ROBUST AND TRANSFERABLE LOG ANOMALY DETECTION

ROBUST AND TRANSFERABLE LOG ANOMALY DETECTION

By

SI TONG LIU,

M.A.Sc. (Electrical and Computer Engineering)

A THESIS

SUBMITTED TO THE ELECTRICAL & COMPUTER

ENGINEERING DEPARTMENT

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

McMaster University

Master of Applied Science (2023)

Electrical and Computer Engineering

McMaster University

Hamilton, Ontario, Canada

TITLE:	Robust and Transferable Log Anomaly Detection
AUTHOR:	Si Tong Liu
	M.A.Sc(Electrical and Computer Engineering)
SUPERVISOR:	Jun Chen
	Professor, Department or Electrical and Computer Engineering,
	McMaster University, ON, Canada
NUMBER OF PAGES:	xiii, 52

To my dear parents, who love and support me,

To my dear mentors, who guide and encourage me.

Abstract

Maintaining the stability and performance of software and computer system has always been an ongoing question. As system logs are present in almost all computer systems and software, log anomaly detection has become a major method towards troubleshooting system failures and conducting in-depth analysis to identify the underlying causes. In recent years, even though there has been lots of research done on log anomaly detection, most of them ignored two major problems: the presence of noise when acquiring and processing logs, and the cold-start problem when handling a newly onboarded system. These are two practical problems that usually coexist in a real-world scenario and hasn't been addressed together effectively yet. There were a few works proposed to apply transfer learning in log anomaly detection so that knowledge from a source log dataset can be transferred to a target log dataset, thus alleviating the cold-start problem when handling new systems being onboarded. However, without first solving the noise issue within the log dataset, these methods become impractical in real-world settings. The existence of noise within the source system can greatly impair feature extraction process from the log dataset, leading to a decreased performance of model when detecting anomalies in the target system dataset which might contain noise as well. In this paper, we propose a novel robust and transfer-learning-based method, called LogRT. LogRT utilizes an Attention-based

Bidirectional-Long-Short-Term-Memory model during the feature extraction process to extract the contextual information in log sequences where importance of different log events can be learned by the model automatically even if there are noises present in the dataset. Through combining this robust module with a state-of-the-art transfer learning method, domain adaptation, LogRT can apply the valuable information learned from a noisy source system into a target system and provide high performance on detecting anomalies in logs even with the presence of multiple types of noise in both source and target systems. Extensive experimental evaluations demonstrate that LogRT has competitive performance in real-world scenarios.

Acknowledgements

I would like to extend my heartfelt appreciation to my esteemed supervisor Dr.Chen for his unwavering guidance, expertise, and mentorship during my academic journey here at McMaster University. His profound knowledge, valuable insights, and continuous support have greatly influenced my growth and development as a researcher. It is an immense honor to be his student and I shall embark on my future journey in life with his lessons. I would also like to express my deepest gratitude to my loving parents. Their unconditional love, encouragement, and sacrifices have always been the foundation of my achievements. Their unwavering belief in my abilities and their constant support have motivated me to overcome challenges and pursue my academic goals.

Contents

List of Figures

List of Tables

Notation, Definitions, and Abbreviations

Definitions

Closed-world assumption

Log patterns are always static and won't be modified for a long period of time

Abbreviations

SOTA	State-of-the-Art
LSTM	Long-Short-Term-Memory
Bi-LSTM	Bidirectional Long-Short-Term-Memory
RNN	Recurrent Neural Network
GRL	Gradient Reversal Layer
Deep SVDD	Deep Support Vector Data Description

DeepLog	Anomaly Detection and Diagnosis from System Logs through Deep Learning
LogRobust	Robust Log-Based Anomaly Detection on Unstable Log Data
LogTransfer	Cross-System Log Anomaly Detection for Software Systems with Transfer Learning
LogTAD	Unsupervised Cross-system Log Anomaly Detection via Domain Adaptation

Declaration of Academic Achievement

I, Si Tong Liu, hereby declare that this thesis is my own original work. To the best of my knowledge, this thesis does not contain material previously published by any other party except where due acknowledgement has been provided. No part of this work has been published or submitted for publication or for a higher degree at another institution.

Chapter 1

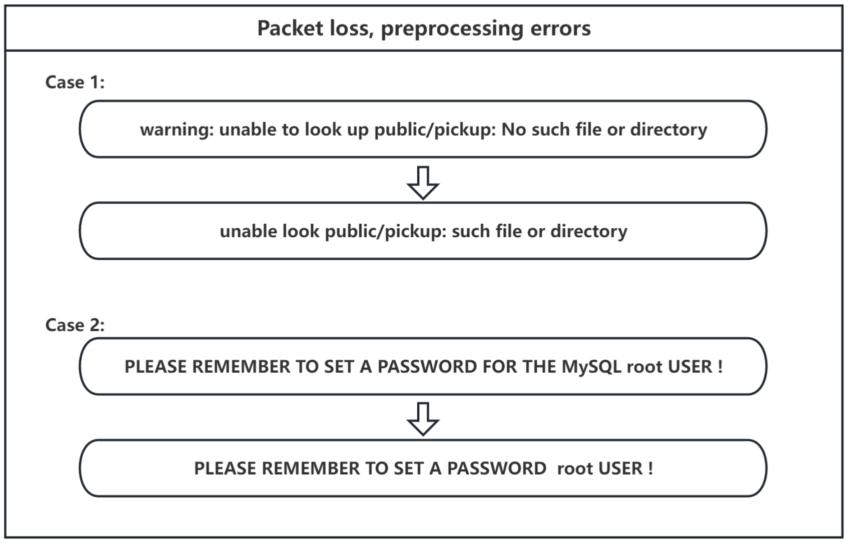
Introduction

With the wide adoption of microservices [1] and cloud technologies, there has been a growing emphasis on the security, stability, and reliability of systems and software. One intuitive approach is to analyze the log data [19, 20, 22, 41, 40] generated by the systems and software during operation time. Log data typically appears in the form of semi-structured text strings and record noteworthy activities about status of systems, thus providing developers with valuable information. In the current stage, a significant number of computer systems and software use log data to provide developers feedback [44, 27] on performance and stability.

However, as the volume of data and the scale of systems continue to grow rapidly, manual observation of logs has become increasingly inefficient and unpractical, with millions of lines of log messages being generated within one system on a daily basis [22, 4]. Therefore, machine learning methods have been introduced to the field of log anomaly detection [14, 42, 25, 8, 28, 29, 26, 43], with the hope of reducing required labor on this task. These machine learning approaches primarily involve identifying shared patterns within normal executing logs and leveraging these patterns to train their models. Subsequently, if the observed patterns in log data deviate from these established normal patterns, the logs will be classified as anomalous.

While machine learning methods have been widely employed to aid in log anomaly detection, there are always new systems being deployed constantly [4, 22]. Despite most of these systems have urgent needs for conducting anomaly detection, it takes a lot of time to gather enough log data so that the neural networks can learn normal executing patterns. Most existing methods do not address this cold-start problem of handling newly onboarded systems. Yet with the widespread strategy of Continuous Integration and Continuous Delivery in the software industry, new applications are being developed and deployed on companies' ecosystems in a much faster rate than they used to [4].

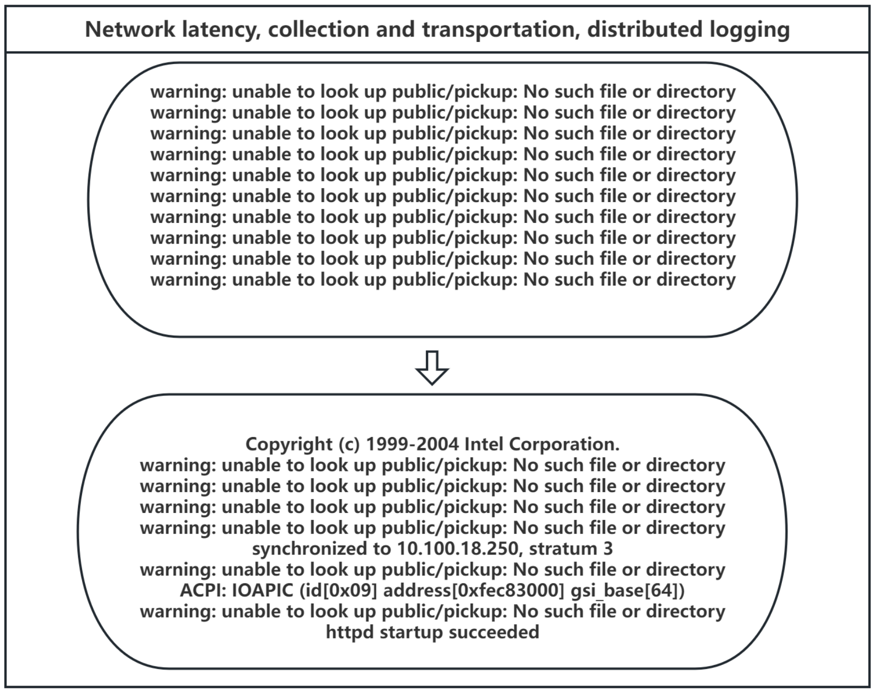
In recent years, some methods for addressing the cold-start problem has been proposed [39, 13, 5], and a few of them have achieved good performance. However, all of those methods are conducted under a closed-world assumption as proposed by Zhang et al. [43]: log patterns are always static and won't be modified for a long period of time. This is unlikely since there are always updates on how log data present themselves, and unstable log data are likely to be gathered in real-world scenarios as it is quite common to introduce noise due to network issues, parsing problems and adversarial attacks. We have demonstrated in Figure 1.1 some of the instability and noise within log datasets that we have simulated and investigated in our work. According to Kabinna, et al [22]. about 20% to 45% of the logging statements they researched on are modified during their lifetime, with 75% of the changes took place with in 145 days after applying those logs to the system. Huo, et al. [19] has also demonstrated how logs can experience massive modifications in their work.



```
warning: unable to look up public/pickup: No such file or directory
unable look public/pickup: such file or directory

PLEASE REMEMBER TO SET A PASSWORD FOR THE MySQL root USER !
PLEASE REMEMBER TO SET A PASSWORD  root USER !
```

(a) Example of log instability after experiencing packet loss and parsing errors


Network latency, collection and transportation, distributed logging

```
warning: unable to look up public/pickup: No such file or directory
warning: unable to look up public/pickup: No such file or directory
warning: unable to look up public/pickup: No such file or directory
warning: unable to look up public/pickup: No such file or directory
warning: unable to look up public/pickup: No such file or directory
warning: unable to look up public/pickup: No such file or directory
warning: unable to look up public/pickup: No such file or directory
warning: unable to look up public/pickup: No such file or directory
warning: unable to look up public/pickup: No such file or directory
warning: unable to look up public/pickup: No such file or directory
```

```
Copyright (c) 1999-2004 Intel Corporation.
warning: unable to look up public/pickup: No such file or directory
warning: unable to look up public/pickup: No such file or directory
warning: unable to look up public/pickup: No such file or directory
warning: unable to look up public/pickup: No such file or directory
synchronized to 10.100.18.250, stratum 3
warning: unable to look up public/pickup: No such file or directory
ACPI: IOAPIC (id[0x09] address[0xfec83000] gsi_base[64])
warning: unable to look up public/pickup: No such file or directory
httpd startup succeeded
```

(b) Example of log instability after experiencing network latency, collection and transportation of logs and distributed logging errors

Figure 1.1: Examples of log instability and noise simulated in our experimental stage

In view of the aforementioned issues, our study adopts an Attention-based Bi-LSTM architecture inspired by Zhang et al. [43] to cope with the presence of updated log statement and noise among log data. Our neural network structure enables the network to focus more effectively on log data with important information, thus enhancing the performance of log anomaly detection even within noisy environments. Then, we employ a domain adaptation approach proposed by Han et al. [13] to address the cold-start problem. By utilizing this approach, we can train our network on an existing labelled source log dataset and then apply it to new unlabelled target log datasets, thereby overcoming the challenge of performing log anomaly detection in the absence of labelled log data for new systems.

We have conducted our study in a cross-dataset setting and introduced noise into log data proportionally to simulate real-world settings. Our network exhibited robust and competitive performance.

