18]. The steps of the algorithms are shown in Figure 6.

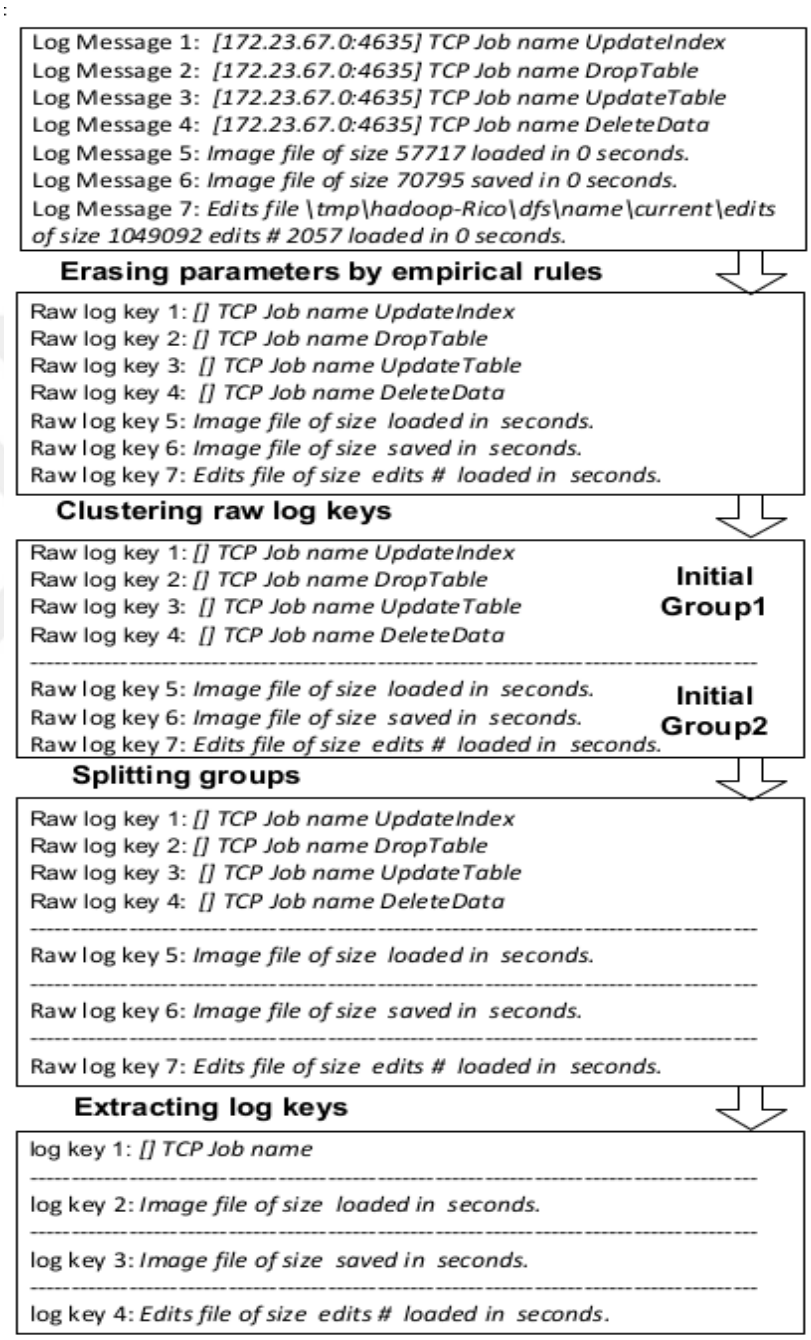

Figure 6 LKE Steps [18]

Tang, Liang et al [19] Proposed another method for parsing logs, namely LogSig. This method generates word pairs from log each line of the log. Users need to determine the

number of clusters using domain knowledge. LogSig uses this information to parse the logs and move them to clusters if needed. Finally, the method extracts log keys from these clusters.

2.3 Log Feature Extraction

After parsing the logs, related features need to be extracted from the logs. Windowing methods can be used for this task. Different windowing methods are applied to separate log data into groups [20].

Time series data can be windowed by fixed time length; all events that happened in a fixed time window are considered a sequence. Sliding windows can be used instead of a fixed window; this method will produce more sequences, and events in these sequences can be repeated depending on the step size. Another approach to windowing is session windows. For this method, a unique identifier is needed to generate the sequences. This method groups the logs according to the objects or tasks and sequences consist of events related to the same object or task [8].

Windowing with fixed time frames is applied to extract features is based on creating fixed time frame windows and merging all the events that happened in that time frame in those windows. Each time window has a fixed size, and logs that occurred in the same window are regarded as log sequences [8].

2.4 Log Anomaly Detection

General methods for detecting anomalies can be grouped as shown in Figure 7. Log anomaly detection methods can be divided into two “Programmed” and “Self-Learning”

groups. Programmed methods include “Rule-Based”, “Limit Based” and “Statistical” methods. Self-Learning methods are usually based on machine learning methods.

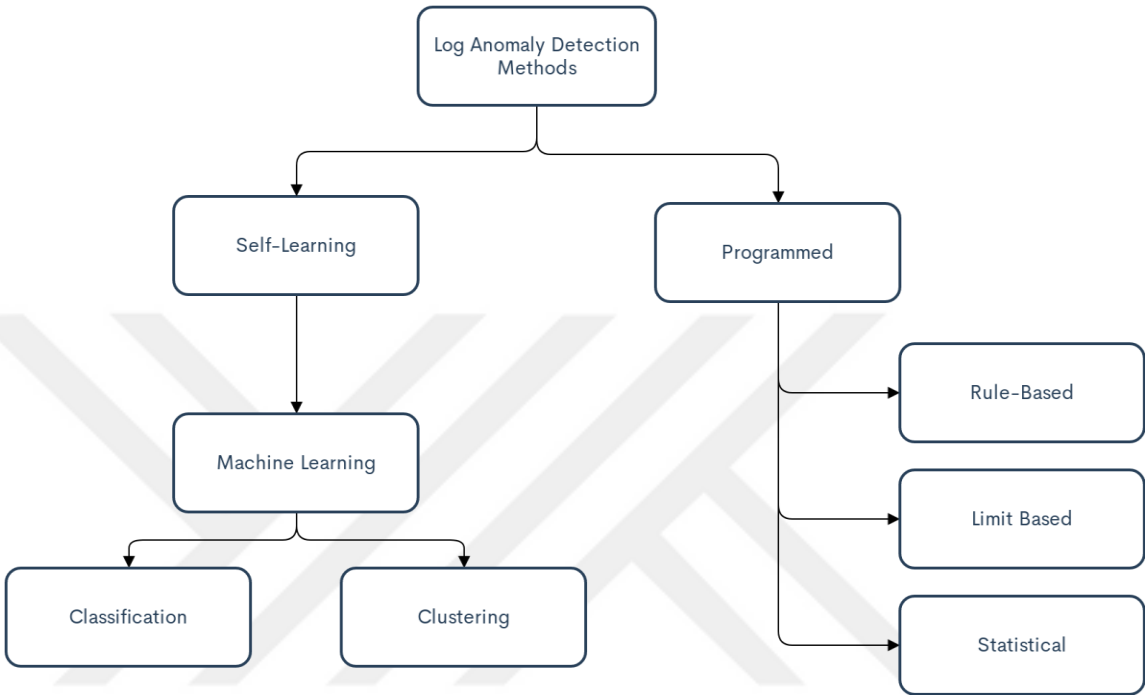

Figure 7 Anomaly Detection Methods

Chandola et al. [21] compares different techniques for detection anomalies, discussing their strengths and weaknesses. They conclude that unsupervised methods are not suitable for anomaly detection at a large scale as they assume that anomalies in the data are rare and not the case for all log types.