The main contributions of this paper can be summarized as the following three points:

- We propose a novel approach, LogRT, which can address the noise issue during log anomaly detection in a cross-system scenario. Our model can effectively focus on important information within existing log data regardless of the presence of noise from the source system, then transfer valuable information into the target system, thus yield competitive performance in log anomaly detection.
- We propose noise-controlled log datasets based on the original Thunderbid and BlueGene/L log datasets, where they contain certain ratios of noise in different forms. We will elaborate on the algorithm of simulating these noise in Chapter 3.

- We perform thorough experiments on multiple log datasets with simulated noise under the cross-system environment to demonstrate the effectiveness of our network. The results show that our network achieves competitive performance in real-world scenarios.

The structure of this thesis is as follows: In Chapter 2, we introduce the research background and discuss works that are closely related to the topic under consideration. In Chapter 3, we present the framework of our network and elaborate on the approaches we apply in this work. Chapter 4 contains our experimental results under simulated real-world scenarios and cross-system settings. Future improvements on our method are discussed in Chapter 5, and conclusion for this study is in Chapter 6.

Chapter 2

Research Background and Related works

2.1 Log Anomaly Detection Preliminaries

In this section, we will give an overview regarding the workflow of log anomaly detection and discuss each process in depth. A typical log anomaly detection procedure is shown in Figure 2.1:

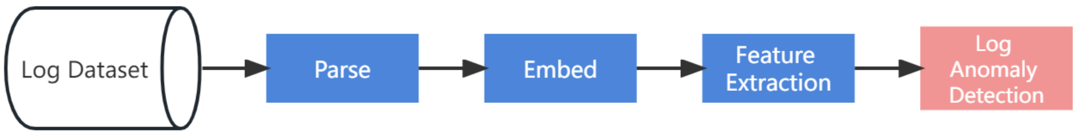


Figure 2.1: Flowchart containing the major processes when performing log anomaly detection, including log parsing, log embedding, feature extraction and anomaly detection

2.1.1 Log Parsing

In order to perform log anomaly detection, our network model must first learn the normal executing log patterns from the log dataset. Subsequently, we need to extract not only valid but also effective information from log datasets before proceeding into further steps. Log data contains comprehensive records on the operation of software and systems, including timestamps, runtime, and most importantly EventIDs. Despite of their crucial role in the task, log datasets don't usually come in a well-structured fashion. Most of the logs are generated as chunks of words that contain valuable information. Without structuralizing them, these valuable information will remain inaccessible to any form of machine learning techniques. As a result, log parsing [15, 7, 14, 45, 23, 6] has become an important step in log anomaly detection with its ability to structuralize chaotic log data into well-organized templates and parameters. There are two predominant methods for parsing log data, Spell [7] and Drain [15].

2.1.2 Log Sequence Representation

With structured log datasets, we look for trends within these datasets so normal executing log patterns can be inferred and later be learned by our deep learning model. However, most of the log data are not stand-alone message lines. For example, when developing a software in microservice architecture, a downtime on a single service would not only induce logs recording the downtime of the service itself, but also multiple failed requests from other services that are connected to it. Thus, it is crucial for logs to be grouped in a way such that contextual information can be captured and packaged for the deep learning model to extract features.

Since machine learning models cannot extract features directly from log data in the form of texts and numbers, we must transform them into vectors. This is equivalent to the word embedding step [30, 32, 3, 21] in natural language processing.

2.1.3 Anomaly Detection

There have been many machine learning methods [27, 42] trying to address the log anomaly detection problem. They can be divided into two major categories: Reconstruction of Normal Logs and Log Patterns Invariance Mining. While these methods are effective and showing good performance, deep learning [43, 8, 5, 23, 26, 29, 28] has become the driving force behind the continually-improved performance of log anomaly detection.

One thing to note is that log anomaly detection poses a unique challenge due to its nature: As systems and software generate an enormous amount of log data on a daily basis, it is very expensive to have them all labelled and classified. Since most of the anomalous log data are generated by unreproducible scenarios, such as a fatal disruption to a running server or sudden power outage during system operation, we typically don't have labelled anomalous log data when performing log anomaly detection. As a result, most works in the field of log anomaly detection is based on unsupervised learning.

2.2 Empirical methods on Log Anomaly Detection

In this section, we will go through some foundational methods in the field of log anomaly detection that are closely related to the topics of our work.

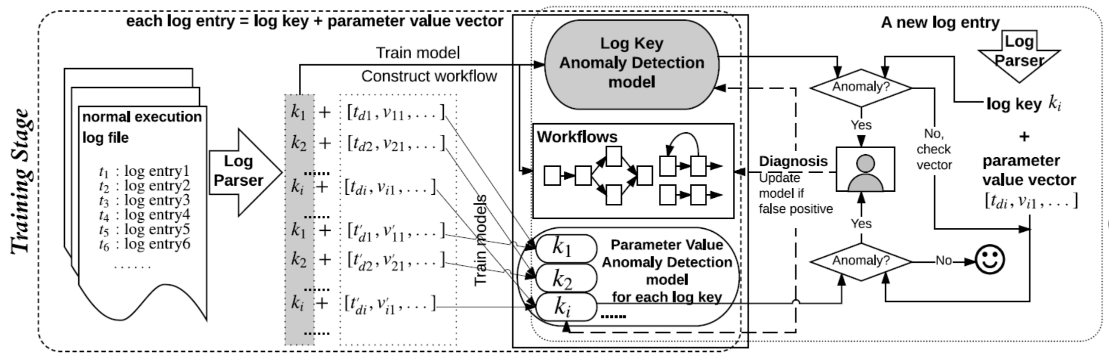
Four representative works are discussed in this section. The first work is DeepLog [8], which is one of the first approaches that brings deep learning to bear upon the subject of log anomaly detection. The second work LogRobust [43] tries to address the noise issue within log datasets, while the third and fourth works LogTransfer [5] and LogTAD [13] attempt to resolve the cross-system log anomaly detection problem.

Table 2.1 lists the aforementioned methods [8, 43, 5, 13]. In addition, we also point out in this table the specific issues these works are trying to address:

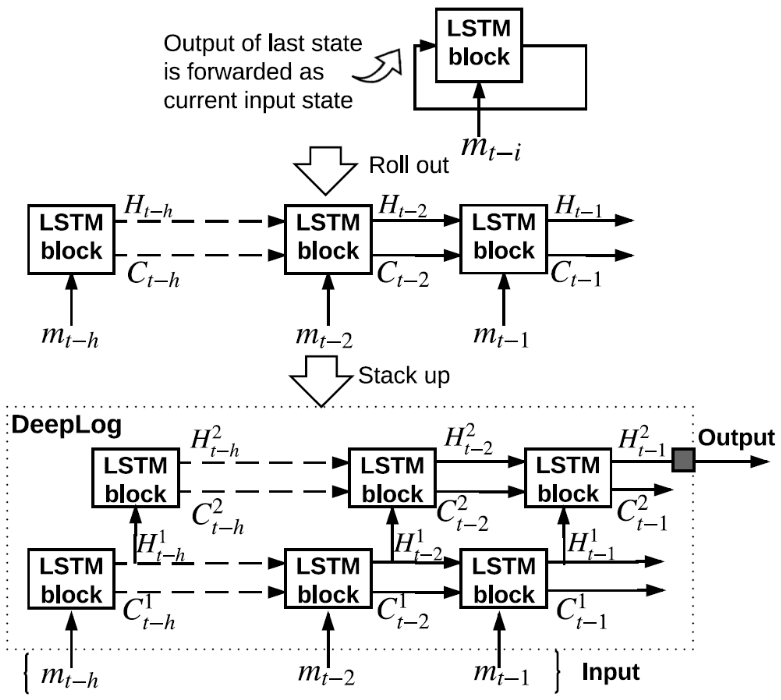
Name	**Noise and Instability**	**Cross-system**
DeepLog	No	No
LogRobust	Yes	No
LogTransfer	No	Yes
LogTAD	No	Yes

Table 2.1: Summary of empirical methods DeepLog [8], LogRobust [43], LogTransfer [5] and LogTAD [13]. The two columns on the right are the topics our work relates to and whether they have been resolved by previous works.

DeepLog: Du et al.'s [8] work on log anomaly detection was one of the earliest attempts in introducing natural language processing and deep learning into the field of log anomaly detection. They proposed using the prevalent technique in natural language processing, Long-Short-Term-Memory(LSTM), to capture underlying normal executing log patterns. After learning this underlying pattern, their model then ranks the log events in a top-k likelihood depending on whether they follow the existing log events. If the incoming log event lies outside of these top-k candidates, the model would raise alert and label this new event as an anomaly. Figures 2.2a and 2.2b demonstrate the structure of the DeepLog model and the LSTM mechanism within their model.



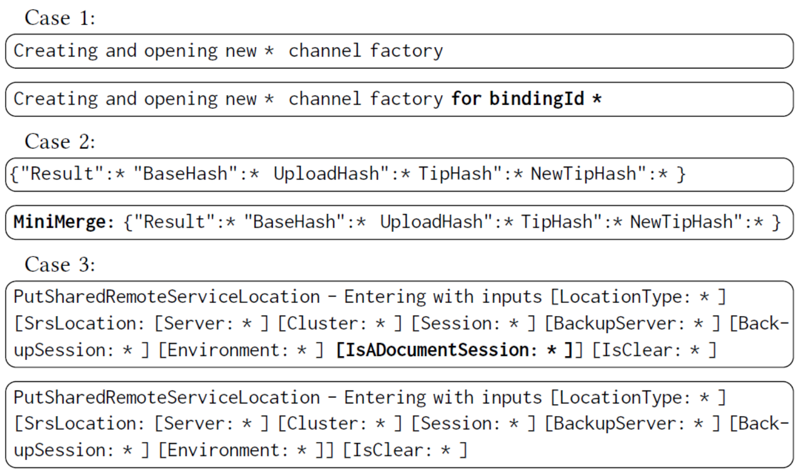
(a) DeepLog (images sourced from [8]) structure utilizing LSTM module



(b) LSTM model (images sourced from [8]) within the DeepLog structure

Figure 2.2: Demonstration of DeepLog structure (images sourced from [8]) and LSTM mechanism applied

LogRobust: Introduced by Zhang et al. [5], LogRobust is one of the first models that effectively addresses the issue of log data instability in log anomaly detection. This work highlights noise and instability of log datasets as a major obstacle when performing log anomaly detection in real-world settings, which in fact has not yet received enough attention. As demonstrated in Figures 2.3a and 2.3b, reference [5] investigated multiple sources of log instability and discussed them in depth.



Case 1:

```
Creating and opening new * channel factory
Creating and opening new * channel factory for bindingId *
```

Case 2:

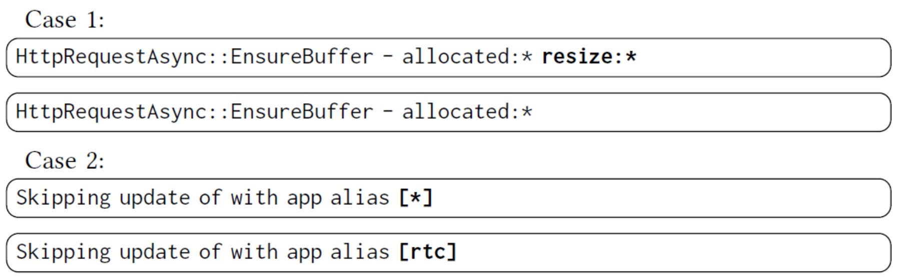
```
{"Result":* "BaseHash":* UploadHash":* TipHash":* NewTipHash":* }
MiniMerge: {"Result":* "BaseHash":* UploadHash":* TipHash":* NewTipHash":* }
```

Case 3:

```
PutSharedRemoteServiceLocation - Entering with inputs [LocationType: * ]
[SrsLocation: [Server: * ] [Cluster: * ] [Session: * ] [BackupServer: * ] [Back-
upSession: * ] [Environment: * ] [IsADocumentSession: * ]] [IsClear: * ]

PutSharedRemoteServiceLocation - Entering with inputs [LocationType: * ]
[SrsLocation: [Server: * ] [Cluster: * ] [Session: * ] [BackupServer: * ] [Back-
upSession: * ] [Environment: * ]] [IsClear: * ]
```

(a) Demonstration of how logs tend to be updated during their life cycle (images sourced from [5]).



Case 1:

```
HttpRequestAsync::EnsureBuffer - allocated:* resize:*
HttpRequestAsync::EnsureBuffer - allocated:*
```

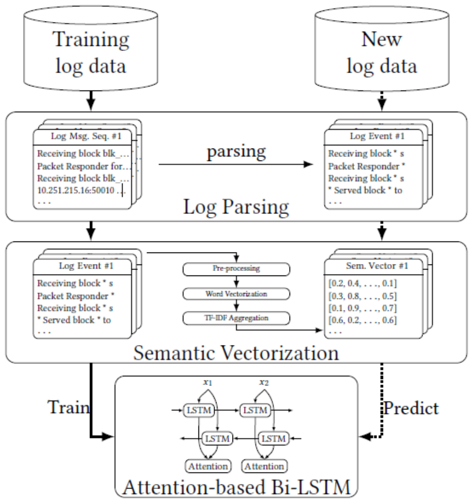
Case 2:

```
Skipping update of with app alias [*]
Skipping update of with app alias [rtc]
```

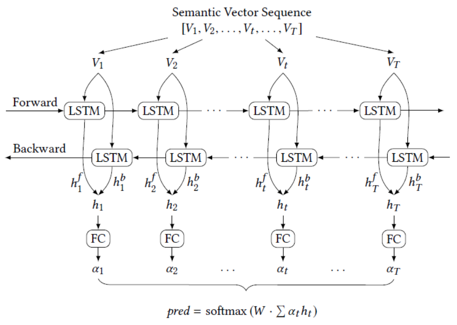
(b) Demonstration of the parsing errors that could occur during log processing (images sourced from [5]).

Figure 2.3: Noise and instability investigated in LogRobust (images sourced from [5])

With the help of the FastText [21] algorithm, LogRobust pre-trains their word embedding model and inputs word vectors into the deep learning network. It is one of the few models that explicitly address the noise and instability in log datasets. Specifically, utilizing an Attention-based Bi-LSTM as shown in Figures 2.4a and 2.4b, the model has the ability to assign different weights to individual log sequences, thus prioritizing valuable log sequences that contain important information over ones that mainly contain noise. This measure effectively mitigates the impact of noise in log data on the log anomaly detection task.



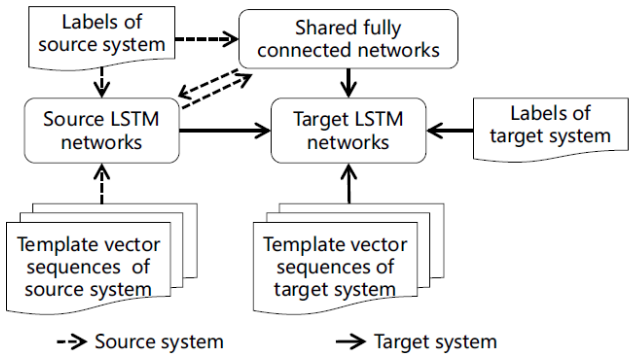
(a) LogRobust structure utilizing Attention-based Bi-LSTM (images sourced from [5])


Semantic Vector Sequence
$[V_1, V_2, \ldots, V_t, \ldots, V_T]$
Forward
Backward
LSTM
FC
$pred = \text{softmax}(W \cdot \sum \alpha_t h_t)$

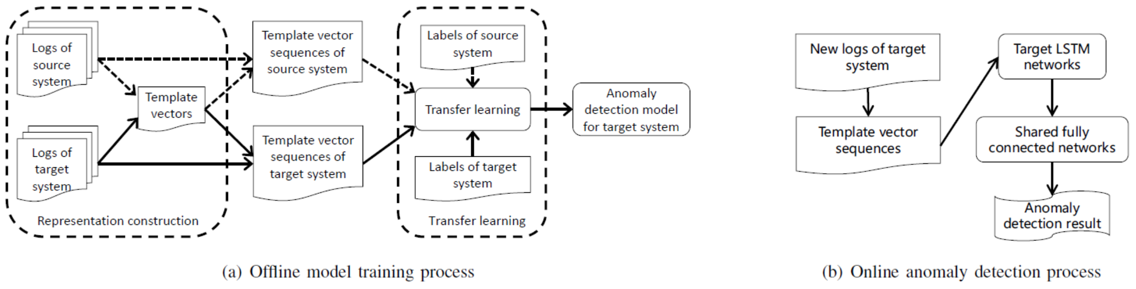
(b) Demonstration of the Attention-Based Bi-LSTM mechanism in LogRobust structure (images sourced from [5])

Figure 2.4: LogRobust structure and Attention-based Bi-LSTM module applied (images sourced from [5])

LogTransfer: LogTransfer [5], proposed by Chen et al., presents a novel solution in transferring execution patterns of log events from a source system to a target system using transfer learning. According to Figure 2.5b, the LogTransfer model is composed of two key components: construction of representation and knowledge transferring between source and target systems. In order to construct the representation of log events, a robust word embedding method, Glove [32], is employed to convert log messages from words into vectors which can be processed by the deep learning model. Then as demonstrated in Figure 2.5a, by training the Long-Short-Term-Memory(LSTM) network [18] together with fully connected layers on normal and anomaly log events obtained from the source system, the LogTransfer model can capture the patterns needed to discriminate normal and anomalous logs in the source system. Then, through fine-tuning on log sequences from the target system and post-processing by the fully connected layers, the LSTM network eventually gains the ability to detect log anomalies in the target system. Through these two steps, LogTransfer can effectively transfer the knowledge on log events from a source system to a target system, enabling high accuracy in log anomaly detection in cross-system settings.


Labels of source system
Shared fully connected networks
Source LSTM networks
Target LSTM networks
Labels of target system
Template vector sequences of source system
Template vector sequences of target system
Source system
Target system

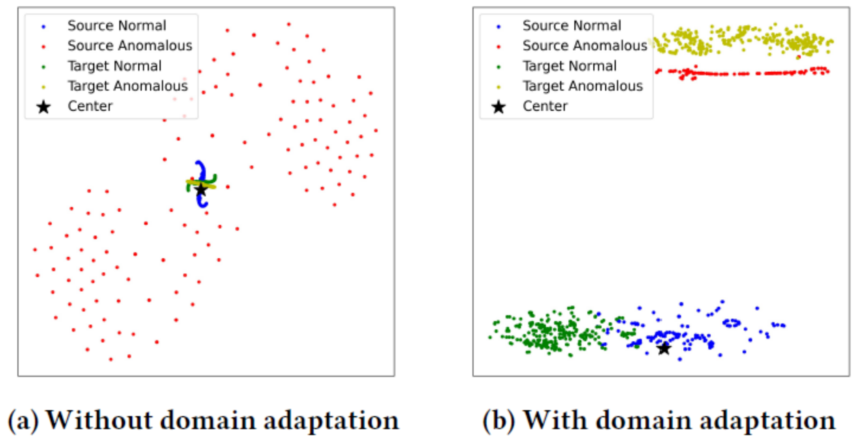
(a) Workflow of the transfer learning component in LogTransfer (Image sourced from [5]).


Logs of source system
Logs of target system
Template vectors
Representation construction
Template vector sequences of source system
Template vector sequences of target system
Labels of source system
Transfer learning
Labels of target system
Transfer learning
Anomaly detection model for target system
(a) Offline model training process
New logs of target system
Template vector sequences
Target LSTM networks
Shared fully connected networks
Anomaly detection result
(b) Online anomaly detection process

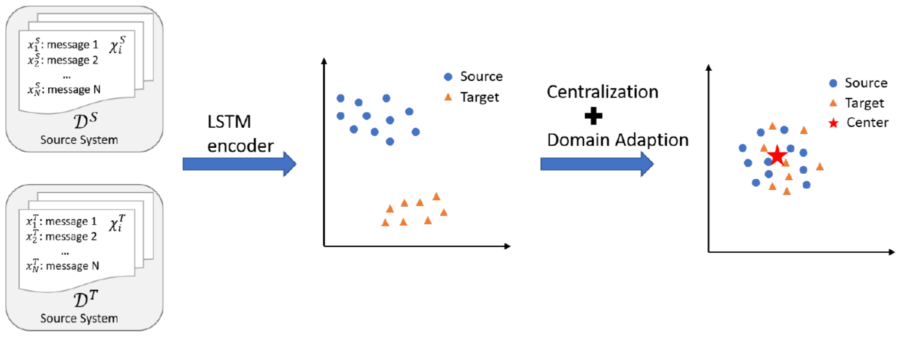
(b) LogTransfer structure (Image sourced from [5]).

Figure 2.5: LogTransfer structure and transfer learning flow (Images sourced from [5])

LogTAD: Han et al. introduced LogTAD [13] as a cross-system log anomaly detection approach, which is enabled by domain adaptation, an important subfield of transfer learning. With a unique domain adaptation structure as shown in Figure 2.6b, LogTAD only requires the normal executing logs from source system and a small amount of normal executing logs from target system for the model to understand how normal patterns of log events are like in both source and target systems. LogTAD leverages a word2vec model [30] during the log embedding process followed by a LSTM model in the feature extraction process. Domain adversarial training [10, 9], as demonstrated in Figure 2.6a, is then used to create a generator-discriminator mechanism where the model's prediction will serve as generated data while ground-truth data will serve as real data so that data from both source and target domains cannot be distinguished by the discriminator. The trained model is then capable of performing log anomaly detection on both the source and target systems. LogTAD has significantly reduced the need for labelled anomalous log data, which is challenging to obtain in real-world scenarios, and thus lowering the barriers on data requirements for log anomaly detection.



(a) Demonstration of data distribution after domain adversarial training in LogTAD structure (Image sourced from [13])



(b) LogTAD workflow (Image sourced from [13])

Figure 2.6: LogTAD's domain adversarial training mechanism and workflow (Images sourced from [13])

Chapter 3

Research Methodologies

In this chapter, we will introduce the framework of LogRT and take a deep dive into how each component in our network contributes to the competitive performance in the experimental stage. We will discuss how the application of Attention-based Bi-LSTM mechanism contributes to an improved performance on the task of log anomaly detection in a cross-system setting.

3.1 Overall Framework

Our work aims to resolve the challenge of instability and presence of noise in log datasets during feature extraction phase when performing log anomaly detection in a cross-system scenario.

We first perform feature extraction on normal execution logs with an Attention-based Bi-LSTM module inspired by LogRobust [43], which was applied in single system log anomaly detection. Through this Attention-based Bi-LSTM module, we can direct more attention to the log sequences which contain valuable information of underlying execution patterns of the log datasets instead of noisy log sequences that contain irrelevant information. Consequently, our model has the ability to reduce the impact of noise and instability in log datasets and extract features in a robust manner. Then, we leverage a domain adversarial training approach, first proposed by Han et al. [13], to map log data points from both source and target datasets into the same domain, thus enabling our model to operate in a cross-system setting. After the feature extraction and transfer learning phases, we empower our network's ability in performing anomaly detection through utilizing the Deep Support Vector Data Description [33] approach. Figure 3.1 provides an overview of our network model.