Another work provides an overview of studies done in the area of anomaly detection [22]. They categorize existing detection methods and systems based on underlying computational techniques and show that classification is the most popular approach among these studies.

2.4.1 Programmed Methods

Earlier works about the log anomaly detection were heavily dependent on rules and domain expertise. These approaches can be more accurate than the neural network-based approaches, but they require domain expertise and labor as well as manual rule updates to stay functional [23]. Yen et al. [24] proposes the Beehive method, which identifies potential anomalies by unsupervised clustering of features and manually labels these clusters. Another downside of rules-based designs is that they usually focus on detecting a particular type of anomalies. This system, called PerfAugur, is designed for identifying performance issues [25]. Bao et al. [26] mention that such methods are also limited to specific applications and require domain expertise.

2.4.2 Self-Learning Methods

Over the years, there have been many studies about log-based anomaly detection, which utilizes log count vectors. These log count vectors hold events in a log sequence. Lou, Jian-Guang et al. [27] proposed the Invariant Mining Method to mine linear relationships between log events from count vectors. New logs are compared with invariants to detect if they include anomalies. Xu et al. [28] utilize the Principal Component Analysis to construct anomaly and normal spaces for log count vectors. The distance of the log count vector to the normal space determines if there is an anomaly.

Classical neural networks are designed to mimic the neural structure of the human brain. A standard neural network is made of many connected units, which are called neurons. These neurons act as processors. At the one end of any network, there are input neurons, which are fed with information about the problems. These neurons process the input and feed the other neurons in the system over weighted connections between them.

Depending on the data, some neurons will be triggered by other neurons. A neural network will learn to exhibit desired behavior by changing the weights of its connections. Basic neural networks are also called shallow neural networks and have been around for the last few decades [29]. Deep learning networks are neural networks with multiple hidden layers. They usually consist of hierarchical architecture with multiple layers. Such networks can represent higher complexities [30].

LeCun et al. [31] first proposed the basis for Convolutional Neural Networks. Such networks can efficiently extract special features in a parallel fashion [32].

Long Short-Term Memory networks are often used for detection. They were first proposed in 1997. One of the design's main features was being less prone to the vanishing gradient problem because of the constant error flow in the design. The model uses gate units to avoid input weight conflicts (Figure 8). These networks have memory cells [7].

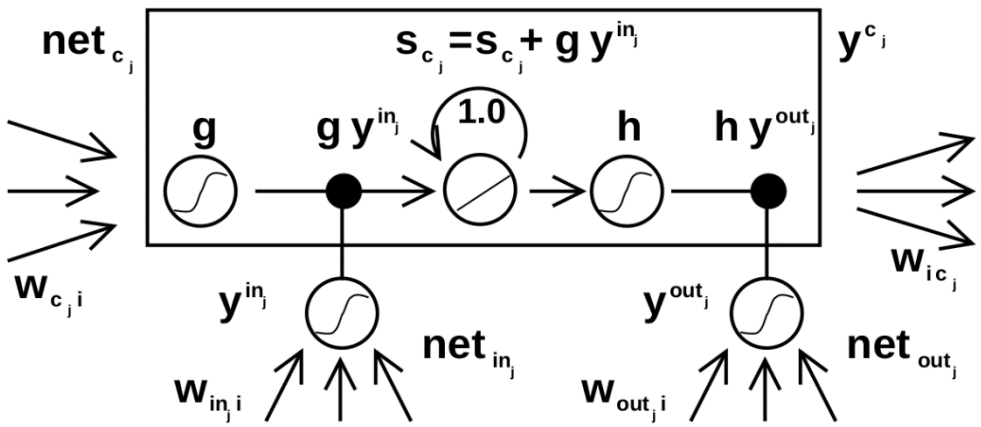

Figure 8 LSTM Network Node [7]

GRU networks control the flow of information with gate units, similar to LSTM networks. But they do not use a memory unit. This makes such networks more computationally efficient [33].

Vaswani et al. proposed an attention mechanism to improve the LSTMs. Attention works by mapping a query and a set of key-value pairs to output, all of which are vectors. The output is computed by a weighted sum of values [34].

The Seq2seq model is based on Encoder and Decoder networks [35]. This method uses LSTM with attention mechanism. Attention improves the performance of the classical LSTM. This model is often used for Neural Machine Translation (NMT) tasks [35] [36] [37] [38] [39] [40] [41]. Seq2seq models can also be used for anomaly detection by teaching networks to reconstruct the normal sequences. Such models fail to reconstruct the sequences when anomaly sequences are received [42] [43]. This approach has also been applied to system logs. [44]

CausalConvLSTM [32] method utilizes CNN networks with the LSTM network at the same time to take advantage of both networks for anomaly detection in system logs. LSTM captures sequential relationships and CNN extracts the special features.

Du, M. et al. [12] created a popular console log dataset by renting 200 EC2 instances from Amazon Web Services to host a HDFS cluster. This dataset has 11, 197, 954 log entries, and 2.9% of them are anomaly events. A domain expert labeled all the data by hand. They also generated an OpenStack dataset for testing. They considered logs of natural language and used NLP methods for parsing the logs before training. They used LSTM networks to detect probability distributions of log sequences. This network

detects sequential patterns. Based on earlier logs, the network predicts the next event and compares the predicted event with a real event. Based on the comparison, the system decides if the event is normal or anomaly. This method supports both online and offline processing.

You, Chenyu et al. [45] developed a Stacked Bidirectional LSTM Network for anomaly detection in system logs. Their models encode the entire log messages such as timestamps, TCP statistics, and packet values. They compared LSTM, stacked LSTM and bidirectional stacked LSTM with a different attention mechanism. They did not cover multi-domain computer systems. In their test, Stacked Bidirectional LSTM with multiplicative attention created the best accuracy.

Nedelkoski et al [46] Presented an approach using AEVB (Auto-Encoding Variational Bayes) and GRU networks. They used HTTP URL, IP address, service name, request type for their log sources. They tested with data from a microservice-based global cloud service provider. They focused on keeping prediction time short for industrial applications. Their method reached 90% accuracy with a prediction time lower than 10ms.

Zhang et al. [47] propose a general log analysis approach for learning regular expressions from heterogeneous logs, which reduces feature space and sparsity. They formalized the problem as a sequential classification problem, and they use an LSTM neural network to predict failures.

Hao et al. [48] This work focuses on Web Application Firewall applications of Bi-LSTM. They used word2vec to encode their logs. They analyzed the logs with Bi-LSTM

network to overcome the vanishing gradient problem. They used CSIC 2010 dataset. They achieved a 98.17% detection rate with a test accuracy of 98.35% by training for ten epochs.

Another framework called loganomaly, leverages template2vector [49] method to extract semantic information from logs. This helps this framework to parse log which it has not seen before in an online manner. They found this method produces much less false alarms than the compared systems [50].

Logrobust framework is designed to deal with log instability [51]. According to the authors, such stability can come from many sources. Firstly, as the newer versions of the software get released, the logging statements get changed. Collection, preprocessing, and retrieval operations can create noise in the log distribution. It deals with this problem by transforming every log into a semantic vector. These vectors are later sequenced before entering the attention-based bidirectional LSTM model (Figure 9).

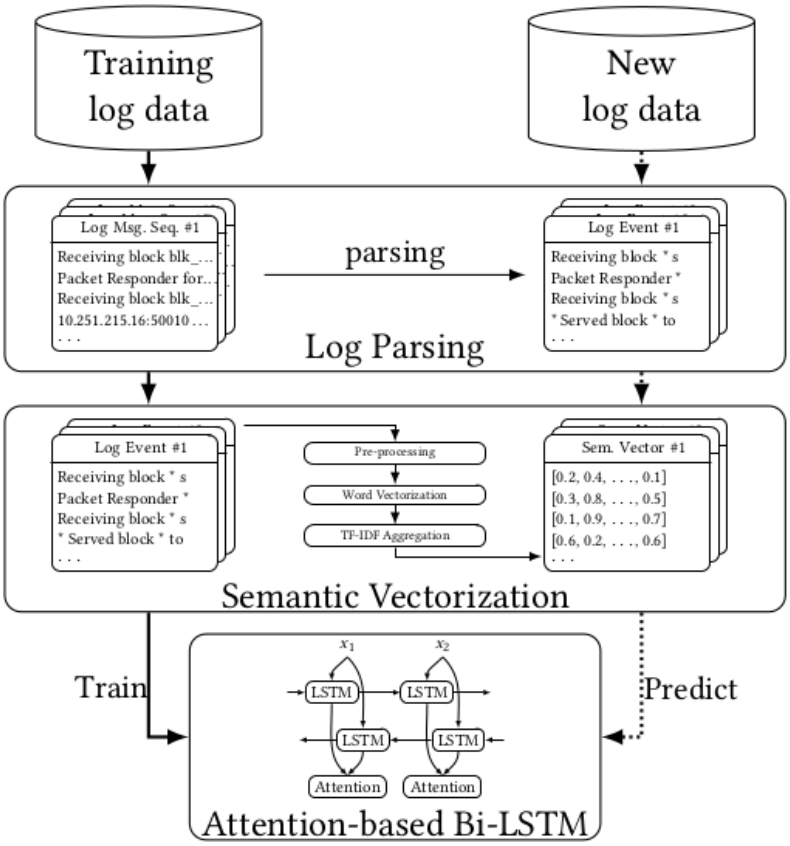

Figure 9 Log Roboust Framework [53]

In another study, the authors compared the results with different implementations [52]. Later same authors study this method for parallel processing with DILAF. DILAF stands for DIstributed Log Analysis Framework for anomaly detection in large-scale software systems [53]. This system works without inspecting the source code of the application, which most of the existing parsers are also capable of [14] [16] [17]. The main focus of the framework is scalability and parallel processing.

Lu, et al. [54] proposed CNN based method for detecting anomalies. This work utilizes logkey2vec embedding layers tree 1D convolutional layers. They used the HDFS dataset for testing. Max-overtime pooling is used for picking up the maximum values.

CHAPTER 3

This section presents information about the methods we developed to detect anomalies in system logs. We discuss the difference between log key-based parsing and time difference-based parsing approaches. We parse the logs with both methods and preprocess the data for time difference-based approach. We train a Seq2seq model to reconstruct the normal log sequences. Then, we detect anomalies by reconstructing sequences in the trained model.

3.1 Detecting Anomalies in Logs Using RNN

Usually, system log anomaly detection systems consist of four different parts, log collection, log parsing, feature extraction, and anomaly detection. Log parsing steps use different methods to extract log keys from the data source and tag every line of the log with a proper log key. Later during the feature extraction, these events are transformed into event sequences. This step allows the neural network to understand the data more accurately. During this stage, the exact timestamp information of the logs is often removed from the data. During this stage, the exact timestamp information of the logs is often removed from the data. The only time-related information that remains in the sequences is the order of events. This approach is useful for failure detection as most of the failure produces unique logs (stack traces, exceptions, warnings). Such log lines are

not present in logs from healthy operations. Yet creating these log keys from the given log files requires an extra step, which requires extra computation and effort.

We propose a different approach for the log parsing and feature extraction steps. Instead of tokenizing logs and removing the time information from the records, we extracted only the time session.

Since our approach uses only the time differences between any two events, it completes parsing and feature extraction in a single step. For the HDFS dataset normal data is split into test and training groups but all anomaly data is used for testing (Figure 10).

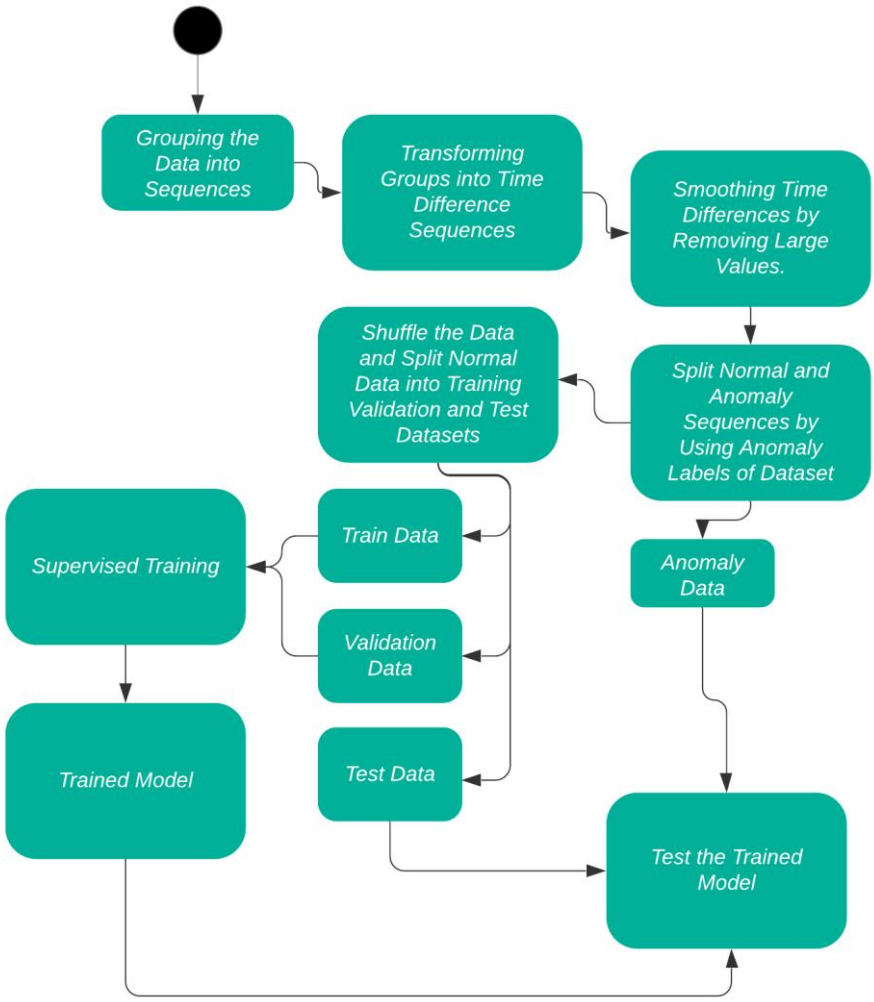

Figure 10 Steps of Data Processing

We make references to the weak points of the state-of-the-art studies, and present our solutions to these weak points below.

- Most of the similar studies mentioned above remove the time information from the data before detecting anomalies [27] [28] [3] [47] [54]. We only focus on the time difference information and completely ignore the log key information. Our results clearly show, there is valuable information in these time differences.
- Existing methods [14] [15] [16] [17] [18] require a log parser method to extract log keys from the logs. Depending on the methods, this step might require domain expertise. Not all parsers can function in an online manner. However, our method happens to functions online without using log keys. It can be plugged into any production system without any domain information.
- Unlike the methods which discard the time information [27] [28] [3] [47] [54], our method can detect performance problems that do not produce any logs related to the issue.