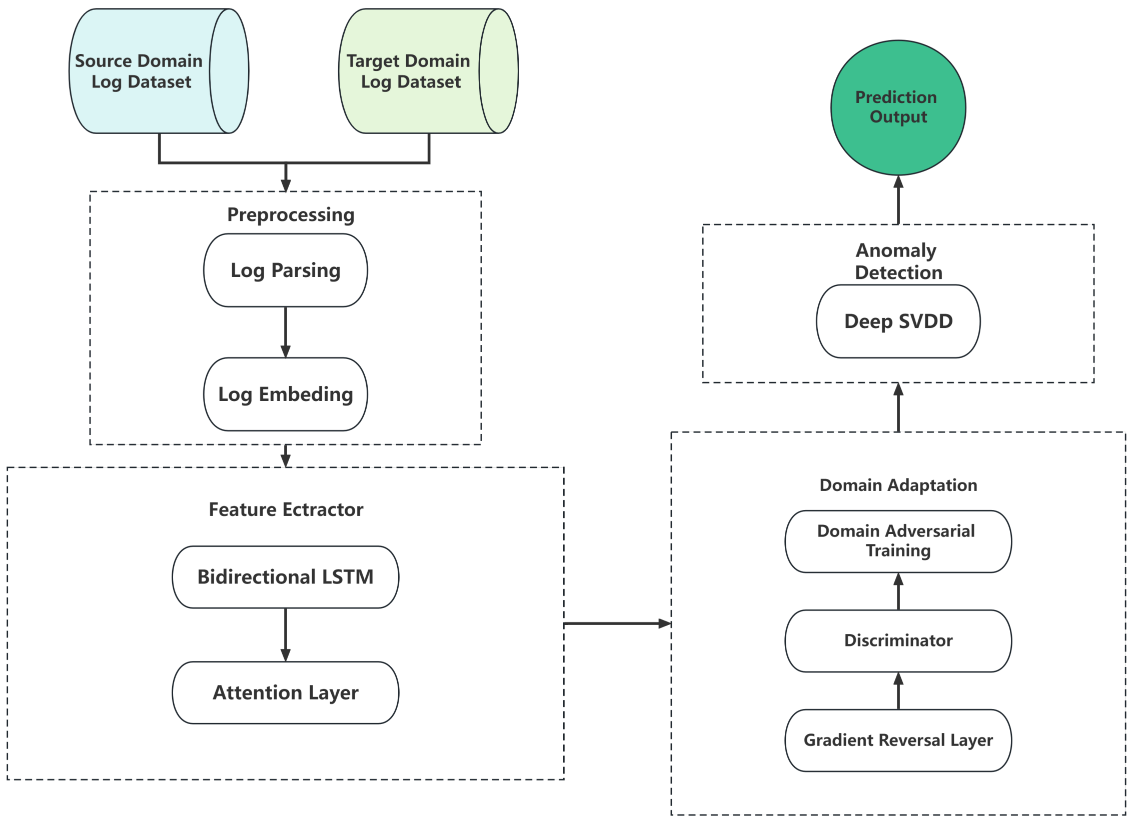


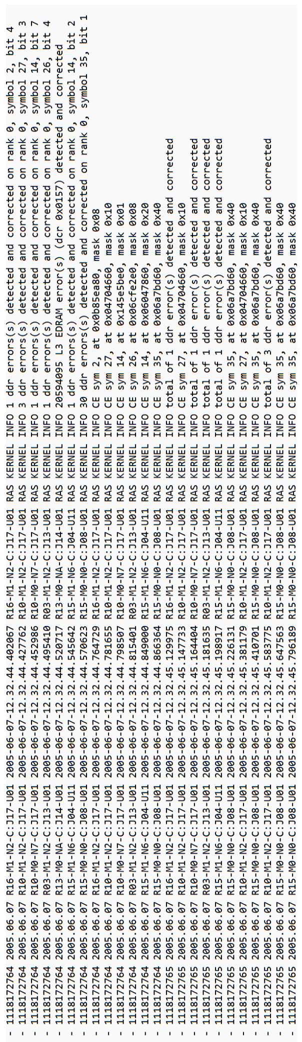
Figure 3.1: LogRT Architecture

3.2 Log Parsing and Log Representation

To begin with, we process the log datasets and embed them into word vectors. In this work, we apply Drain to parse our raw logs into structured log lines and word2vec [30] for log embedding.

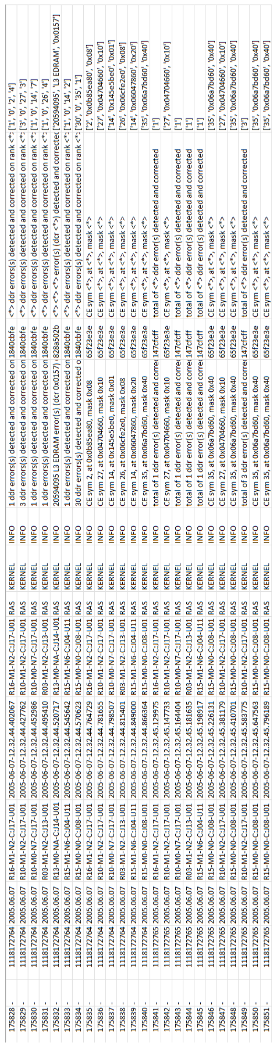
Log Parsing and Dimension Reduction: The features we want to extract from log datasets are those that carry the information regarding normal executing patterns of systems and software so the model can learn and distinguish anomalous log sequences if the input pattern starts to deviate. Figure 3.2 provides a comparison of our dataset and the original logs after using Drain [15]. Columns of parsed log from left to right are LineId, Label, Timestamp, Date, Node, Time, NodeRepeat, Type, Component, Level, Content, EventId, EventTemplate, and ParameterList. As demonstrated, the raw chunk of logs have been parsed into well-structured log sequences.

We use EventTemplate to extract information from each line of log events. As shown in Figure 3.2, there is also a column named Content, which is similar to EventTemplate and contains the specific parameters for each line of log. The reason we don't use Content to conduct feature extraction is because the information in Content is noisy and volatile with more dimensions than we need. For example, if we compare the same software when it has only one user to a scenario where it is globally deployed with millions of users, their log data might share the same pattern yet the parameters within them will differ greatly. The model will have to understand that even though the software's log under these two scenarios have a huge difference in parameters, they are essentially the same pattern-wise. Thus, compared to using EventTemplate, using the Content column of log dataset would only increase the dimension and complexity of the data input to our model while adding little benefit to our model's learning process.



```
- 1118172764 2005.06.07 R16-M1-N2-C:J17-U01 2005-06-07-12.32.44.402067 R16-M1-N2-C:J17-U01 RAS KERNEL INFO 1 ddr errors(s) detected and corrected on rank 0, symbol 2, bit 4
- 1118172764 2005.06.07 R10-M1-N2-C:J17-U01 2005-06-07-12.32.44.427762 R10-M1-N2-C:J17-U01 RAS KERNEL INFO 3 ddr errors(s) detected and corrected on rank 0, symbol 27, bit 3
- 1118172764 2005.06.07 R10-M0-N7-C:J17-U01 2005-06-07-12.32.44.452986 R10-M0-N7-C:J17-U01 RAS KERNEL INFO 1 ddr errors(s) detected and corrected on rank 0, symbol 14, bit 7
- 1118172764 2005.06.07 R03-M1-N2-C:J13-U01 2005-06-07-12.32.44.495410 R03-M1-N2-C:J13-U01 RAS KERNEL INFO 1 ddr errors(s) detected and corrected on rank 0, symbol 26, bit 4
- 1118172764 2005.06.07 R13-M0-NA-C:J14-U01 2005-06-07-12.32.44.520717 R13-M0-NA-C:J14-U01 RAS KERNEL INFO 20594095 L3 EDRAM error(s) (dcr 0x0157) detected and corrected
- 1118172764 2005.06.07 R15-M1-N6-C:J04-U11 2005-06-07-12.32.44.545642 R15-M1-N6-C:J04-U11 RAS KERNEL INFO 1 ddr errors(s) detected and corrected on rank 0, symbol 14, bit 2
- 1118172764 2005.06.07 R15-M0-N0-C:J08-U01 2005-06-07-12.32.44.570623 R15-M0-N0-C:J08-U01 RAS KERNEL INFO 30 ddr errors(s) detected and corrected on rank 0, symbol 35, bit 1
- 1118172764 2005.06.07 R16-M1-N2-C:J17-U01 2005-06-07-12.32.44.764729 R16-M1-N2-C:J17-U01 RAS KERNEL INFO CE sym 2, at 0x0b85ea80, mask 0x08
- 1118172764 2005.06.07 R10-M1-N2-C:J17-U01 2005-06-07-12.32.44.781655 R10-M1-N2-C:J17-U01 RAS KERNEL INFO CE sym 27, at 0x04704660, mask 0x10
- 1118172764 2005.06.07 R10-M0-N7-C:J17-U01 2005-06-07-12.32.44.798507 R10-M0-N7-C:J17-U01 RAS KERNEL INFO CE sym 14, at 0x145e5be0, mask 0x01
- 1118172764 2005.06.07 R03-M1-N2-C:J13-U01 2005-06-07-12.32.44.815401 R03-M1-N2-C:J13-U01 RAS KERNEL INFO CE sym 26, at 0x06cfe2e0, mask 0x08
- 1118172764 2005.06.07 R15-M1-N6-C:J04-U11 2005-06-07-12.32.44.849000 R15-M1-N6-C:J04-U11 RAS KERNEL INFO CE sym 14, at 0x06047860, mask 0x20
- 1118172764 2005.06.07 R15-M0-N0-C:J08-U01 2005-06-07-12.32.44.866364 R15-M0-N0-C:J08-U01 RAS KERNEL INFO CE sym 35, at 0x06a7bd60, mask 0x40
- 1118172765 2005.06.07 R16-M1-N2-C:J17-U01 2005-06-07-12.32.45.129975 R16-M1-N2-C:J17-U01 RAS KERNEL INFO total of 1 ddr error(s) detected and corrected
- 1118172765 2005.06.07 R10-M1-N2-C:J17-U01 2005-06-07-12.32.45.147733 R10-M1-N2-C:J17-U01 RAS KERNEL INFO CE sym 27, at 0x04704660, mask 0x10
- 1118172765 2005.06.07 R10-M0-N7-C:J17-U01 2005-06-07-12.32.45.164404 R10-M0-N7-C:J17-U01 RAS KERNEL INFO total of 1 ddr error(s) detected and corrected
- 1118172765 2005.06.07 R03-M1-N2-C:J13-U01 2005-06-07-12.32.45.181635 R03-M1-N2-C:J13-U01 RAS KERNEL INFO total of 1 ddr error(s) detected and corrected
- 1118172765 2005.06.07 R15-M1-N6-C:J04-U11 2005-06-07-12.32.45.198917 R15-M1-N6-C:J04-U11 RAS KERNEL INFO total of 1 ddr error(s) detected and corrected
- 1118172765 2005.06.07 R15-M0-N0-C:J08-U01 2005-06-07-12.32.45.226131 R15-M0-N0-C:J08-U01 RAS KERNEL INFO CE sym 35, at 0x06a7bd60, mask 0x40
- 1118172765 2005.06.07 R10-M1-N2-C:J17-U01 2005-06-07-12.32.45.381179 R10-M1-N2-C:J17-U01 RAS KERNEL INFO CE sym 27, at 0x04704660, mask 0x10
- 1118172765 2005.06.07 R15-M0-N0-C:J08-U01 2005-06-07-12.32.45.410701 R15-M0-N0-C:J08-U01 RAS KERNEL INFO CE sym 35, at 0x06a7bd60, mask 0x40
- 1118172765 2005.06.07 R10-M1-N2-C:J17-U01 2005-06-07-12.32.45.583775 R10-M1-N2-C:J17-U01 RAS KERNEL INFO total of 3 ddr error(s) detected and corrected
- 1118172765 2005.06.07 R15-M0-N0-C:J08-U01 2005-06-07-12.32.45.647563 R15-M0-N0-C:J08-U01 RAS KERNEL INFO CE sym 35, at 0x06a7bd60, mask 0x40
- 1118172765 2005.06.07 R15-M0-N0-C:J08-U01 2005-06-07-12.32.45.796189 R15-M0-N0-C:J08-U01 RAS KERNEL INFO CE sym 35, at 0x06a7bd60, mask 0x40
```

(a) Caption of BlueGene/L [24] log dataset before parsing.