3.1.1 Log Parsing

For the first step of our method, we need to group raw data based on a unique ID. For the HDFS dataset, anomaly labels are given for HDFS block IDs, which is also used as unique ID for grouping logs into sequences. Depending on the system logs, any unique ID can be used to group the events. Request ID, job names, or even IP addresses can be used for grouping.

These groups include all the events for the given block. These events later transformed into sequences of time differences between any two events belonging to the same block ID. The time difference for the first event of the given group is set to 0. Events logged in the same second also get the time difference of 0.

Most works use key-based classical encoding methods for this step. Classical encoding methods convert the log rows into log keys. These keys represent the log key of the given row. To show the difference between the proposed time difference-based parser and the classical key-based parser, the given lines are also converted to log keys by using the Drain method [15]. This is only for comparison reasons. Drain [15] method requires a small amount of domain information, but it assures near-perfect tokenization. The Drain method's results are not used in our approach and are given only to show the difference between log key-based parsing and time difference-based parsing. Table 1 presents the results of the same log sequence under different parsing methods.

Table 1 Time Difference Information Parsing

Event Number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Drain Encoding	1	1	2	14	3	15	19	20
Time Difference Encoding	0	0	0	60	1	0	657	0

Time differences are calculated as seconds since the dataset only included timestamps in seconds. The difference between some log records can be relatively large. Under this assumption events that occur more than 100 seconds apart are not very significant for our approach; values larger than 100 seconds are smoothed out with the following function. These values can get too high unless cleaning with such a function is done (Table 2).

Table 2 Time Difference Preprocessing Function

Condition	Result
$n \leq 100$	n
$n > 100^x$	100 + x

This function only alters the 7th event in our example as it is the only event larger than 100:

if n=657 then $n > 100^1$ and x = 1, therefore $n_{new} = 100 + 1 = 101$

Results of preprocessing of a sequence is presented in Table 3.

Table 3 Time Difference Processing

Event Number	1	2	3	4	5	6	7	8
Time Difference Encoding	0	0	0	60	1	0	657	0
Preprocessed Time Difference Encoding	0	0	0	60	1	0	**101**	0

3.1.2 Data Preparation

The given dataset comes with anomaly labels. These labels provide ground-truth for the block ID'S which had anomalies. In earlier steps, we grouped our data by the block ids.

Using anomaly labels, our data is split into two groups. Normal and anomaly data frames are created. Block ID information is discarded in this step since we no longer need this information. Within these two data frames, duplicate sequences are removed. After having cleaned the cleaning duplicates, we create two files (Table 4).

Table 4 Dataset Statistics

File Name	Number of Log Sequences	Percentage
Anomaly	16,838	3%
Normal	558,223	97%
Total	575,061	

Our method is based on the Seq2seq method, which is generally used as a supervised algorithm. Our approach uses the Seq2seq network in an unsupervised manner. The network is trained only using normal data and tested with abnormal and normal data. The normal file is split into test, training, and validation files. Training and validation files will be used during the training, and the test file is used for evaluating our results.

Before we split the normal file, we shuffled it based on hash values to make the results reproducible. Linux sort command from Linux coreutils[7] was used with a fixed seed. The bash[8] script used for sorting and splitting is available in Appendix A. Table 5 presents the resulting files. This file consists of ordered sequences, shuffling them by rows does not alter the order of events in sequences.

Table 5 Training Files

File Name	Number of Log Sequences	Split Percentage	Usage
Anomaly_test	16,838	100%	Evaluation
Normal_test	111,644	20%	Evaluation

[7] https://www.gnu.org/software/coreutils/manual/coreutils.html
[8] https://www.gnu.org/software/bash/

Normal_train	334,935	60%	Training
Normal_validate	111,644	20%	Training

The network is trained only using a partition of normal data. “Anomaly_test” and “Normal_test” are used to evaluate the results.

3.1.3 Anomaly Detection

Using these data files, the Seq2seq network is trained to reconstruct the sequences. We used the IBM Pytorch Seq2seq implementation[9]. Seq2seq network trains with normal sequences and learns to reconstruct normal sequences accurately. Table 6 presents the parameters of the implementation. Training is done with the following parameters.

Table 6 Seq2seq Parameters

Epochs	22
Teacher Forcing Ratio	0.5
Batch Size	32
Hidden Size	128

Finally, to sum up briefly the approach details, our approach:

- Compatible with all types of logs.
- Does not require domain knowledge.
- More capable of detecting performance issues.

[9] https://github.com/vincentzlt/ibm-pytorch-Seq2seqSeq2seq/

CHAPTER 4

EVALUATION

This section presents the evaluation of our methods, the test environment details, information about the dataset used, and the results in detail. Software libraries and hardware devices used for the experiment are explained. Dataset and properties of the data are explored. Finally, we compare our results to other studies.

4.1 Experimental Setup

4.1.1 Test Environment

This section represents the results of the proposed framework. Experiments are conducted on a workstation with an i7-7000 processor, 64GB memory, a Nvidia GeForce RTX 2080 graphics card. The computers run on Debian Linux 10 and have CUDA 10.1 libraries.

Pytorch[10] is a Python[11] library for developing neural network models that support multiple runtimes, including Nvidia Cuda[12]. Pytorch supports dynamic graphs, data parallelism, and training models using numerous GPUs on numerous servers at the same time.

[10] https://pytorch.org/
[11] https://www.python.org/
[12] https://developer.nvidia.com/cuda-downloads

Log parsing was done with custom code. The preprocessing was done with pandas[13] data frames. The neural network was trained and tested on the GPU.

The random seed for Pytorch is 2222.

4.1.2 Dataset

HDFS dataset was used for several experiments (Table 7). To create the dataset, the original authors have written all the logs to the local disk on each node and collected them after completing the tests. The collection was done in an offline manner by the basic copy operation. The test system had 200EC2 nodes. The software was an unmodified off-the-shelf version. The log level was set to default [28].

Table 7 HDFS dataset characteristics

Time Span	Messages	Data Size
38.7 hours	11,175,629	1.47GB

Dataset consists of the files presented in Table 8.

Table 8 HDFS Dataset Numbers

File Name	**Number of Logs**
HDFS.log	11,175,629
Anomaly_labels.csv	575,062

When counted by log keys, some regular events made up 61% of all logs (Table 9).

Table 9 Most occurring character groups

Log Key	**Number of Occurrences**

[13] https://pandas.pydata.org/

Receiving block <*> src: <*> dest: <*>,"	1,723,232
NameSystem.addStoredBlock: blockMap updated: <*> is added to <*> size <*>,	1,719,741
PacketResponder <*> for block <*> <*>,	1,706,728
Received block <*> of size <*> from <*>,	1,706,514

4.2 Evaluation Metrics

The following metrics were calculated to evaluate the model.

True Positive (TP)

The sequence was detected as an anomaly, and it was an anomaly. Detection is correct.

False Negative (FN)

The sequence was not detected as an anomaly, but it was an anomaly. Detection failed.

False Positive (FP)

The sequence was not detected as an anomaly, but it was an anomaly. Wrong detection.

True Negative (TN)

The sequence was not detected as an anomaly, but it was not an anomaly. Detection is correct.

Precision

Precision is defined as, what proportion of positives was correct.

$$Precision = \frac{TP}{TP + FP}$$

Recall

Recall was defined as what proportion of real positives were detected.

$$Recall = \frac{TP}{TP + FN}$$

F1 Measure

F1 measure is a measurement of test accuracy.

$$F1 = 2\ x \frac{Precision * Recall}{Precision + Recall}$$

4.3 Results

Our proposed framework works only by using time differences between events and does not require any special parser for log key extraction. Our method is based on detecting anomalies without looking into the content of the logs. Our feature extraction method removes this information. This makes the method more efficient and effective.

3.3.1 Difference-based Parser Experiment

The Seq2seq network trained with difference-based parser reproduced 111,072 normal sequences perfectly. This extraction is described in section 3.1.1. The network also managed to reproduce 7,185 anomaly sequences perfectly, which were anomalies. These anomalies got reported as negatives which hindered the overall detection performance of the network. Detection results of the experiment based on time differences parser are presented in the confusion matrix in Table 10.

Table 10 Confusion Matrix for Time Differences

Real Class	**Predicted Class**	
	Anomaly	**Normal**

Anomaly	True Positives	False Negatives
	9,653	7,185
Normal	False Positives	True Negatives
	572	111,072

3.3.3 Comparison of Experiment Results

To properly compare log parsing approaches with each other, we run the experiment with log keys extracted by key-based parser like other methods. This is only to compare the detection rate under the same detection circumstances. For this test, log keys are generated by Drain [15] to compare the methods' performance under the same circumstances.

Table 11 shows the number of false positives and false negatives based on key-based parser on HDFS data. Method assigned most of the samples to correct classes. A low false positive value means that alerts from this can be considered accurate.

Table 11 Confusion Matrix for Log Keys

Real Class	**Predicted Class**	
	Anomaly	**Normal**
Anomaly	True Positives	False Negatives
	16,477	**361**
Normal	False Positives	True Negatives
	982	**110,662**

Original sequences are compared with the reconstructed sequences for anomaly detection. If they are not the same, sequences are considered as an anomaly. We also noted the average similarity between original and reconstructed sequences for normal and anomaly data.

The average difference between the original sequence and produced sequences for normal logs are 0.9994621628598735. Which shows the networks constructed normal log sequences with a high similarity average.

The average difference between the original sequence and produced sequences for anomaly logs are 0.6799552992148196. These were much worse than normal sequences as expected.

The results of the two experiments are presented in Table 12. Both parsing methods achieved almost identical precision values. Log-key based parser achieved a much better recall value than the difference-based parser. Therefore, F1 score is much better for log-key based parser with 96%. Difference-based parser is much easier to implement and does not require a specialized log parser.

Table 12 Seq2seq Results

	Precision	**Recall**	**F1**
Seq2seq with Time Differences	0.9441	0.5733	0.7134
Seq2seq with Log Keys	0.9438	0.9786	0.9608

Both approaches shown similar precision values, meaning both models were able to produce most of the sequences successfully. Logs key-based method achieved much better recall and F1 values as it was more capable of detecting anomalies (Figure 11).

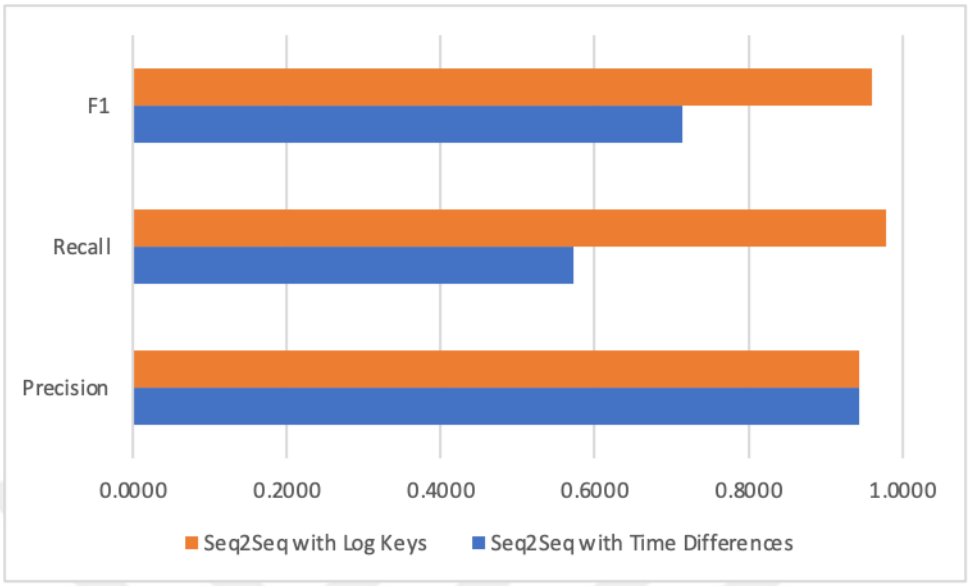

Figure 11 Comparison of Results

Time difference-based method does not require any specialized parsers. Log key-based methods require parsers [14] [15] [16] [17] [18] .

4.4 Comparison of Results

We compared our findings with works on the same dataset [12] [32] [52] [53] [54].

DILAF stands for DIstributed Log Analysis Framework for anomaly detection in large-scale software systems. This method uses Message Count Vectors as primary features [53]. CausalConvLSTM [32] achieved better results than compared GRU, LSTM, and CNN implementation for the problem. Our implementation works only with time differences and ignores the log keys, unlike the given CausalConvLSTM. Lu, Siyang et al. proposed a CNN with logkey2vec embedding layer [54].

All of the methods we are comparing against [32] [52] [53] [54] ignore the time difference information. Deeplog [12] uses both log key and timestamp information at the same time.

While comparing the results, it is essential to keep in mind; our implementation has no way of knowing the information in the given log event. There is no information about the difference between an error log that includes a stack trace and a notice log, which says the health check is successful. All other methods [12] [32] [52] [53] [54] has this information while detecting anomalies. Regardless of this lack of information, there is only 3% difference in precision between the time-based method and key-based methods (Figure 12).

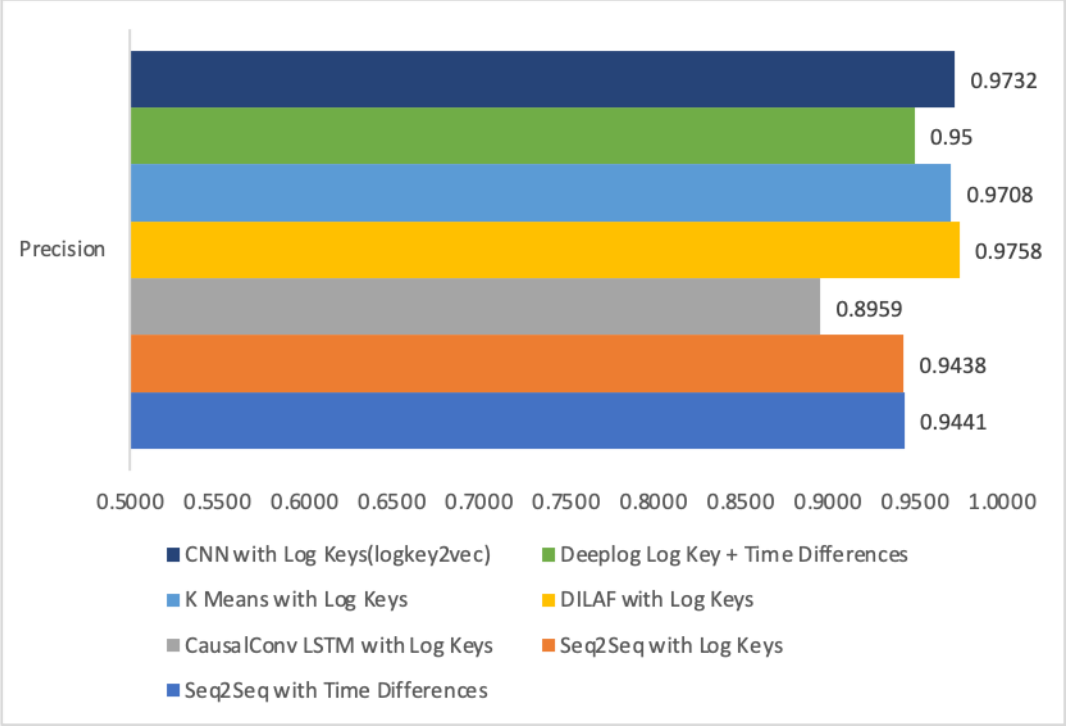

Figure 12 Comparison of Precision Values

When we compare the precision values, even without log key information, our method shows better precision values than CasualConvLSTM [32]. Precision value is worse than all other methods