175828	-	1118172764	2005.06.07	R16-M1-N2-C:J17-U01	2005-06-07-12.32.44.402067	R16-M1-N2-C:J17-U01	RAS	KERNEL	INFO	1 ddr errors(s) detected and corrected on	1840cbfe	<*> ddr errors(s) detected and corrected on rank <*:	['1', '0', '2', '4']
175829	-	1118172764	2005.06.07	R10-M1-N2-C:J17-U01	2005-06-07-12.32.44.427762	R10-M1-N2-C:J17-U01	RAS	KERNEL	INFO	3 ddr errors(s) detected and corrected on	1840cbfe	<*> ddr errors(s) detected and corrected on rank <*:	['3', '0', '27', '3']
175830	-	1118172764	2005.06.07	R10-M0-N7-C:J17-U01	2005-06-07-12.32.44.452986	R10-M0-N7-C:J17-U01	RAS	KERNEL	INFO	1 ddr errors(s) detected and corrected on	1840cbfe	<*> ddr errors(s) detected and corrected on rank <*:	['1', '0', '14', '7']
175831	-	1118172764	2005.06.07	R03-M1-N2-C:J13-U01	2005-06-07-12.32.44.495410	R03-M1-N2-C:J13-U01	RAS	KERNEL	INFO	1 ddr errors(s) detected and corrected on	1840cbfe	<*> ddr errors(s) detected and corrected on rank <*:	['1', '0', '26', '4']
175832	-	1118172764	2005.06.07	R13-M0-NA-C:J14-U01	2005-06-07-12.32.44.520717	R13-M0-NA-C:J14-U01	RAS	KERNEL	INFO	20594095 L3 EDRAM error(s) (dcr 0x0157) (	828a502b	<*> <*> <*> error(s) (dcr <*>) detected and correctec	['20594095', 'L3 EDRAM', '0x0157']
175833	-	1118172764	2005.06.07	R15-M1-N6-C:J04-U11	2005-06-07-12.32.44.545642	R15-M1-N6-C:J04-U11	RAS	KERNEL	INFO	1 ddr errors(s) detected and corrected on	1840cbfe	<*> ddr errors(s) detected and corrected on rank <*:	['1', '0', '14', '2']
175834	-	1118172764	2005.06.07	R15-M0-N0-C:J08-U01	2005-06-07-12.32.44.570623	R15-M0-N0-C:J08-U01	RAS	KERNEL	INFO	30 ddr errors(s) detected and corrected o	1840cbfe	<*> ddr errors(s) detected and corrected on rank <*:	['30', '0', '35', '1']
175835	-	1118172764	2005.06.07	R16-M1-N2-C:J17-U01	2005-06-07-12.32.44.764729	R16-M1-N2-C:J17-U01	RAS	KERNEL	INFO	CE sym 2, at 0x0b85ea80, mask 0x08	65f23e3e	CE sym <*>, at <*>, mask <*>	['2', '0x0b85ea80', '0x08']
175836	-	1118172764	2005.06.07	R10-M1-N2-C:J17-U01	2005-06-07-12.32.44.781655	R10-M1-N2-C:J17-U01	RAS	KERNEL	INFO	CE sym 27, at 0x04704660, mask 0x10	65f23e3e	CE sym <*>, at <*>, mask <*>	['27', '0x04704660', '0x10']
175837	-	1118172764	2005.06.07	R10-M0-N7-C:J17-U01	2005-06-07-12.32.44.798507	R10-M0-N7-C:J17-U01	RAS	KERNEL	INFO	CE sym 14, at 0x145e5be0, mask 0x01	65f23e3e	CE sym <*>, at <*>, mask <*>	['14', '0x145e5be0', '0x01']
175838	-	1118172764	2005.06.07	R03-M1-N2-C:J13-U01	2005-06-07-12.32.44.815401	R03-M1-N2-C:J13-U01	RAS	KERNEL	INFO	CE sym 26, at 0x06cfe2e0, mask 0x08	65f23e3e	CE sym <*>, at <*>, mask <*>	['26', '0x06cfe2e0', '0x08']
175839	-	1118172764	2005.06.07	R15-M1-N6-C:J04-U11	2005-06-07-12.32.44.849000	R15-M1-N6-C:J04-U11	RAS	KERNEL	INFO	CE sym 14, at 0x06047860, mask 0x20	65f23e3e	CE sym <*>, at <*>, mask <*>	['14', '0x06047860', '0x20']
175840	-	1118172764	2005.06.07	R15-M0-N0-C:J08-U01	2005-06-07-12.32.44.866364	R15-M0-N0-C:J08-U01	RAS	KERNEL	INFO	CE sym 35, at 0x06a7bd60, mask 0x40	65f23e3e	CE sym <*>, at <*>, mask <*>	['35', '0x06a7bd60', '0x40']
175841	-	1118172765	2005.06.07	R16-M1-N2-C:J17-U01	2005-06-07-12.32.45.129975	R16-M1-N2-C:J17-U01	RAS	KERNEL	INFO	total of 1 ddr error(s) detected and correc	147cfcff	total of <*> ddr error(s) detected and corrected	['1']
175842	-	1118172765	2005.06.07	R10-M1-N2-C:J17-U01	2005-06-07-12.32.45.147733	R10-M1-N2-C:J17-U01	RAS	KERNEL	INFO	CE sym 27, at 0x04704660, mask 0x10	65f23e3e	CE sym <*>, at <*>, mask <*>	['27', '0x04704660', '0x10']
175843	-	1118172765	2005.06.07	R10-M0-N7-C:J17-U01	2005-06-07-12.32.45.164404	R10-M0-N7-C:J17-U01	RAS	KERNEL	INFO	total of 1 ddr error(s) detected and correc	147cfcff	total of <*> ddr error(s) detected and corrected	['1']
175844	-	1118172765	2005.06.07	R03-M1-N2-C:J13-U01	2005-06-07-12.32.45.181635	R03-M1-N2-C:J13-U01	RAS	KERNEL	INFO	total of 1 ddr error(s) detected and correc	147cfcff	total of <*> ddr error(s) detected and corrected	['1']
175845	-	1118172765	2005.06.07	R15-M1-N6-C:J04-U11	2005-06-07-12.32.45.198917	R15-M1-N6-C:J04-U11	RAS	KERNEL	INFO	total of 1 ddr error(s) detected and correc	147cfcff	total of <*> ddr error(s) detected and corrected	['1']
175846	-	1118172765	2005.06.07	R15-M0-N0-C:J08-U01	2005-06-07-12.32.45.226131	R15-M0-N0-C:J08-U01	RAS	KERNEL	INFO	CE sym 35, at 0x06a7bd60, mask 0x40	65f23e3e	CE sym <*>, at <*>, mask <*>	['35', '0x06a7bd60', '0x40']
175847	-	1118172765	2005.06.07	R10-M1-N2-C:J17-U01	2005-06-07-12.32.45.381179	R10-M1-N2-C:J17-U01	RAS	KERNEL	INFO	CE sym 27, at 0x04704660, mask 0x10	65f23e3e	CE sym <*>, at <*>, mask <*>	['27', '0x04704660', '0x10']
175848	-	1118172765	2005.06.07	R15-M0-N0-C:J08-U01	2005-06-07-12.32.45.410701	R15-M0-N0-C:J08-U01	RAS	KERNEL	INFO	CE sym 35, at 0x06a7bd60, mask 0x40	65f23e3e	CE sym <*>, at <*>, mask <*>	['35', '0x06a7bd60', '0x40']
175849	-	1118172765	2005.06.07	R10-M1-N2-C:J17-U01	2005-06-07-12.32.45.583775	R10-M1-N2-C:J17-U01	RAS	KERNEL	INFO	total of 3 ddr error(s) detected and correc	147cfcff	total of <*> ddr error(s) detected and corrected	['3']
175850	-	1118172765	2005.06.07	R15-M0-N0-C:J08-U01	2005-06-07-12.32.45.647563	R15-M0-N0-C:J08-U01	RAS	KERNEL	INFO	CE sym 35, at 0x06a7bd60, mask 0x40	65f23e3e	CE sym <*>, at <*>, mask <*>	['35', '0x06a7bd60', '0x40']
175851	-	1118172765	2005.06.07	R15-M0-N0-C:J08-U01	2005-06-07-12.32.45.796189	R15-M0-N0-C:J08-U01	RAS	KERNEL	INFO	CE sym 35, at 0x06a7bd60, mask 0x40	65f23e3e	CE sym <*>, at <*>, mask <*>	['35', '0x06a7bd60', '0x40']

(b) Caption of BlueGene/L [24] log dataset after parsing.

Figure 3.2: Demonstration of our BlueGene/L [24] log dataset after utilizing Drain [15] as log parser

Log Embedding: With the logs being parsed into structured datasets, we embed them into word vectors utilizing word2vec [30]. This is a crucial step in which we transform log dataset into the form that semantic and contextual information can be captured by the LogRT model to yield prediction results.

3.3 Noise Simulation

Since we want our network to be applicable in real-world scenarios, after researching on different types of noise that tend to exist in log dataset, we choose to simulate three types of noise within our datasets which are also investigated in [43].

Table 3.1: Source of log instability and noise in real-world environments and our equivalent simulations

Sources from real-world scenarios	**Simulations**
Packet loss, updated log templates, pre-processing errors	Removal of words within log lines
Packet loss, updated log templates, pre-processing errors	Removal of log lines within log sequence
Network latency, collection and transportation, distributed logging	Shuffle log lines of different timestamps

In order for the noise to be as realistic as possible, we divide the log datasets into windows with certain length, where we modify a specific percentage of the log events in the window to inject noise under our desired percentages. Below is the pseudo code for the noise construction functions.

Algorithm 1: Removal of words within log line

```
Input  : df: log dataset, window: size of each window, percentage: deletion
         percentage
Output: df with words deleted
1  delete word index = [ ];
2  total rows = length of df;
3  iterations = total rows//window;
4  for i in range(iterations) do
5      start = i × window;
6      end = start + window;
7      number of lines to delete from = integer value of (window × percentage);
8      indices of line to delete words = randomly select (number of lines to
         delete from) unique indices from the range [start, end);
9      for j in indices of line to delete words do
10         curr sentence = 'EventTemplate' at index j in df;
11         word list = split curr sentence into list of words;
12         if length of word list > 1 then
13             num words to remove = random integer between 1 and length of
                 word list − 1;
14             words to remove = randomly select num words to remove indices
                 from the range [0, length of word list);
15             new word list = [word list[j] after deleting words to remove];
16             new sentence = join the words in new word list with space;
17             replace 'EventTemplate' at index j with new sentence;
18         else
19             continue
20 return df
```

Algorithm 2: Removal of log lines within log sequences(*df*, *window*, *percentage*)

```
Input  : df: log dataset, window: size of each window, percentage: deletion
         percentage
Output: df with lines deleted
1  delete line index = [ ];
2  total rows = length of  df;
3  iterations = total rows//window;
4  for i in range(iterations) do
5      start = i × window;
6      end = start + window;
7      number of lines to delete = integer value of (window × percentage);
8      indices of deleted lines = randomly select  (number of lines to delete)
         unique indices from the range  [start, end);
9      df = df exclude indices of deleted lines;
10 reset the index of df;
11 return df
```

Algorithm 3: Shuffle log lines of different timestamps(*df*, *window*, *percentage*)

```
Input  : df: log dataset, window: size of each window, percentage: deletion
         percentage
Output: df with lines shuffled
1  shuffle line index = [ ];
2  total rows = length of  df;
3  iterations = total rows//window;
4  for i in range(iterations) do
5      start = i × window;
6      end = start + window;
7      number of lines to shuffle = integer value of (window × percentage);
8      indices of shuffled lines = randomly select  (number of lines to shuffle)
         unique indices from the range  [start, end);
9      while  length of number of lines to shuffle >=2 do
10         randomly select  (a line k)  from the range  number of lines to shuffle;
11         randomly select  (a line l)  from the range  number of lines to shuffle;
12         shuffle the two lines k and l;
13 return df
```

3.4 Feature Extraction

When performing log anomaly detection in real-world scenarios, we usually run into noise simulated in Section 3.3. Regular running logs with such instability and noise will lose contextual information since logs that yield normal execution patterns are impaired or misplaced. We want to reduce the impact of instability and noise so the model can still extract valuable features. Through the application of the Attention-based Bi-LSTM mechanism, our model not only captures more contextual information from the log dataset but can also focus on the logs within these data that contributes more to the construction of normal executing logs instead of the noisy logs, thus exhibiting competitive performance regardless of noise or instability.

LSTM: Because our network aims to capture the normal executing log patterns so it can differentiate logs when they deviate from these normal patterns, it is essential to extract informative features. Inspired by the work DeepLog [8], we look at underlying patterns in log datasets as if they are grammatical rules in natural languages. Since grammatical rules present very strong sequential patterns, LSTM is adopted in our work as the basis for feature extraction due to the strong ability of Recurrent Neural Network [18] in handling sequential data compared to other deep neural network structures.

Bi-LSTM: As demonstrated above, instability and noise in log dataset will jeopardize valuable contextual information and damage the integrity of log sequences. Extracting features from log dataset under a given noisy environment is significantly more difficult than under closed-world settings. The only way to yield satisfying performance is to extract more information on a unit basis with the given dataset. Through the application of the Bidirectional LSTM module [36, 11, 43], we are able to capture bidirectional contextual information from both future and past hidden states, while traditional LSTM model could only use the information from the past time states.