Most of the failures in the given dataset were represented by special logs. These error logs correspond to log keys which were never present in the healthy logs. These error

messages make detection much easier for such cases. Even with a simple rule-based approach, such events could easily be detected. Since time difference-based extraction removes this information, our implementation suffers from False Negatives which gets reflected negatively in recall value (Figure 13). This extraction is described in section 3.1.1

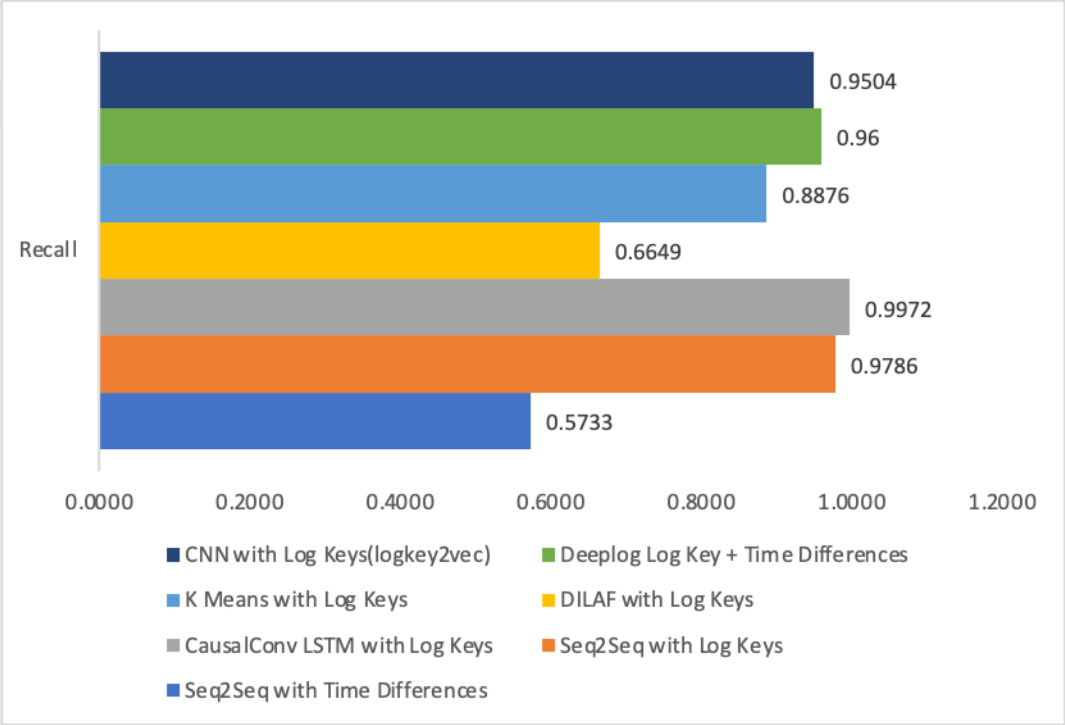

Figure 13 Comparison of Recall

Comparison between the F1 values of the methods which shows the overview of compared methods is presented in Figure 14. Only three methods achieved more than %96 and Seq2seq with key-based parser was one of them.

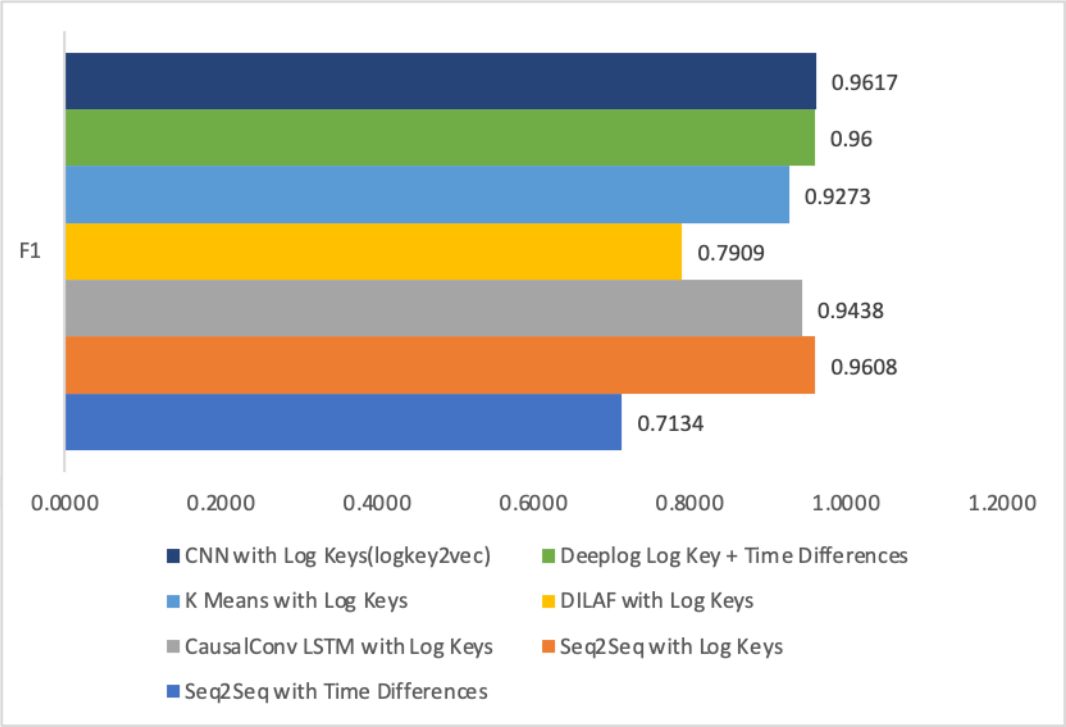

Figure 14 Comparison of F1 Values

The summary of the evaluation metrics of all the compared studies is presented in Figure 13. This shows the performance of our method compared to the other studies over the HDFS dataset using recall, precision, and F-measure.

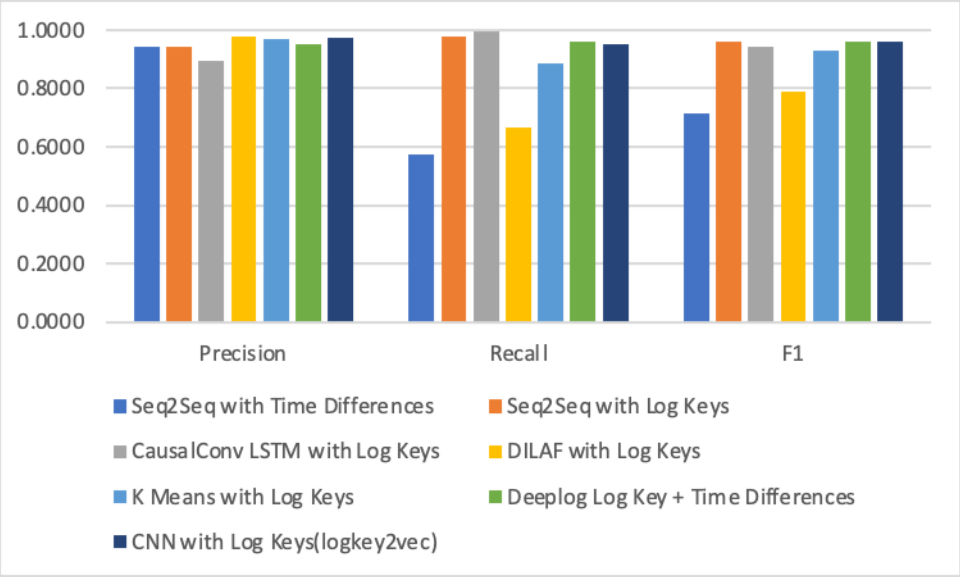

Figure 15 Seq2seq compared to other methods

Our approach produced a precision level similar to that of other studies using only the timing information. Results presented in Table 12 show that the difference-based parser shows the worst F1 value. Even though the value is worse compared to other studies, it is still an acceptable value. There is valuable information in the time differences between log events. It is important to keep in mind that difference-based parser does not know the information within the log line. Network succeeds at detecting anomalies by using only time differences within a reasonable level. Achieving the same feat by using the same data is near impossible with a human.

With proper implementation, we can detect anomalies in such a system by only working with timestamps.

Table 13 Comparison of Results

	Precision	Recall	F1
Seq2seq with Time Differences	0.9441	0.5733	0.7134
Seq2seq with Log Keys	0.9438	0.9786	**0.9608**
CausalConvLSTM with Log Keys	0.8959	0.9972	0.9438

DILAF with Log Keys	0.9758	0.6649	0.7909
K Means with Log Keys	0.9708	0.8876	0.9273
Deeplog Log Key + Time Differences	0.9500	0.9600	**0.9600**
CNN with Log Keys(logkey2vec)	0.9732	0.9504	**0.9617**

Our method is capable of processing logs in a streaming manner. Our method supports online processing. Some log parsers only work in an offline manner [16] [17] [18] [19], which requires batch processing which requires all the data to be available before processing. A recent study compared existing log parsers, showed that most of them are not capable of processing logs in an online manner [55]. Drain parser [15] supports online parsing but requires domain knowledge about the internals of the application. Spell [14] parser supports online parsing but it requires parameter tuning.

In real life deployments, extreme slowness in the systems is usually considered anomalies. Key-based parsers remove this information from data. This makes it much harder for pure key-based methods to detect slowness.

CHAPTER 5

CONCLUSION

Many anomaly detection systems have been developed with different approaches to protect computer systems from downtime. Such systems are still far away from detecting root causes without human intervention, yet identifying possible anomalies in such logs provides a valuable advantage for professional technical teams. Inspecting problems over a selected number of lines of logs can improve response times instead of reading through lengthy log files. New methods, like LSTM based Seq2seq networks, provide promising solutions for such issues.

Our proposed methods do not rely on any specific parser. Sequences were generated purely based on log timestamps and time differences between events. The Seq2seq model was later trained to reconstruct sequences. The difference between a given original sequence and a reconstructed sequence is used for detecting anomalies. HDFS dataset is used for testing. Since our model does not require any form of parsing, it can be plugged into any system without any domain expertise or modification.

Experiments have shown that anomaly detection with difference-based parsing using Seq2seq networks shown precision values close to the detection methods based on key-based parsers. Seq2seq networks also learned to reconstruct key-based sequences with a good performance by only using the normal data for training.

In future work, information from multiple log producers can be analyzed to create a framework that can learn the relationship between various systems. Most of the modern

systems allow sub-second level timing between log records. Such detailed timing systems can improve the performance of the time difference-based anomaly detection methods greatly.

REFERENCES

[1] Zhu, J., He, P., Fu, Q., Zhang, H., Lyu, M.R., & Zhang, D. (2015). Learning To Log: Helping Developers Make Informed Logging Decisions. 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, 1, 415-425.

[2] Chen, A. (2019). An Empirical Study on Leveraging Logs for Debugging Production Failures. 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), 126-128.

[3] Yu, X., Joshi, P., Xu, J., Jin, G., Zhang, H., & Jiang, G. (2016). loudSeer: Workflow Monitoring of Cloud Infrastructures via Interleaved Logs. In Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '16). Association for Computing Machinery, 489–502.

[4] Mi, H., Wang, H., Zhou, Y., Lyu, M.R., & Cai, H. (2013). Toward Fine-Grained, Unsupervised, Scalable Performance Diagnosis for Production Cloud Computing Systems. IEEE Transactions on Parallel and Distributed Systems, 24, 1245-1255..

[5] Safhi, H.M., Frikh, B., & Ouhbi, B. (2019). Assessing Reliability of Big Data Knowledge Discovery Process. Procedia Computer Science, 148, 30-36.

[6] Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (Eds.). (2013). Machine

Learning: An Artificial Intelligence Approach. Springer Science & Business Media.

[7] Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735-1780.

[8] He, S., Zhu, J., He, P., & Lyu, M.R. (2016). Experience Report: System Log Analysis for Anomaly Detection. 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), 207-218.

[9] Lonvick, C. (2001). the BSD Syslog Protocol. RFC, 3164, 1-29.

[10] Rochim, A.F., Aziz, M.A., & Fauzi, A. (2019). Design Log Management System of Computer Network Devices Infrastructures Based on ELK Stack. 2019 International Conference on Electrical Engineering and Computer Science (ICECOS), 338-342.

[11] Haque, A., Delucia, A., & Baseman, E. (2017). Markov Chain Modeling for Anomaly Detection in High Performance Computing System Logs. Fourth Annual Workshop on HPC User Support Tools, Article 3, 1–8.

[12] Du, M., Li, F., Zheng, G., & Srikumar, V. (2017). Deeplog: Anomaly Detection and Diagnosis From System Logs through Deep Learning. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 1285–1298.

[13] He, P., Zhu, J., He, S., Li, J., & Lyu, M.R. (2018). Towards Automated Log Parsing for Large-Scale Log Data Analysis. IEEE Transactions on Dependable and Secure

Computing, 15, 931-944.

[14] Du, M., & Li, F. (2019). Spell: Online Streaming Parsing of Large Unstructured System Logs. IEEE Transactions on Knowledge and Data Engineering, 31, 2213-2227.

[15] He, P., Zhu, J., Zheng, Z., & Lyu, M.R. (2017). Drain: An Online Log Parsing Approach With Fixed Depth Tree. 2017 IEEE International Conference on Web Services (ICWS), 33-40.

[16] Vaarandi, R. (2003). A Data Clustering Algorithm for Mining Patterns From Event Logs. Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM 2003) (IEEE Cat. No.03EX764), 119-126.

[17] Makanju, A., Zincir-Heywood, A.N., & Milios, E.E. (2009). Clustering Event Logs Using Iterative Partitioning. International Conference on Knowledge Discovery and Data Mining, 1255–1264.

[18] Fu, Q., Lou, J., Wang, Y., & Li, J. (2009). Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis. 2009 Ninth IEEE International Conference on Data Mining, 149-158.

[19] Tang, L., Li, T., & Perng, C. (2011). Logsig: Generating System Events From Raw Textual Logs. In Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM '11). Association for Computing Machinery, 785–794.