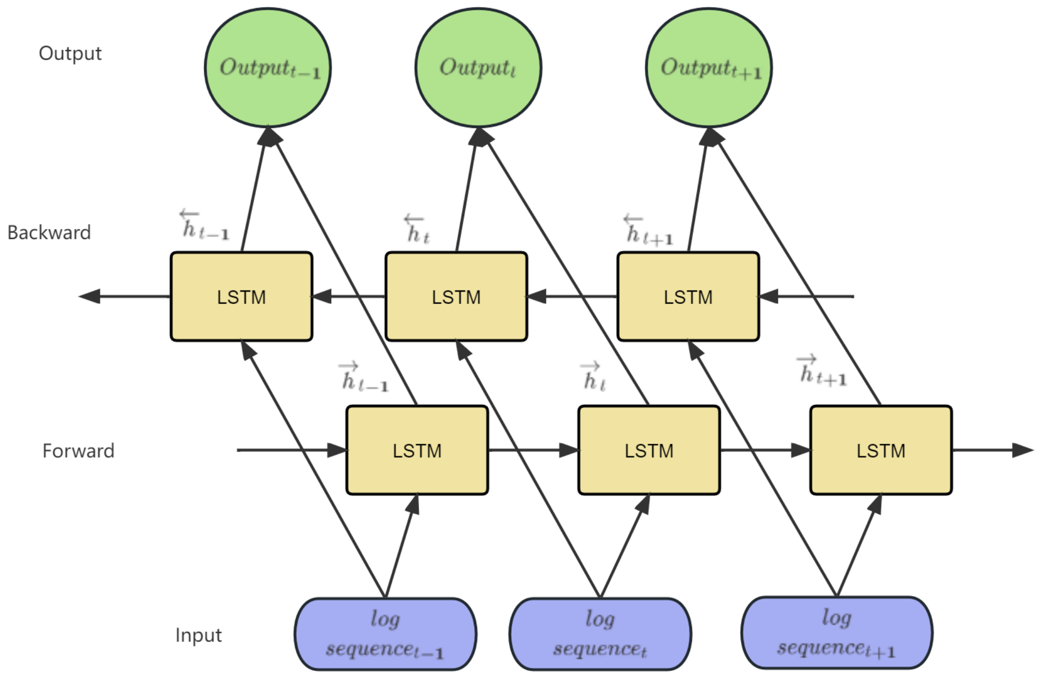


Figure 3.3: Bi-LSTM Architecture in LogRT

As shown in Figure 3.3, our Bi-LSTM module explores an extra direction of hidden state and contextual information to enhance its representation power. The forward process is able to carry dependencies and patterns from the past position to current sequence, while the backward process can communicate dependencies and patterns from the future sequence to the current position. With information from both past and future, our Bi-LSTM model is able to capture more intricate and long-range features compared to a regular LSTM model, thus enabling itself to provide a much more robust and expressive representation of the log patterns. This will aid the log anomaly detection process under the real-world settings with an improved performance.

Attention Mechanism: Aside from the Bi-LSTM module for extracting bidirectional contextual information, our model adds another mechanism into the feature extraction process: Attention. Attention was first introduced by Bahdanau et al. [2] and has become a robust and effective approach in the natural language processing toolbox. By viewing our log dataset as a language with underlying grammatical patterns, attention mechanism [38] leads to a performance boost when it comes to our task. The attention mechanism employed in our design has a structure as in shown Figure 3.4.

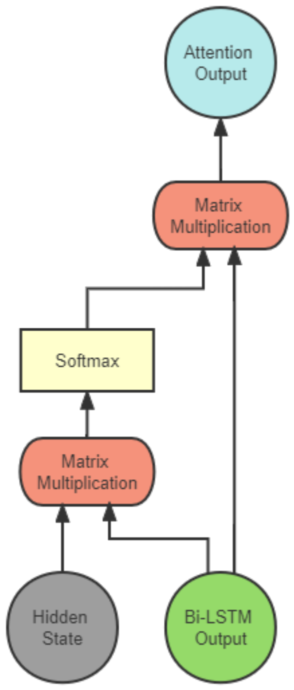


Figure 3.4: Architecture of Attention Mechanism in LogRT

Our attention mechanism's formula of calculating attention weights is as shown in Equation 3.4.1, in which **Q**, **K**, and **V** individually represents **Query**, **Key** and

Value respectively [38]. We treat the output generated from our Bi-LSTM module as the **Key** vector and **Value** vector, and the hidden state of the Bi-LSTM as the **Query** vector.

$$Attention(Q, K, V) = softmax(QK^T)V \tag{3.4.1}$$

Since the hidden state and output are generated by a Bi-LSTM module, we need to concatenate the hidden state of both directions as shown in Equation 3.4.2 and modify their dimensions first before feeding them into the attention mechanism.

$$h_{current} = concate[h_{forward}, h_{backward}] \tag{3.4.2}$$

Now with both directions of hidden state comprising a **Query** vector, we perform matrix multiplication with the output as demonstrated in Equation 3.4.3. This step is to obtain a measure of relevance and importance each **Key** vector has towards the **Query** vector. The output is the attention weight that directly decides how much attention should be allocated to each **Key**.

$$Attention\ \ Weight = h_{current} \times output_{Bi\text{-}LSTM} \tag{3.4.3}$$

Next, we run the obtained attention weight through a softmax layer such that we can normalize them into a probability distribution in the range of [0,1]. This process is shown in Equation 3.4.4 and produces a soft attention weight assigned to each input vector, indicating the contribution of each input vector towards the final attention output. This probability distribution allows the model to put more attention on the input log data that possess more importance and information, as higher weights indicate more importance and lower weights indicate less importance.

$$\textit{Softmax Attention} = softmax(\textit{Attention Weight}) \tag{3.4.4}$$

With the soft attention weight as the **Query** vector to guide the decision on which part of the log sequence to focus on, we multiply this weight with the output of the Bi-LSTM module as the **Value** vector. We arrive at the weighted sum of the **Value** vector as shown in Equation 3.4.5. Even if the log dataset has lost contextual information and valuable patterns due to noise and instability, with the attention weight, our model can still focus on the important log data regardless of the noise and instability issue and yield competitive results.

$$\textit{Attention Output} = \textit{Softmax Attention} \times output_{\textit{Bi-LSTM}} \tag{3.4.5}$$

3.5 Domain Adaptation

With the robust feature extraction modules, we can now capture important contextual information from noisy and unstable log dataset. Because we aim to provide a robust log anomaly detection solution under cross-system settings, we need to have the ability of transferring the learned log patterns from a source dataset to a target dataset. Among many applications of transfer learning in the field of log anomaly detection, domain adaptation has shown outstanding performance. Inspired by Han et al. [13], we apply domain adversarial training in our work.

Gradient Reversal Layer: This is a layer that follows the attention output directly. Since we want to learn the normal executing patterns of a source system log dataset and then apply them to the target system log dataset, we need the model to extract features that can represent both source and target domains. However, as the datasets might not share the same distribution, the patterns demonstrated during the feature extraction stage might differ as well, leading to low performance of the model. The employment of a gradient reversal layer [9] can force the model to enhance the pattern similarity learned between those two datasets through reversing the sign of the gradient and thus deliver good performance.

As we are using the same feature extractor for both datasets, the GRL reverses the gradient of our feature extractor and makes the gradient computed within the extractor to have opposite directions during the update of parameters. As a result, the model is compelled to learn features that can optimally represent both the source and target log datasets, consequently achieving good generalization and performance in log anomaly detection on the target system.

Domain Adversarial Training: After extracting the feature and generalizing them on both source and target domains, we use domain adversarial training inspired by Han et al. [13] to map the feature representations of the two domains into the same one as we demonstrate in Figure 3.5. This is achieved through utilizing the minmax optimization function [10] shown in Equation 3.5.1, in which D is the discriminator differentiating if data is from source or target domain. The feature extractor Attention Bi-LSTM module acts as a generator G. Through minimizing the loss of the generator G in producing samples similar to the ground-truth log dataset, the Attention Bi-LSTM module gains the ability to effectively simulate samples resembling the ones from the source domain. Then, through maximizing the loss of the discriminator in differentiating data from source and target domains, we force the discriminator to discriminate effectively between results from the Attention Bi-LSTM model and the target domain. After this training process, the Attention Bi-LSTM module will learn shared patterns between source and target domains and facilitate the transfer of log patterns from source to target domain.

$$\begin{aligned} L_{adv} = \min_{G}\max_{D}(E_{X^S \sim P_{Source}}[logD(G(X^S))] \\ + E_{X^T \sim P_{Target}}[log(1 - D(G(X^T)))] \end{aligned} \tag{3.5.1}$$

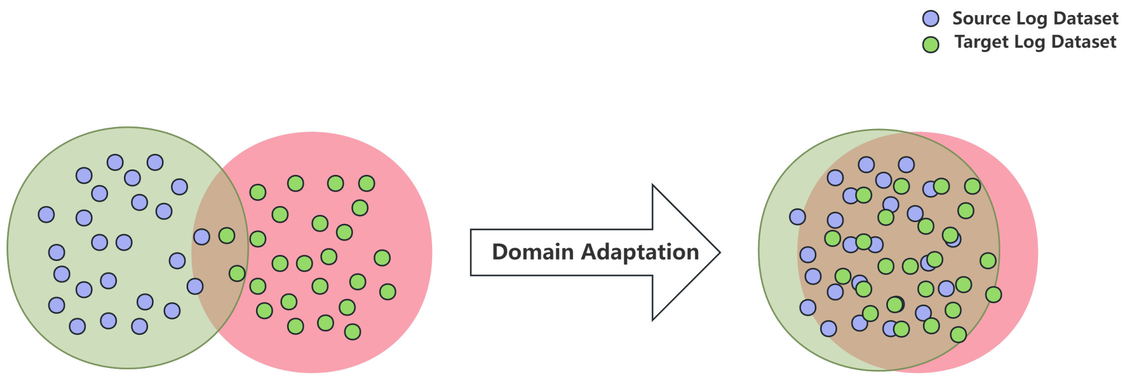


Figure 3.5: Data distribution of both source and target domains after domain adaptation

3.6 Anomaly Detection

As the feature extractor has captured valuable information to support the model in formulating normal executing patterns, the model acquires through the domain adaptation process the ability to map both source and target system domains into the same distribution, which is demonstrated in Figure 3.5.

Our training datasets consist of only normal execution logs from both source and target systems, and we need our model to perform classifications of normal and anomalous logs based only on this type of data. As a result, this task is a one-class classification [35, 37, 17, 34]. In this work, we apply a predominant one-class classification method that has been proven to be effective in log anomaly detection by many other works [12, 13], the Deep Support Vector Data Description [33].

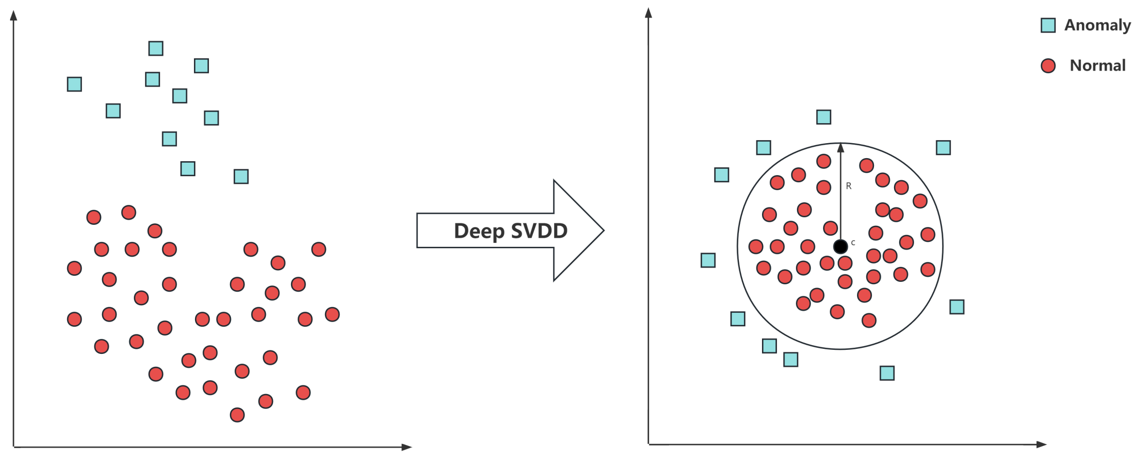


Figure 3.6: Data distribution of normal and anomalous logs after applying Deep SVDD

By applying Deep SVDD [33], we aim to include all the normal log points inside a single hypersphere while excluding the anomaly data points outside of the hypersphere, in which the center c is derived from the normal log data representations and the volume of the sphere is minimized as shown in Figure 3.6. The minimization of the hypersphere forces the model to extract common factors of the log sequences. The construction of the model is delicate since we need to prevent the hypersphere from collapsing, which happens under many circumstances according to [33]. With a log data point x, the distance between x and the center c is given by Equation

3.6.1. Here d is the distance between log data point x and center c, L is the model representation of LogRT for the given point x, and W is the weight within the model.

$$d = \|L(x;W) - c\|^2 \tag{3.6.1}$$

In the testing stage where the dataset consists of both normal and anomalous log data, the normal logs will be confined within the hypersphere while the anomalous log data will have a further distance towards the center and lie outside of the boundary.