[20] Akidau, T., Bradshaw, R., Chambers, C., Chernyak, S., Fernández-Moctezuma, R., Lax, R., Mcveety, S., Mills, D., Perry, F., Schmidt, E., & Whittle, S. (2015). The Dataflow Model: A Practical Approach To Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-Of-Order Data Processing. Proc. VLDB Endow, 8, 1792-1803.

[21] Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly Detection: A Survey. ACM Computing Surveys, Article 15,58.

[22] Bhuyan, M.H., Bhattacharyya, D.K., & Kalita, J.K. (2014). Network Anomaly Detection: Methods, Systems, and Tools. IEEE Communications Surveys & Tutorials, 16, 303-336.

[23] Cinque, M., Cotroneo, D., & Pecchia, A. (2013). Event Logs for the Analysis of Software Failures: A Rule-Based Approach. IEEE Transactions on Software Engineering, 39, 806-821.

[24] Yen, T., Oprea, A., Onarlioglu, K., Leetham, T., Robertson, W., Juels, A., & Kirda, E. (2013). Beehive: Large-Scale Log Analysis for Detecting Suspicious Activity in Enterprise Networks. In Proceedings of the 29th Annual Computer Security Applications Conference 2013, 199–208.

[25] Roy, S., König, A.C., Dvorkin, I., & Kumar, M. (2015). Perfaugur: Robust Diagnostics for Performance Anomalies in Cloud Services. 2015 IEEE 31st International Conference on Data Engineering, 1167-1178.

[26] Bao, L., Li, Q., Lu, P., Lu, J., Ruan, T., & Zhang, K. (2018). Execution Anomaly Detection in Large-Scale Systems through Console Log Analysis. J. Syst. Softw., 143, 172-186.

[27] Lou, J., Fu, Q., Yang, S., Xu, Y., & Li, J. (2010). Detecting Large-Scale System Problems by Mining Console Logs.. Int. Conf. on Machine Learning, 37-46.

[28] Xu, W., Huang, L., Fox, A., Patterson, D., & Jordan, M.I. (2009). Detecting Large-Scale System Problems By Mining Console Logs. 22nd Symposium on Operating Systems Principles , 117–132.

[29] Schmidhuber, J. (2015). Deep Learning in Neural Networks: An Overview. Neural Networks : The Official Journal of the International Neural Network Society, 61, 85-117.

[30] Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., & Alsaadi, F.E. (2017). A Survey of Deep Neural Network Architectures and the ir Applications. Neurocomputing, 234, 11-26.

[31] Lecun Y., Haffner P., Bottou L., Bengio Y. (1999) Object Recognition With Gradient-Based Learning. In: Shape, Contour and Grouping in Computer Vision. Lecture Notes in Computer Science, Vol 1681. Springer, Berlin, Heidelberg.

[32] Yen, S., Moh, M., & Moh, T. (2019). Causalconvlstm: Semi-Supervised Log Anomaly Detection through Sequence Modeling. 2019 18th IEEE International Conference on Machine Learning and Applications , 1334-1341.

[33] Cho, K., Merrienboer, B.V., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations Using Recurrent Neural Network Encoder-Decoder for Statistical Machine Translation. Empirical Methods in Natural Language Processing, 1724-1734.

[34] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., & Polosukhin, I. (2017). Attention Is All You Need. Arxiv, Abs/1706.03762.

[35] Sutskever, I., Vinyals, O. & Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. Advances in Neural Information Processing Systems 3104-3112.

[36] Liu, L., Malak, D., & Médard, M. (2019). Guesswork for Inference in Machine Translation With Seq2seq Model. 2019 IEEE Information the ory Workshop (ITW), 1-5.

[37] Dessì, R., & Baroni, M. (2019). Cnns Found To Jump Around More Skillfully Than Rnns: Compositional Generalization in Seq2seq Convolutional Networks. Arxiv, Abs/1905.08527.

[38] Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural Machine Translation By Jointly Learning To Align and Translate. Computing Research Repository, Abs/1409.0473.

[39] Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., & Hovy, E. (2016). Hierarchical Attention Networks for Document Classification. Conference of the North American Chapter of the Association for Computational Linguistics: Human

Language Technologies, 1480-1489.

[40] Gehring, J., Auli, M., Grangier, D., & Dauphin, Y. (2017). A Convolutional Encoder Model for Neural Machine Translation. Association for Computational Linguistics, 123–135

[41] Yao, L., Torabi, A., Cho, K., Ballas, N., Pal, C., Larochelle, H., & Courville, A.C. (2015). Describing Videos By Exploiting Temporal Structure. 2015 IEEE International Conference on Computer Vision , 4507-4515.

[42] Shin, D., Park, R.C., & Chung, K. (2020). Decision Boundary-Based Anomaly Detection Model Using Improved Anogan From ECG Data. IEEE Access, 8, 108664-108674.

[43] Malhotra, P., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P., & Shroff, G. (2016). LSTM-Based Encoder-Decoder for Multi-Sensor Anomaly Detection. Arxiv, Abs/1607.00148.

[44] Wu, P., Lu, Z., Zhou, Q., Lei, Z., Li, X., Qiu, M., & Hung, P. (2019). Bigdata Logs Analysis Based on Seq2seq Networks for Cognitive Internet of Things. Future Generation Computer Systems, 90, 477-488.

[45] You, C., Wang, Q., & Sun, C. (2019). Sbilsan:Stacked Bidirectional Self-Attention LSTM Network for Anomaly Detection and Diagnosis From System Logs.

[46] Nedelkoski, S., Cardoso, J., & Kao, O. (2019). Anomaly Detection and Classification Using Distributed Tracing and Deep Learning. 2019 19th IEEE/ACM

International Symposium on Cluster, Cloud and Grid Computing (CCGRID), 241-250.

[47] Zhang, K., Xu, J., Min, M.R., Jiang, G., Pelechrinis, K., & Zhang, H. (2016). Automated IT System Failure Prediction: A Deep Learning Approach. 2016 IEEE International Conference on Big Data, 1291-1300.

[48] Hao S., Long J., Yang Y. (2019) BL-IDS: Detecting Web Attacks Using Bi-LSTM Model Based on Deep Learning. In: Li J., Liu Z., Peng H. (Eds) Security and Privacy in New Computing Environments, Vol 284, 551-563.

[49] Mikolov, T., Chen, K., Corrado, G.S., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. Computing Research Repository, abs/1301.3781.

[50] Meng, W., Liu, Y., Zhu, Y., Zhang, S., Pei, D., Chen, Y., Zhang, R., Tao, S., Sun, P., & Zhou, R. (2019). Loganomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs. International Joint Conference on Artificial Intelligence.

[51] Zhang, X., Xu, Y., Lin, Q., Et Al.(2019). Robust Log-Based Anomaly Detection on Unstable Log Data. Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 807–817.

[52] Astekin, M., Zengin, H., & Sözer, H. (2018). Evaluation of Distributed Machine Learning Algorithms for Anomaly Detection From Large-Scale System Logs: A

Case Study. 2018 IEEE International Conference on Big Data, 2071-2077.

[53] Astekin, M., Zengin, H., & Sözer, H. (2019). DILAF: A Framework for Distributed Analysis of Large-Scale System Logs for Anomaly Detection. Software: Practice and Experience, 49, 153 - 170.

[54] Lu, S., Wei, X., Li, Y., & Wang, L. (2018). Detecting Anomaly in Big Data System Logs Using Convolutional Neural Network. 2018 IEEE 16th International Conference on Dependable, Autonomic and Secure Computing, 151-158.

[55] Zhu, J., He, S., Liu, J., He, P., Xie, Q., Zheng, Z., & Lyu, M.R. (2019). Tools and Benchmarks for Automated Log Parsing. 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), 121-130.