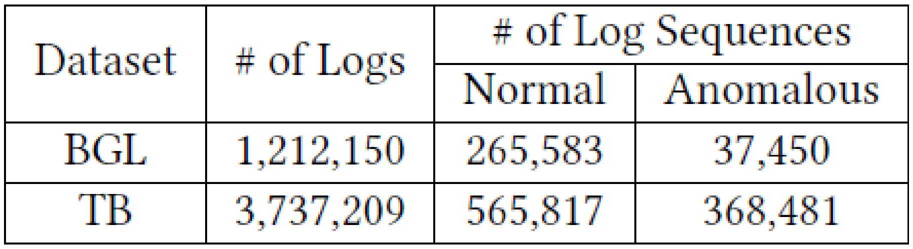
Chapter 4

Experiments

In this section, we discuss the experimental details of LogRT. We start by introducing the datasets we experiment with, Thunderbird [31] and BlueGene/L [24] datasets. Then, we move on to evaluation metrics of our study and simulation of the noisy and unstable logs during the experiment. Our comparison and ablation studies demonstrate the effectiveness of LogRT, with LogRT outperforming state-of-the-art network.

4.1 Dataset

The datasets we use to test our work are Thunderbird and BlueGene/L supercomputer system datasets. The Thunderbird log dataset was obtained from a Thunderbird supercomputer at the Sandia National Laboratory in Albuquerque [31]. It has 9,024 processors and a memory capacity of 27,072 GB. The BlueGene/L log dataset was obtained from a BlueGene/L supercomputer at the Lawrence Livermore National Laboratory in Livermore, California [24]. It has 131,072 processors and a memory capacity of 32,768 GB. These two datasets are both from Loghub, a cornerstone work by He et al. [16] in which they collected and organized log datasets sourced from a variety of systems for future studies in the field of log data analysis. The work of Han et al. [13] provided statistics regarding these two log datasets as shown in Figure 4.1.



Dataset	# of Logs	# of Log Sequences	
		Normal	Anomalous
BGL	1,212,150	265,583	37,450
TB	3,737,209	565,817	368,481

Figure 4.1: Number of normal and anomalous log sequences within Thunderbird and BlueGene/L datasets (Image sourced from [13])

We can tell from Figure 4.1 that both of these datasets consist mainly of normal log sequences together with a small proportion of anomalous log sequences. Their distribution of normal and anomalous log data fits our research goal as our training procedure is carried out using only normal log sequences.

4.2 Evaluation Metrics

The metrics we utilize to evaluate the performance of LogRT are F1-score and AUC score. There are several other alternatives that we could have chosen, specifically precision and accuracy. The reason we did not use them is that when performing log anomaly detection, log dataset usually exhibits an imbalance between the normal sequences and the anomalous sequences, with the former constituting the main part of the dataset. This imbalance, illutrated in Figure 4.2, leads to less informative results in terms of accuracy and precision produced by deep learning models, because misclassification of anomalous logs will have minimal impact on the prediction result compared to the majority of normal sequences that are correctly classified.

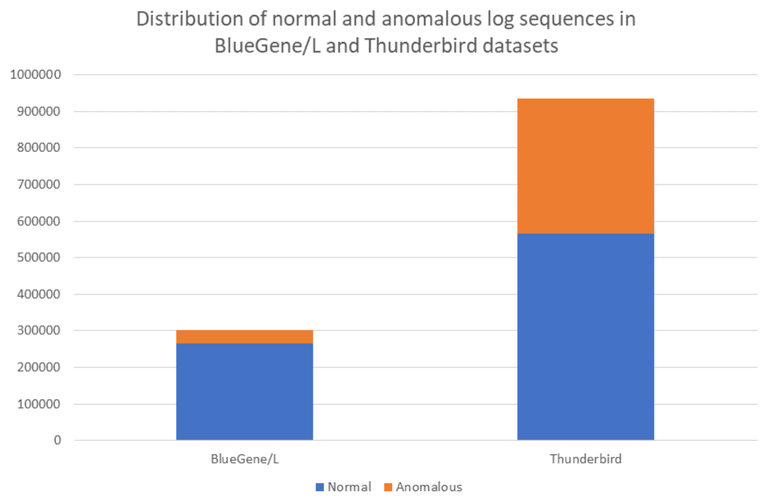


Figure 4.2: Distribution of normal and anomalous log sequences in BlueGene/L and Thunderbird datasets

On the other hand, we can see from Equation 4.2.1 that F1-score takes into account both recall rate and precision, in which the recall rate can provide us with information on the proportion of normal and anomalous log sequences that are correctly classified into the classes they belong, thus giving a much more comprehensive insight into the performance of our model.

$$F_1 = 2 \cdot \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{4.2.1}$$

AUC score measures the area under the Receiver Operating Characteristic curve, which is drawn with true positive rate as the y axis and false positive rate as the x axis. Equations 4.2.2 and 4.2.3 demonstrate the components of AUC score and provide a holistic assessment of the model even with an imbalanced distribution of datasets.

$$\text{True Positive Rate} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{4.2.2}$$

$$\text{False Positive Rate} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}} \tag{4.2.3}$$

4.3 Noise Simulation

Since the datasets we use do not contain the specific noise we want to investigate as in most real-world scenarios, we choose to develop simulation algorithms to simulate noisy environments with the relevant pseudo code given in Section 3.3. Three types of noise are simulated, including: 1. Removal of words within log lines, 2. Removal of log lines within log sequence, 3. Shuffle log lines of different timestamps. They are then injected into the original log datasets of Thunderbird and BlueGene/L with designated ratios of 5%, 10%, 15%, and 20%. The setup of these percentages and noise is inspired by the work LogRobust [43].

4.4 Comparison

We demonstrate LogRT's competitive performance through the comparison between LogRT and LogTAD [13], as LogTAD utilizes a LSTM model as feature extractor and addresses the cold-starting issue when handling newly onboarded systems with domain adversarial training.

4.4.1 Training Details

To make a fair comparison, we train LogRT and LogTAD on the same Thunderbird and BlueGene/L datasets with the same ratios of simulated noise. We train both models for 100 epochs with a training size of 100,000 and a testing size of 1,000. By varying the learning rate between 0.001 to 0.0005, we settle at 0.001 as it provides the best trade off between convergence rate and model performance. Same as LogTAD, we use a batch size of 1024 and a weight decay of 1e-6 for the Adam optimizer.

4.4.2 Results

Table 4.1 shows the result of LogRT model compared to the LogTAD model on the task of log anomaly detection transferring from BlueGene/L to Thunderbird. Experiments are carried out on the various ratios of noise simulated respectively within Thunderbird and BlueGene/L log datasets. A noteworthy point is, because these noises are not simulated cumulatively, the model performance under increasing ratios of the same type of noise might not degenerate monotonically. This is due to the randomness in simulation of noise, which we will further discuss in Chapter 5.

We have also conducted our experiment on datasets with a mixture of the simulated noise. We simulated different types of noise with the same ratios on both Thunderbird and BlueGene/L datasets. Table 4.2 shows the comparison result of LogRT with the LogTAD model when performing on these mixed ratio of noise. It can be seen that LogRT outperforms LogTAD when the noise ratio is above 5%.

According to both tables, our model LogRT presents better performance in most cases compared to the state-of-the-art LogTAD model, thus confirming the efficacy of our proposed robust transferring learning-based log anomaly detection approach.

Performance(AUC/F1) Models / Dataset	LogTAD	LogRT
Original	0.79716/0.75197	**0.80945/0.76389**
Remove words 5%	0.77165/0.73145	**0.80692/0.76485**
Remove words 10%	**0.78743/0.74961**	0.78506/0.74287
Remove words 15%	**0.79301/0.75939**	0.78703/0.75414
Remove words 20%	0.79020/0.75963	**0.80180/0.77086**
Delete lines 5%	0.80277/0.76135	**0.81658/0.77477**
Delete lines 10%	**0.79706/0.76005**	0.79639/0.75980
Delete lines 15%	0.78877/0.75647	**0.79146/0.75925**
Delete lines 20%	0.78673/0.75782	**0.79019/0.76044**
Shuffle lines 5%	0.78809/0.75112	**0.81343/0.77565**
Shuffle lines 10%	0.77379/0.74412	**0.78421/0.75425**
Shuffle lines 15%	0.75508/0.73219	**0.77977/0.75545**
Shuffle lines 20%	0.77499/0.75671	**0.78276/0.76575**

Table 4.1: Result of LogRT and LogTAD models transferring from BlueGene/L to Thunderbird dataset with three types of noise and 4 noise ratios

Performance(AUC/F1) \ Models / Dataset	LogTAD	LogRT
5%of mixed noise	**0.79008/0.76043**	0.77900/0.74904
10%of mixed noise	0.76251/0.74576	**0.78015/0.76230**
15%of mixed noise	0.72926/0.72155	**0.77543/0.76674**
20%of mixed noise	0.76348/0.76287	**0.77872/0.77479**

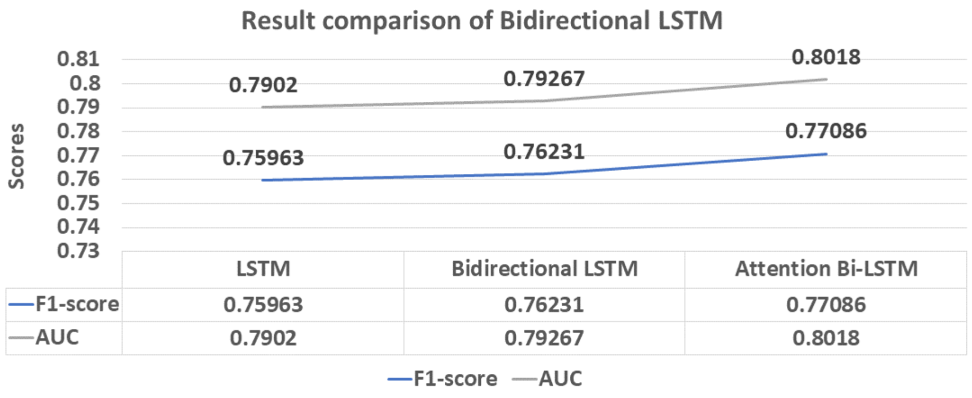
Table 4.2: Result of LogRT and LogTAD models transferring from BlueGene/L to Thunderbird dataset with a mixture of the three types of simulated noise

4.5 Ablation study

In this section, we conduct ablation studies regarding the factors that contribute to the competitive performance of our proposed method.

4.5.1 Bidirectional LSTM

We verify the contribution of the Bi-LSTM mechanism to an increase in performance of our model by testing three models on the same dataset. The first model is a regular LogRT model, the second model is a LogRT model without the bidirectional LSTM mechanism, and the third model is the LogTAD model using only traditional LSTM. The dataset being used are BlueGene/L and Thunderbird with 20% removal of words. Figure 4.3 shows the result.



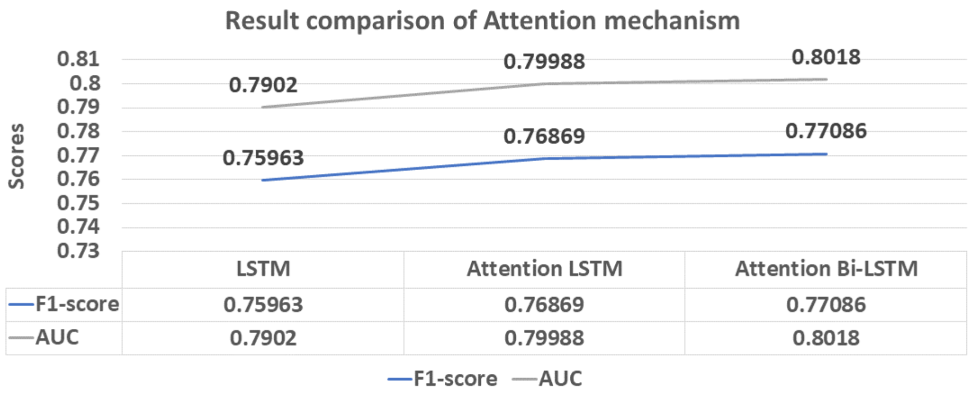
	LSTM	Bidirectional LSTM	Attention Bi-LSTM
F1-score	0.75963	0.76231	0.77086
AUC	0.7902	0.79267	0.8018

Figure 4.3: Result comparison of Bidirectional LSTM with other models

As demonstrated, the contribution of bidirectional LSTM mechanism to the competitive performance of our model is significant and certain.

4.5.2 Attention Mechanism

In order to verify the effectiveness of Attention mechanism in our work, we compare a regular LogRT model, a LogRT model without the Attention mechanism, and a LogTAD model using only traditional LSTM together on the BlueGene/L and Thunderbird datasets with 20% removal of words. Comparison result is shown in Figure 4.4.



	LSTM	Attention LSTM	Attention Bi-LSTM
F1-score	0.75963	0.76869	0.77086
AUC	0.7902	0.79988	0.8018

Figure 4.4: Result comparison of Attention mechanism with other models

As illustrated, our Attention mechanism contributes significantly to the effectiveness of our model LogRT.

Chapter 5

Future Improvements

In this section, we discuss some of the potential improvements that can be made to the design of our work.

5.1 Dataset Selection

The creation of a well-labelled log dataset is usually difficult, as for a log dataset to be informative, the size of the dataset must be large enough. When we first explored the field of log anomaly detection, we noticed that nearly all the existing works on log analysis obtained their dataset from the work of Loghub [16], in which He et al. collected and organized the log dataset that had been generated by other labs, institutions, and corporations. Only a small portion of them is labelled normal and anomalous as shown in Figure 5.1.

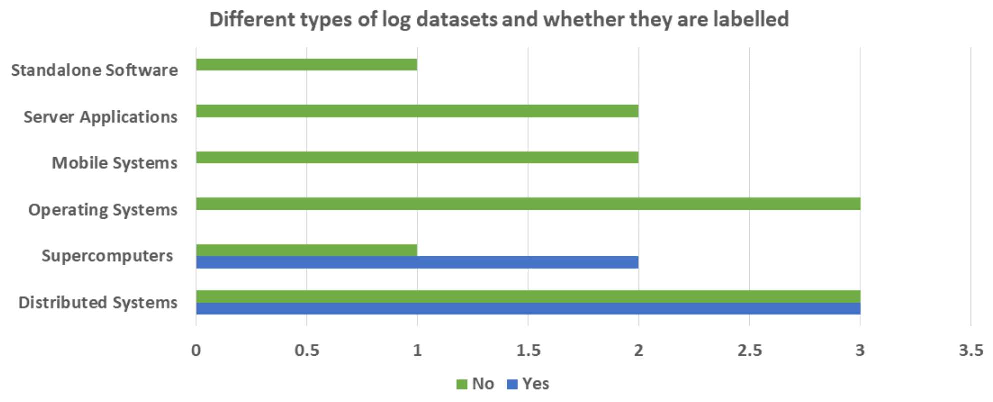


Figure 5.1: Proportion of different type of log datasets in Loghub that are labelled

Some of these log datasets are more than 10 years of age. Log templates of these system have been updated many times over the past decade, rendering a diminishing utility in providing information for log analysis.

We tried generating our own log dataset, but due to the large amount of work and resources required on domain knowledge, we decided to use existing datasets eventually. For a more systemtaic investigation of the the log anomaly detection task, it is favorable to construct datasets that are up to date and customized.

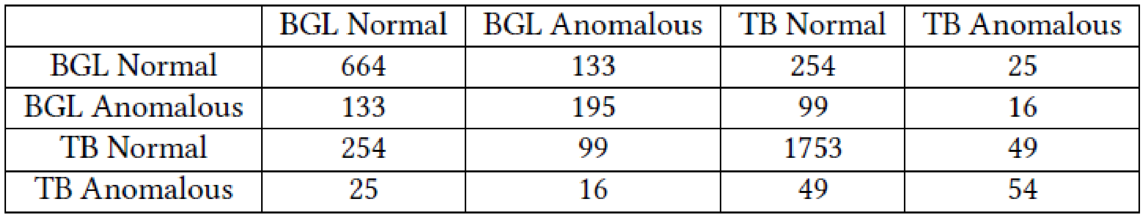
5.2 Injection of Noise

As mentioned in Section 4.4, the noise simulated within the BlueGene/L and Thunderbird datasets is not injected cumulatively. The reason we did not adopt the cumulative injection approach is because it is error prone. Ensuring the simulations we have already performed not to be affected by the new simulations we conduct is much more difficult than we first imagined, specifically due to the size of the dataset where BlueGene/L has 1,212,150 lines and Thunderbird has 3,737,209 lines [13].

However, injecting noise cumulatively can potentially provide us with a more straightforward demonstration of the efficacy of LogRT, in which the performance of model under the same type of noise should degrade as the proportion of noise increases. In the future, it is desirable to have a cumulative noise simulation algorithm.

5.3 Ability of Information Transfer

To demonstrate the robustness of our model in a cross-system setting, we have investigated both directions of transferring from BlueGene/L to Thunderbird dataset and from Thunderbird to BlueGene/L dataset. The result from BlueGene/L to Thunderbird has been demonstrated in Section 4.4, in which we observe LogRT outperforming LogTAD in most cases. Normal log sequences of the Thunderbird dataset have 1,753 unique words while the BlueGene/L dataset only has 664 unique words as illustrated in Figure 5.2. Transferring from Thunderbird to BlueGene/L should be an easier task with the Thunderbird log dataset as the source domain providing a much larger feature space and also volume of training samples according to Figure 4.1 and Figure 5.2. However, the performance of LogRT when transferring from Thunderbird to BlueGene/L is not as good as the performance of LogTAD.



	BGL Normal	BGL Anomalous	TB Normal	TB Anomalous
BGL Normal	664	133	254	25
BGL Anomalous	133	195	99	16
TB Normal	254	99	1753	49
TB Anomalous	25	16	49	54

Figure 5.2: Amount of shared words between BlueGene/L and Thunderbird log datasets(Image source from [13])

It is counter-intuitive that LogRT can outperform SOTA models when transferring from a small domain to a large domain, which is a more difficult task compared to transferring from a large domain to a small domain. Another thing to note is that both LogRT and LogTAD exhibit a decrease of around 15% in F1-score when transferring from BlueGene/L to Thunderbird compared to the other direction. This might be because LogRT's Attention-based Bi-LSTM mechanism has a stronger ability in focusing on informative word features, thus diffusing its attention when learning from a larger domain and leading to a decrease in model performance. We think conducting feature selection on the Thunderbird dataset to extract features that are most common between Thunderbird and BlueGene/L might help increase the performance of model.

With that being said, LogRT still outperforms on this more difficult task than LogTAD, further verifying the efficiency of our work.

Chapter 6

Conclusion

We have proposed a robust approach named LogRT for cross-system log anomaly detection. Utilizing an Attention-based Bi-LSTM mechanism, our model is able to reduce the impact of noise within the log datasets and extract valuable features for the anomaly detection process. We also have provided a systematic method for simulating noise within log datasets which we hope can be of help to other researchers in the future. With thorough experimental results and ablation studies, we have proven the effectiveness of LogRT.

Bibliography

[1] N. Alshuqayran, N. Ali, and R. Evans. A systematic mapping study in microservice architecture. In *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 44–51. IEEE, 2016.

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

[3] C. Bertero, M. Roy, C. Sauvanaud, and G. Trédan. Experience report: Log mining using natural language processing and application to anomaly detection. In *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*, pages 351–360. IEEE, 2017.

[4] L. Chen. Continuous delivery: Huge benefits, but challenges too. *IEEE software*, 32(2):50–54, 2015.

[5] R. Chen, S. Zhang, D. Li, Y. Zhang, F. Guo, W. Meng, D. Pei, Y. Zhang, X. Chen, and Y. Liu. Logtransfer: Cross-system log anomaly detection for software systems with transfer learning. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, pages 37–47. IEEE, 2020.

[6] Z. Chen, J. Liu, W. Gu, Y. Su, and M. R. Lyu. Experience report: Deep

learning-based system log analysis for anomaly detection. *arXiv preprint arXiv:2107.05908*, 2021.

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

[8] 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*, pages 1285–1298, 2017.

[9] Y. Ganin and V. Lempitsky. Unsupervised domain adaptation by backpropagation. In *International conference on machine learning*, pages 1180–1189. PMLR, 2015.

[10] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[11] A. Graves, S. Fernández, and J. Schmidhuber. Bidirectional lstm networks for improved phoneme classification and recognition. In *Artificial Neural Networks: Formal Models and Their Applications–ICANN 2005: 15th International Conference, Warsaw, Poland, September 11-15, 2005. Proceedings, Part II 15*, pages 799–804. Springer, 2005.

[12] H. Guo, S. Yuan, and X. Wu. Logbert: Log anomaly detection via bert. In *2021*

international joint conference on neural networks (IJCNN), pages 1–8. IEEE, 2021.

[13] X. Han and S. Yuan. Unsupervised cross-system log anomaly detection via domain adaptation. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 3068–3072, 2021.

[14] 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*, 15(6):931–944, 2017.

[15] 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)*, pages 33–40. IEEE, 2017.

[16] S. He, J. Zhu, P. He, and M. R. Lyu. Loghub: a large collection of system log datasets towards automated log analytics. *arXiv preprint arXiv:2008.06448*, 2020.

[17] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[18] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[19] Y. Huo, C. Lee, Y. Su, S. Shan, J. Liu, and M. Lyu. Evlog: Evolving log analyzer for anomalous logs identification. *arXiv preprint arXiv:2306.01509*, 2023.

[20] T. Jia, L. Yang, P. Chen, Y. Li, F. Meng, and J. Xu. Logsed: Anomaly diagnosis through mining time-weighted control flow graph in logs. In *2017 IEEE 10th*

International Conference on Cloud Computing (CLOUD), pages 447–455. IEEE, 2017.

[21] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[22] S. Kabinna, C.-P. Bezemer, W. Shang, M. D. Syer, and A. E. Hassan. Examining the stability of logging statements. *Empirical Software Engineering*, 23:290–333, 2018.

[23] V.-H. Le and H. Zhang. Log-based anomaly detection with deep learning: How far are we? In *Proceedings of the 44th International Conference on Software Engineering*, pages 1356–1367, 2022.

[24] Y. Liang, Y. Zhang, A. Sivasubramaniam, R. K. Sahoo, J. Moreira, and M. Gupta. Filtering failure logs for a bluegene/l prototype. In *2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 476–485. IEEE, 2005.

[25] 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)*, pages 583–588. IEEE, 2007.

[26] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen. Log clustering based problem identification for online service systems. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 102–111, 2016.

[27] 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*, pages 1–14, 2010.

[28] 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)*, pages 151–158. IEEE, 2018.

[29] 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*, volume 19, pages 4739–4745, 2019.

[30] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[31] 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)*, pages 575–584. IEEE, 2007.

[32] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[33] 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*, pages 4393–4402. PMLR, 2018.

[34] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In *Information Processing in Medical Imaging: 25th International Conference, IPMI 2017, Boone, NC, USA, June 25-30, 2017, Proceedings*, pages 146–157. Springer, 2017.

[35] B. Schölkopf, R. C. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt. Support vector method for novelty detection. *Advances in neural information processing systems*, 12, 1999.

[36] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

[37] D. M. Tax and R. P. Duin. Support vector data description. *Machine learning*, 54:45–66, 2004.

[38] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[39] Y. Xie and K. Yang. Domain adaptive log anomaly prediction for hadoop system. *IEEE Internet of Things Journal*, 9(20):20778–20787, 2022.

[40] W. Xu. *System problem detection by mining console logs*. University of California, Berkeley, 2010.

[41] W. Xu, L. Huang, A. Fox, D. A. Patterson, and M. I. Jordan. Mining console logs for large-scale system problem detection. *SysML*, 8:4–4, 2008.

[42] 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*, pages 117–132, 2009.

[43] 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*, pages 807–817, 2019.

[44] J. Zhu, P. He, Q. Fu, H. Zhang, M. R. Lyu, and D. Zhang. Learning to log: Helping developers make informed logging decisions. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 415–425. IEEE, 2015.

[45] 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)*, pages 121–130. IEEE, 2019.